Hasso Plattner Institute for Digital Engineering
at the University of Potsdam
Enterprise Platform and Integration Concepts Research Group
Prof. Dr. mult. h.c. Hasso Plattner

Memory-Efficient Data Management for Spatio-Temporal Applications

Workload-Driven Fine-Grained Configuration Optimization for Storing Spatio-Temporal Data in Columnar In-Memory Databases

Dissertation
submitted in partial fulfillment
of the requirements for the academic degree of
Doctor of Natural Sciences
(Dr. rer. nat.)

to the
Digital Engineering Faculty
at the University of Potsdam

by
Keven Richly, M.Sc.

Potsdam, 31st March, 2024

Supervisors

Prof. Dr. mult. h.c. Hasso Plattner
Hasso Plattner Institute (University of Potsdam)

Prof. Dr. Eleni Tzirita Zacharatou
IT University of Copenhagen

Prof. Dr. Michael Grossniklaus
University of Konstanz

Published online on the
Publication Server of the University of Potsdam:
https://doi.org/10.25932/publishup-63547
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-635473

Abstract

The wide distribution of location-acquisition technologies means that large volumes of spatio-temporal data are continuously being accumulated. Positioning systems such as GPS enable the tracking of various moving objects' trajectories, which are usually represented by a chronologically ordered sequence of observed locations. The analysis of movement patterns based on detailed positional information creates opportunities for applications that can improve business decisions and processes in a broad spectrum of industries (e.g., transportation, traffic control, or medicine). Due to the large data volumes generated in these applications, the cost-efficient storage of spatio-temporal data is desirable, especially when in-memory database systems are used to achieve interactive performance requirements.

To efficiently utilize the available DRAM capacities, modern database systems support various tuning possibilities to reduce the memory footprint (e.g., data compression) or increase performance (e.g., additional indexes structures). By considering horizontal data partitioning, we can independently apply different tuning options on a fine-grained level. However, the selection of cost and performance-balancing configurations is challenging, due to the vast number of possible setups consisting of mutually dependent individual decisions.

In this thesis, we introduce multiple approaches to improve spatio-temporal data management by automatically optimizing diverse tuning options for the application-specific access patterns and data characteristics. Our contributions are as follows: (1) We introduce a novel approach to determine fine-grained table configurations for spatio-temporal workloads. Our linear programming (LP) approach jointly optimizes the (i) data compression, (ii) ordering, (iii) indexing, and (iv) tiering. We propose different models which address cost dependencies at different levels of accuracy to compute optimized tuning configurations for a given workload, memory budgets, and data characteristics. To yield maintainable and robust configurations, we further extend our LP-based approach to incorporate reconfiguration costs as well as optimizations for multiple potential workload scenarios. (2) To optimize the storage layout of timestamps in columnar databases, we present a heuristic approach for the workload-driven combined selection of a data layout and compression scheme. By considering attribute decomposition strategies, we are able to apply application-specific optimizations that reduce the memory footprint and improve performance. (3) We introduce an approach that leverages past trajectory data to improve the dispatch pro-

cesses of transportation network companies. Based on location probabilities, we developed risk-averse dispatch strategies that reduce critical delays. (4) Finally, we used the use case of a transportation network company to evaluate our database optimizations on a real-world dataset. We demonstrate that workload-driven fine-grained optimizations allow us to reduce the memory footprint (up to 71% by equal performance) or increase the performance (up to 90% by equal memory size) compared to established rule-based heuristics.

Individually, our contributions provide novel approaches to the current challenges in spatio-temporal data mining and database research. Combining them allows in-memory databases to store and process spatio-temporal data more cost-efficiently.

Zusammenfassung

Durch die starke Verbreitung von Systemen zur Positionsbestimmung werden fortlaufend große Mengen an Bewegungsdaten mit einem räumlichen und zeitlichen Bezug gesammelt. Ortungssysteme wie GPS ermöglichen, die Bewegungen verschiedener Objekte (z. B. Personen oder Fahrzeuge) nachzuverfolgen. Diese werden in der Regel durch eine chronologisch geordnete Abfolge beobachteter Aufenthaltsorte repräsentiert. Die Analyse von Bewegungsmustern auf der Grundlage detaillierter Positionsinformationen schafft in unterschiedlichsten Branchen (z. B. Transportwesen, Verkehrssteuerung oder Medizin) die Möglichkeit Geschäftsentscheidungen und -prozesse zu verbessern. Aufgrund der großen Datenmengen, die bei diesen Anwendungen auftreten, stellt die kosteneffiziente Speicherung von Bewegungsdaten eine Herausforderung dar. Dies ist insbesondere der Fall, wenn Hauptspeicherdatenbanken zur Speicherung eingesetzt werden, um die Anforderungen bezüglich interaktiver Antwortzeiten zu erfüllen.

Um die verfügbaren Speicherkapazitäten effizient zu nutzen, unterstützen moderne Datenbanksysteme verschiedene Optimierungsmöglichkeiten, um den Speicherbedarf zu reduzieren (z. B. durch Datenkomprimierung) oder die Performance zu erhöhen (z. B. durch Indexstrukturen). Dabei ermöglicht eine horizontale Partitionierung der Daten, dass unabhängig voneinander verschiedene Optimierungen feingranular auf einzelnen Bereichen der Daten angewendet werden können. Die Auswahl von Konfigurationen, die sowohl die Kosten als auch Leistungsanforderungen berücksichtigen, ist jedoch aufgrund der großen Anzahl möglicher Kombinationen – die aus voneinander abhängigen Einzelentscheidungen bestehen – komplex.

In dieser Dissertation präsentieren wir mehrere Ansätze zur Verbesserung der Datenverwaltung, indem wir die Auswahl verschiedener Datenbankoptimierungen automatisch für die anwendungsspezifischen Zugriffsmuster und Dateneigenschaften anpassen. Diesbezüglich leistet die vorliegende Dissertation die folgenden Beiträge: (1) Wir stellen einen neuen Ansatz vor, um feingranulare Tabellenkonfigurationen für räumlich-zeitliche Workloads zu bestimmen. In diesem Zusammenhang optimiert unser Linear Programming (LP) Ansatz gemeinsam (i) die Datenkompression, (ii) die Sortierung, (iii) die Indizierung und (iv) die Datenplatzierung. Hierzu schlagen wir verschiedene Modelle mit unterschiedlichen Kostenabhängigkeiten vor, um optimierte Konfigurationen für einen gegebenen Workload, ein Speicherbudget und die vorliegenden Dateneigen-

schaften zu berechnen. Durch die Erweiterung des LP-basierten Ansatzes zur Berücksichtigung von Modifikationskosten und verschiedener potentieller Workloads ist es möglich, die Wartbarkeit und Robustheit der bestimmten Tabellenkonfiguration zu erhöhen. (2) Um die Speicherung von Timestamps in spalten-orientierten Datenbanken zu optimieren, stellen wir einen heuristischen Ansatz für die kombinierte Auswahl eines Speicherlayouts und eines Kompressionsschemas vor. Zudem sind wir durch die Berücksichtigung von Strategien zur Aufteilung von Attributen in der Lage, anwendungsspezifische Optimierungen anzuwenden, die den Speicherbedarf reduzieren und die Performance verbessern. (3) Wir stellen einen Ansatz vor, der in der Vergangenheit beobachtete Bewegungsmuster nutzt, um die Zuweisungsprozesse von Vermittlungsdiensten zur Personenbeförderung zu verbessern. Auf der Grundlage von Standortwahrscheinlichkeiten haben wir verschiedene Strategien für die Vergabe von Fahraufträgen an Fahrer entwickelt, die kritische Verspätungen reduzieren. (4) Abschließend haben wir unsere Datenbankoptimierungen anhand eines realen Datensatzes eines Transportdienstleisters evaluiert. In diesem Zusammenhang zeigen wir, dass wir durch feingranulare workload-basierte Optimierungen den Speicherbedarf (um bis zu 71% bei vergleichbarer Performance) reduzieren oder die Performance (um bis zu 90% bei gleichem Speicherverbrauch) im Vergleich zu regelbasierten Heuristiken verbessern können.

Die einzelnen Beiträge stellen neuartige Ansätze für aktuelle Herausforderungen im Bereich des Data Mining und der Datenbankforschung dar. In Kombination ermöglichen sie eine kosteneffizientere Speicherung und Verarbeitung von Bewegungsdaten in Hauptspeicherdatenbanken.

Acknowledgements

First of all, I would like to thank my supervisor Professor Hasso Plattner. I greatly benefited from his extraordinary commitment to supporting academic education in Germany during my bachelor's and master's studies, as well as being a Ph.D. student at the Hasso Plattner Institute. I am genuinely grateful for my time at the Enterprise Platform and Integration Concepts research group, where I gained valuable experience, followed my research interests, and became part of an outstanding team. Professor Hasso Plattner and his chair representatives, Dr. Michael Perscheid, Dr. Matthias Uflacker, and Dr. Jürgen Müller provided helpful guidance and support for my research over the last years.

Furthermore, I would like to thank my colleagues for all the fruitful discussions and invaluable feedback. Most notably, my thanks go to Dr. Rainer Schlosser, Martin Boissier, Christopher Hagedorn, Stefan Halfpap, Jan Kossmann, Johannes Hügle, Dr. Ralf Teusner and Dr. Markus Dreseler. I am glad to have found excellent collaborators and friends in our research group. Thank you for making the Ph.D. journey a pleasant time.

Also, I wish to thank all students who worked under my supervision on different research projects or supported me as teaching assistants in various lectures. Moreover, I would like to thank all co-authors for the productive collaboration. On an organizational level, I also thank Marilena Davis and Matthias Herzog for keeping everything together from an administrative and hardware perspective.

Finally, I want to thank my parents, Birgit and Frank, for their enduring and unconditional support as well as for sparking my interest in technology.

Contents

1

Introduction

Through the wide distribution of location-acquisition technologies, vast amounts of spatio-temporal data are continuously accumulated. Positioning systems such as the Global Positioning System (GPS) enable the tracking of a broad spectrum of moving objects ranging from vehicles and persons to natural phenomena [79, 341, 395]. The data generated by these systems are referred to as spatio-temporal or trajectory data and reflect the traces of moving objects [319]. Spatio-temporal data enable the analysis of movement patterns, which are increasingly used in various applications (e.g., transportation and traffic optimizations)[113, 320].

The trajectory of a moving object is usually represented by a chronologically ordered sequence of observed locations. In this context, each observed location that a moving object has passed is described by a position (e.g., a multi-dimensional coordinate) in a geographic reference system and a temporal component (e.g., a timestamp)[323]. Insights based on the analysis of spatio-temporal data can significantly impact business decisions in various industries (e.g., transport and logistics industry). However, storing and processing large amounts of trajectory data with interactive response times is challenging [109, 355, 367, 377].

1.1 Business Impact of Spatio-Temporal Data Mining Applications

In recent years, the market for devices with built-in position tracking capabilities has grown considerably [379]. Whereas, in the past, dedicated tracking equipment was required to capture the traces of moving objects, nowadays, a broad spectrum of devices (e.g., mobile phones) include positioning services, such as GPS. Furthermore, there are various technologies besides GPS which are applied to produce trajectory data, including radio-frequency identification (RFID) sensors, location estimation via 802.11, infrared and ultrasonic systems, and Global System for Mobile Communication (GSM) beacons [147]. Additionally, social media services (e.g., Twitter, Facebook, or Swarm) represent a comprehensive source of user-generated location sequences through geotagged photos, posts, or check-ins at various locations [46, 227, 228].

These advances in tracking technologies build the foundation for data mining approaches and enable novel applications in various use cases by providing large

spatio-temporal data volumes [226, 395]. Trajectory data mining summarizes the entire process of analyzing spatio-temporal data and extracting useful information [349]. For example, several hundred million users use fitness applications and wearables to record their running activities [55]. In team sports, professional teams track their players' movements to improve performance as well as to avoid injuries and overloading [118]. Several million timestamped locations are generated per game (e.g., about three million data points per game in the German Bundesliga) [233, 281]. Even in the medical area, spatio-temporal data mining techniques are applied in neuroscience (e.g., brain activity analysis based on neural activations or blood flowing) and precision medicine [23, 71, 288, 318, 356]. Additionally, trajectory data is the foundation for a broad spectrum of urban services and smart city applications [370, 374]. For transportation network companies (TNC), such as Uber, Lyft, or DiDi, the current position information and its course over time can be helpful for demand estimations and optimizations of the fleet management [126]. To optimize their business processes, these companies constantly track the positions of the drivers. Uber alone generates spatio-temporal data for over 14 million trips each day [111]. The largest ride-sharing company in China, DiDi, accumulates more than 106 terabytes of trajectory data per day to provide route planning and travel time estimation services [96]. Novel spatio-temporal data mining approaches and applications that analyze the data can significantly impact and improve business decisions in various industries. Detailed demand predictions and efficient order dispatching strategies represent a crucial advantage over competitors for transport network companies. However, managing, storing, and processing spatio-temporal data is not a trivial task due to the massive volumes of continuously captured data and the performance requirements in various spatio-temporal use cases [104, 136, 367].

1.2 Challenges of Spatio-Temporal Data Management

Data management for spatio-temporal applications is challenging as various use cases have critical performance demands which require interactive response times on large data volumes (e.g., trajectory-based navigation systems or passenger request dispatching for ride-hailing services) [302]. Based on the characteristics of spatio-temporal trajectory data, there are four key challenges [367]: (i) the data volume, (ii) the high update rate (data velocity), (iii) the required query latencies of analytical queries, and (iv) the inherent inaccuracy of the data. Furthermore, the data characteristics which are determined by the specific application and used technologies – such as the accuracy of the tracked locations, the dimension of the coordinates, and the sample rate – differ, which makes the development of a general trajectory data management system complicated [104]. Additionally, the total cost of ownership is critical for companies that store spatio-temporal data. Due to the amounts of data accumulated in various use cases, a reduction of costs (e.g., necessary storage capacities) significantly impacts the profitability of such systems [31, 33, 237]. For these reasons, the need to efficiently analyze and query this data requires the development of sophisticated techniques [125].

In recent years, various systems have been developed for trajectory data management and analytics. However, traditional approaches are usually designed for particular needs, which forces users to stitch together heterogeneous

systems to analyze trajectory data in an inefficient manner [79]. In contrast to standalone storage systems specialized for trajectory data, relational database systems enable a simplified integration of different data sources (e.g., business data). Database systems are heavily used in all kinds of software to store and retrieve data [110]. Based on the widespread acceptance of Database Management Systems (DBMSs), a variety of storage techniques and access methods have been developed and optimized by researchers [126]. Relational databases are also applicable for storing spatio-temporal data in the sample point format. The sample point format is the most common data format for trajectory data and stores each observed location as a tuple with a set of attributes [302, 320, 395]. On the one hand, by integrating spatio-temporal data management into relational database systems, the data querying benefits from the optimized data processing capabilities and continuous improvements in the research area of database systems. On the other hand, traditional relational databases are general-purpose systems designed to manage most real-world use cases and workloads sufficiently [165]. As spatial, temporal, and spatio-temporal data benefit from being treated specifically [126], general-purpose systems usually do not achieve the performance and compression ratios of specialized systems [64]. To mitigate these effects and integrate the access methods (e.g., intersects for spatial data) required by Geographical Information Systems (GISs), modern data management platforms include specialized engines for specific data types (e.g., spatial, temporal, or spatio-temporal data) [14, 217, 244, 345, 352, 384, 391].

To provide low latency query services for spatio-temporal data mining applications, the data management has shifted to in-memory systems [250, 352, 390]. Based on the relatively limited and expensive DRAM capacities of main memory-optimized DBMS, the efficient utilization of the available resources is necessary to lower the memory footprint and consequently reduce the related total cost of ownership to store large volumes of spatio-temporal data [33, 37]. To efficiently utilize the available DRAM capacities, modern database systems support various tuning possibilities to reduce the memory footprint (e.g., data compression or tiering) or increase performance (e.g., additional index structures). Data encoding has been applied to database systems for decades to mitigate bandwidth bottlenecks and reduce storage requirements. While removing additional data structures or applying compression techniques with higher compression rates reduces the memory footprint, it also influences the runtime performance. Therefore, these tuning options are only used defensively by database administrators (DBAs) [58].

In general, the optimization of the DBMS for spatio-temporal applications is complex as the data characteristics (e.g., value distribution) are constantly changing [390]. For instance, in the TNC use case (cf. Section 1.1), the number of drivers at some locations may be larger during the day and relatively small at night. This means that the query execution (e.g., the number of accessed partitions or efficiency of index structures) for querying the same region may be quite different at different times [58]. Additionally, properties such as sample rate, spatial reference system, and data accuracy vary between applications and positioning systems. Consequently, different optimization approaches for spatio-temporal data have been developed for specific use cases, which cannot easily be transferred to other application domains. Furthermore, we observed that various optimizations based on general data properties do not sufficiently take into account the application-specific access patterns implemented in the

application layer. For example, the access frequency and required resolution of spatio-temporal data points usually change over time. Specific applications need detailed positional information to incorporate the current circumstances and ignore trajectory data after a selected timeframe, as the data no longer reflect the current situation (e.g., the traffic situation in a specific timeframe [397]). Sophisticated analytical applications (e.g., demand predictions) often use less detailed data, as various machine learning approaches cannot manage the high volumes and granularities of raw trajectory data [209]. Here, we identified high potentials to reduce the operating costs by minimizing the data footprint or increasing the performance by considering application-specific access patterns and changing data characteristics in the various tuning opportunities of modern databases.

1.2.1 Optimization Capabilities of Modern Database Systems

Database management systems provide a broad spectrum of opportunities to tune their physical design and configurations [21, 169, 398]. By dividing the data of a table into various partitions, modern database systems enable fine-grained tuning decisions [86, 178, 241]. As depicted in Figure 1.1, this concept allows the application of different configuration optimizations (e.g., sorting, indexing, and compression scheme) independently for each of these data partitions. All these configuration decisions impact the overall system's performance and resource consumption.

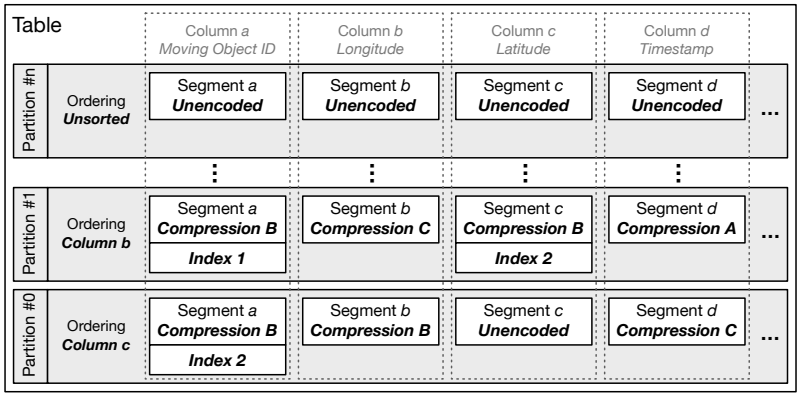

Fig. 1.1: Depiction of the storage layout and table configuration for an exemplary table with n data partitions. Based on the horizontal data partitioning, different configuration optimizations (e.g., compression and indexing tuning options) can be applied for each partition independently.

To efficiently utilize the available resources (e.g., DRAM capacities), modern database systems can apply different table configuration optimizations (e.g., data compression or secondary indexes) to reduce the memory footprint or increase performance. While removing additional data structures (e.g., indexes) or applying more heavy-weight compression techniques reduces the memory footprint, these decisions also affect the runtime performance. Many approaches have

been developed that focus on improving a specific aspect but do not sufficiently consider the impact on other configuration decisions. In existing work, there are approaches to improve the compression schema selection [1, 33, 69, 178]. Other research focuses on the selection of optimized index structures [73, 166, 236, 295] or data tiering [6, 84, 130, 348]. All these individual decisions optimize a specific aspect of a table configuration and impact the overall memory consumption and runtime performance. Furthermore, they mutually influence each other, making the computation of performance-optimized and memory-efficient configurations challenging due to the high number of possible combinations [11, 402].

For DBAs, the implications of different tuning options on the runtime performance are difficult to estimate [13, 38, 58]. Non-stable workloads that change over time, a lack of domain knowledge, and contradicting requirements of multiple applications working on the same data further complicate the situation [72]. Due to its complexity, database tuning often requires considerable effort from experienced DBAs [343]. The costs for qualified database personnel constitute a significant factor in the total cost of ownership (TCO) of database systems [52]. As the overall tuning is hard to optimize, various vendors use relatively simple threshold-based approaches, which tier data partitions to lower-cost storage mediums with higher latencies or apply compression techniques with a higher compression rate based on defined thresholds (e.g., data volume or timeframe). We argue that more advanced strategies for tuning decisions can significantly improve the overall system's performance and reduce operating costs. By applying fine-grained table configurations considering multiple tuning options (e.g., compression, data tiering, sorting, and indexing), we are able to optimize each data segment (cf. Figure 1.1) individually for the specific workload and data characteristics. Furthermore, we can utilize the available memory resources more efficiently and reduce the manual tuning overhead by automatically improving data management.

1.2.2 Research Context

In this thesis, we focus on different aspects to improve the trajectory data mining process, which summarizes the entire procedure of accumulating, analyzing, and visualizing spatio-temporal data. Based on our trajectory data mining framework, the process can be divided into different steps. In this context, the complexity of each step depends on the collected data and requirements of the application scenario. As depicted in Figure 1.2, we introduce novel approaches for two main research questions in this thesis. First, we investigate the use case of a TNC to analyze the impact of trajectory data-driven decision support systems on relevant business processes. In this context, the developed approaches cover the entire trajectory data mining process. Second, we focus on improving the data management and query processing of spatio-temporal data. By optimizing multiple database tuning options for the specific workload and data characteristics of spatio-temporal applications, we are able to increase performance or reduce memory consumption. These optimization approaches are not limited to the TNC use case and can be applied to further application scenarios (e.g., sports analytics or smart city applications). Based on these aspects, we pose the following research questions:

- *How can we improve business processes in specific application scenarios by spatio-temporal data mining?*

  To demonstrate the impact of spatio-temporal data mining applications on business decisions and processes, we analyzed the use case of a TNC. These companies regularly track their drivers' positions for route planning, travel time estimation, and urban capacity analyses [96]. The data enables the development of novel applications and algorithms to optimize processes such as the order dispatching of incoming passenger requests.
  In the highly competitive ride-sharing market, optimized and cost-efficient dispatching strategies represent a crucial business advantage. One weakness of many state-of-the-art dispatch algorithms is the accuracy of the last observed location of available drivers that are used as input data. The current location of a driver is not exactly known since the observed locations can be outdated for several seconds and affected by noise. These inaccuracies affect dispatch decisions and cause critical delays or inefficient assignments. To address the problem, we introduced an approach based on location probabilities that are determined by spatio-temporal data mining of past trajectory data. In this context, we analyzed the capabilities of our prediction results to (i) avoid critical delays, (ii) estimate waiting times with higher confidence, and (iii) enable risk-averse dispatching strategies.

- *How can we determine cost and performance-balancing table configuration optimizations consisting of fine-grained mutually dependent tuning decisions for the specific data properties and workloads of spatio-temporal applications?*

  The unprecedented scale of positional information generated in various applications poses an urgent demand for a practical storage mechanism for trajectory databases [379]. As DBMSs are usually implemented in a general fashion, spatio-temporal data can benefit from being treated specifically [126]. Modern databases have multiple tuning options to improve the storage configuration. As various optimizations for spatio-temporal data have been developed for specific application characteristics, the impact on memory consumption and runtime performance is hard to predict for particular workload and data characteristics. Additionally, the data properties (e.g., number and distribution of moving objects) and the access characteristics for specific data partitions vary over time and space. [58, 390]. Hence, it remains challenging to choose an efficient tuning configuration for a concrete application [379].
  There have been several efforts to automate the optimization of database management systems [343]. The constant adjustment of the applied tuning options based on currently existing data and workload characteristics could save costs and lead to more efficient table configurations [169]. In the context of fine-grained database optimizations, the challenge of such systems is to find efficient configurations with a scalable method [11]. In contrast to existing solutions that often optimize a single aspect (e.g., index or compression scheme selection), we propose a joint optimization approach for these tuning options. Zilo et al. [402] argue that mutually dependent configuration decisions should be optimized simultaneously as long as the problem complexity allows a joint optimization approach. As the different configuration decisions mutually influence each other, the joint optimization allows for the

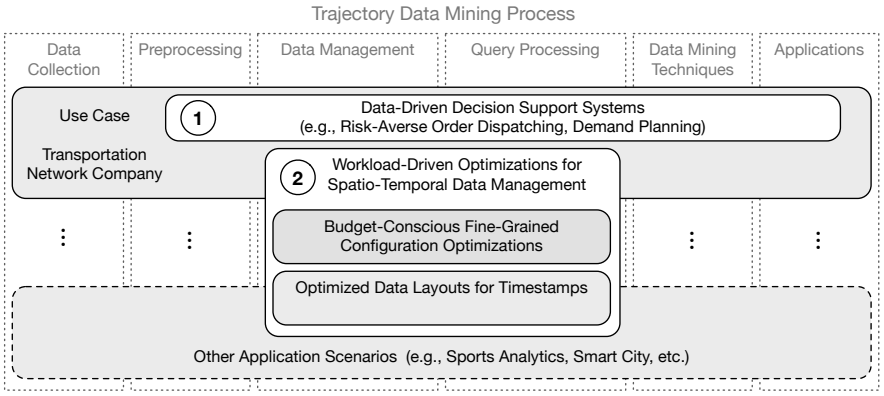

Fig. 1.2: Overview of the research context based on the different steps of the trajectory data mining process defined by our trajectory data mining framework (cf. Chapter 2). In research question ①, we focus on the application scenario of a transportation network company and develop solutions that cover the entire process (cf. Chapter 4). In research question ②, we focus on workload-driven optimization approaches for spatio-temporal data. In this context, we analyze the joint optimization of fine-grained tuning options (cf. Chapter 5) and storage layouts for timestamps (cf. Chapter 6) to improve data management and query processing for various application scenarios.

consideration of side effects, which are difficult to identify in the sequential optimization of features.

Each of those individual tuning problems is, in general, already challenging due to the high number of possible combinations of mutually dependent individual decisions. By incorporating domain knowledge, we can still address a joint optimization of these dimensions as we exploit the specific characteristics of spatio-temporal data and applications, i.e., a limited number of columns and few query types. With the independent selection of configurations for data partitions, we can reflect the specific access patterns of spatio-temporal data mining applications in the data management, which are usually only implemented in the application layer [275]. By considering the specific data and access characteristics of spatio-temporal applications in the selection process of different database optimizations, we are able to leverage these to reduce the memory footprint and increase performance.

Furthermore, we identified that, for various spatio-temporal applications, the performance is significantly impacted by the temporal component. In contrast to storing the positions of moving objects, there is less focus on optimized storage concepts for efficiently storing large sequences of timestamps. We identified that the chosen storage layout significantly impacts memory consumption and runtime performance based on analyzing different storage concepts for timestamps. In addition, based on the data and workload characteristics (e.g., number of distinct values, sequences of equal values, or sample rate), the effectiveness of different compression techniques depends on the application-specific properties. To improve the storing of timestamps

for spatio-temporal data mining applications, we focused on a joint optimization approach of the storage layout and compression techniques for a given workload.

1.3 Contributions

This work contributes the following advances to the current state of research in spatio-temporal data management and data mining:

- **A data mining framework for spatio-temporal applications**

  To cover different aspects of the spatio-temporal data mining process (e.g., data collection, preprocessing, and query processing), we introduce a trajectory data mining framework that summarizes the entire analysis process. In advance of existing approaches, the framework has a specific focus on data management. Based on the framework, we surveyed concepts to store and process trajectory data. This research was published in the following paper:

  [275] Richly, K.: *A Survey on Trajectory Data Management for Hybrid Transactional and Analytical Workloads*. In *Proceedings of the IEEE International Conference on Big Data (BigData)*. 2018, pp. 562–569

- **A probabilistic location prediction approach to enable risk-averse dispatch decisions for transportation network companies**

  Based on an analysis of the dispatch processes of a real-world transportation network company, we identified the use of inaccurate and outdated location information of the drivers as one reason for inefficient dispatch decisions. Therefore, we developed an algorithm to determine location probabilities based on driving patterns observed in past trajectories. We explain the impact of predicted locations on dispatch decisions and introduce strategies to use these predictions to apply risk-averse dispatch decisions. This research was published in the following papers:

  [279] Richly, K.; Brauer, J.; Schlosser, R.: *Predicting Location Probabilities of Drivers to Improve Dispatch Decisions of Transportation Network Companies based on Trajectory Data*. In *Proceedings of the International Conference on Operations Research and Enterprise Systems (ICORES)*. 2020, pp. 47–58
  [284] Richly, K.; Schlosser, R.; Brauer, J.: *Enabling Risk-averse Dispatch Processes for Transportation Network Companies by Probabilistic Location Prediction*. In *Operations Research and Enterprise Systems*. Springer, 2022, Volume 1623 by *Communications in Computer and Information Science*, pp. 21–42
  [285] Richly, K.; Schlosser, R.; Brauer, J.; Plattner, H.: *A Probabilistic Location Prediction Approach to Optimize Dispatch Processes in the Ride-Hailing Industry*. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS)*. 2021, pp. 1830–1840

Publication [284] is an extended journal version of paper [279]. The publication [285] is a more detailed explanation of the developed algorithm to predict location probabilities. The three authors collaboratively wrote the papers. Janos Brauer implemented and conducted the experiments presented in the papers.

- **A general approach to jointly determine fine-grained table configuration optimizations for spatio-temporal applications**

  We introduce fine-grained table configurations to reflect the application-specific access patterns for spatio-temporal data in the data management layer and optimize the configurations correspondingly. Moreover, we present a linear programming (LP) approach to determine memory-efficient and performance-balancing table configurations. As we jointly optimize the compression, indexing, and sorting tuning options for each data partition individually, the computation of an efficient configuration is challenging due to the vast number of possible setups of mutually dependent tuning decisions. Based on different cost dependencies (e.g., considering sorting effects and indexing strategies), we developed three LP models and evaluated them concerning runtime performance and scalability. Furthermore, we demonstrate that the model with relaxed cost dependencies can improve the general approach's scalability by determining comparable results. Further, we demonstrate on a real-world dataset with cost efficiency constraints that our models allow the memory footprint to be significantly reduced with equal performance or increased performance with equal memory size, compared to established rule-based heuristics. This research was published in the following papers:

  [282] Richly, K.; Schlosser, R.; Boissier, M.: *Joint Index, Sorting, and Compression Optimization for Memory-Efficient Spatio-Temporal Data Management*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2021, pp. 1901–1906
  [276] Richly, K.: *Optimized Spatio-Temporal Data Structures for Hybrid Transactional and Analytical Workloads on Columnar In-Memory Databases*. In *Proceedings of the VLDB PhD Workshop*. 2019, pp. 1–4

  The publication [276] is a contribution to the VLDB Ph.D. workshop that introduced the concepts of leveraging fine-grained database optimizations to reflect spatio-temporal access patterns and optimize the configurations correspondingly. In publication [282], the author is the first author and wrote the paper, implemented the approaches, and conducted the experiments. Rainer Schlosser contributed many ideas and detailed many sections. The co-authors improved the material and its presentation.

- **Extensions of the linear programming models to consider data tiering decisions, robust configuration selection, and reconfiguration costs**

  We further adapt the developed LP models to the advanced requirements of different use cases by introducing three enhancements: (i) data tiering

capabilities, (ii) robust configuration selection, and (iii) consideration of reconfiguration costs. Based on the data volumes of spatio-temporal applications, the consumed DRAM is a significant cost factor for main-memory optimized databases. To reduce the TCO, modern database systems tier infrequently accessed parts of the data to slower, but also less costly storage mediums (e.g., solid-state drive (SSD) or non-volatile random access memory (NVRAM)). Consequently, we extend the base models to integrate data placement decisions. The developed models can optimize the configuration of a table for multiple storage devices with a given capacity per storage device. Furthermore, we enhance our LP approaches to incorporate a worst-case optimization for potential workload scenarios. As real-world workloads typically change over time and future workloads are not entirely predictable, the performance can be negatively affected if the actual workload differs from the predicted one. The enhanced models are able to determine robust configurations with adequate performance for different workload scenarios specified by the DBA.
Additionally, we present an extension to consider reconfiguration costs. Based on changing access patterns, the applied table configuration might be outdated. All individual tuning optimizations produce modification costs (e.g., changing the sorting order of a chunk). To determine optimized configurations for the given input parameters, the models often apply numerous reconfigurations with only a minor impact on the overall performance. However, huge modification costs are not desirable in practice. By considering modification costs in the models, we can identify and perform only minimal-invasive modifications. This research was published in the following paper:

[283] Richly, K.; Schlosser, R.; Boissier, M.: *Budget-Conscious Fine-Grained Configuration Optimization for Spatio-Temporal Applications*. In *Proceedings of the VLDB Endowment* 15(13), 2022: pp. 4079 – 4092

The thesis author prepared the majority of the original draft for publication [283]. The author developed the underlying concept, implemented the approach, and designed and executed all experiments. Martin Boissier implemented the necessary enhancements of the *Hyrise* query optimizer and supported the implementation of the benchmark infrastructure in *Hyrise*. Rainer Schlosser supported the development of the different models and model extensions. Martin Boissier and Rainer Schlosser co-authored the paper and improved its material and presentation.

- **A workload-driven optimization approach for memory-efficient storage layouts for timestamps in columnar in-memory databases**

  Memory-efficient data management of observed locations' timestamps is challenging as numerous compression approaches in columnar databases support contradicting data characteristics (e.g., low number of distinct values, sequences of equal values). Based on an evaluation of different storage concepts for timestamps, we introduce a heuristic approach to jointly optimize the applied data layout and compression scheme for a given workload. Furthermore, we demonstrate the impact of attribute decomposition strategies that store parts of timestamps in independent columns to improve

the data characteristics for various lightweight compression approaches (e.g., dictionary-encoding) and reduce the memory traffic for standard access patterns. This research was published in the following paper:

[277] Richly, K.: *Memory-Efficient Storing of Timestamps for Spatio-Temporal Data Management in Columnar In-Memory Databases*. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*. 2021, pp. 542–557

The previous list of contributions spans the main scope of this work. Further contributions were introduced and published by the author of this thesis that impacted the development of data management optimizations. We list these contributions here but do not discuss the approaches and applications in detail. A list of all publications is presented in the appendix (see Appendix A.3).

- **Sports Analytics: An approach for the automated analysis and detection of events in team sports**

  For professional sport clubs, performance analytics (e.g., training control via key performance indicators (KPIs) or video analysis) of the overall tactics of a match or the players' skill levels are an essential aspect. These analytics are based on positional data on the one hand and specific game events (e.g., pass or shot on target) on the other hand. The positional data of the ball and players are tracked automatically by cameras or via sensors. However, the events are still captured manually by humans, which is time-consuming and error-prone. We implemented and evaluated different machine-learning approaches to detect such events automatically in trajectory data. Furthermore, coaches, scouts, and video analysts extract information about the strengths and weaknesses of their team and opponents by manually analyzing video recordings and statistics. As video recordings are an unstructured data source, finding specific game situations and identifying similar patterns is a complex and time-intensive task. We developed an application that enables users to find specific situations in video recordings and calculate situation-specific KPIs by analyzing spatio-temporal data. This research was published in the following papers:

  [274] Richly, K.: *Leveraging Spatio-Temporal Soccer Data to Define a Graphical Query Language for Game Recordings*. In *Proceedings of the IEEE International Conference on Big Data (BigData)*. 2018, pp. 3456–3463
  [281] Richly, K.; Moritz, F.; Schwarz, C.: *Utilizing Artificial Neural Networks to Detect Compound Events in Spatio-Temporal Soccer Data*. In *Proceedings of the ACM SIGKDD Workshop on Mining and Learning from Time Series (MiLeTs)*. 2017, pp. 13–17
  [278] Richly, K.; Bothe, M.; Rohloff, T.; Schwarz, C.: *Recognizing Compound Events in Spatio-Temporal Football Data*. In *Proceedings of the International Conference on Internet of Things and Big Data (IoTBD)*. 2016, pp. 27–35

  The publication [274] is a contribution that describes a visual query language for video recordings developed by the author. Publication [281] is a collaborative effort of all three authors. Florian Moritz implemented and conducted

the experiments presented in the publication. Publication [278] was collaboratively written by the thesis author, Max Bothe, and Tobias Rohloff. Max Bothe and Tobias Rohloff implemented and conducted the experiments presented in the publication. Christian Schwarz improved the material and its presentation in both manuscripts.

- **Different visualization and analysis concepts for spatio-temporal data in the context of smart city applications**

  We developed two example applications to demonstrate the impact of spatio-temporal data mining for smart city use cases. Based on the NYC Taxi and Limousine Commission trip record data [332], we implemented an interactive visualization concept to identify profitable areas and time frames. This application enables users to improve demand predictions and optimize pricing strategies by providing detailed information about past passenger request distributions. Additionally, we introduced a recommendation system that uses the trip record data to optimize existing public transportation networks. This material was published in the following papers:

  [286] Richly, K.; Teusner, R.: *Where is the Money Made? An Interactive Visualization of Profitable Areas in New York City*. In *Proceedings of the International Conference on IoT in Urban Space (Urb-IoT)*. ACM, 2016, pp. 43–46
  [287] Richly, K.; Teusner, R.; Immer, A.; Windheuser, F.; Wolf, L.: *Optimizing Routes of Public Transportation Systems by Analyzing the Data of Taxi Rides*. In *Proceedings of the International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*. 2015, pp. 70–76

  In publication [282], the author is the first author who wrote the paper and developed the visualization concept. Ralf Teusner improved the material and its presentation. Publication [287] is a collaborative effort of all five authors. The presented application was developed by Alexander Immer, Fabian Windheuser, and Leanard Wolf. The paper was written collaboratively and the experiments were conducted by the five authors.

Some of the content presented in this thesis was originally published in condensed form in conference proceedings and journal publications with Springer, EAI, ACM, or IEEE. The corresponding publications containing reused sections are first-authored by the author of this thesis. All substantial components of the work, including conceptualization and experiment design, have been directly carried out or led by the author of this thesis. Reprints were made with permission from the publishers (see Appendix A.1).

1.4 Outline

The remainder of the thesis follows the structure outlined in this section. Chapter 2 introduces the foundations. We describe the characteristics of spatio-temporal data and workloads, set the research context based on a developed trajectory data mining framework, and introduce the architecture of modern

columnar database systems using the research database *Hyrise* as an example. In Chapter 3, we present related work with a focus on spatio-temporal data management systems and database configuration optimization approaches. The following Chapter 4 presents an approach to optimize the passenger request dispatching of TNCs. We describe the limitations of applied state-of-the-art dispatch strategies based on the last observed location of drivers and introduce an algorithm to determine location probabilities by analyzing driving patterns in past trajectory data. Further, we evaluate the accuracy and runtime of the presented algorithm based on a real-world dataset of a TNC. Based on the predicted location probabilities, we describe different risk-averse dispatch strategies that enable the consideration of critical delays caused by inaccurate positional information. Chapter 5 introduces a workload-driven joint optimization approach for spatio-temporal data management. In this chapter, we present the implications of different tuning options on memory consumption and the performance of modern DBMS. Furthermore, the optimization process is described and different linear programming models are introduced to determine performance and cost-balancing fine-grained table configurations. Additionally, we present different enhancements to the base models to integrate data tiering decisions, reconfiguration costs, and robustness. In Chapter 6, we describe an optimized approach to store timestamps in columnar databases. In Chapter 7, we describe the experimental setup based on the real-world dataset of a TNC introduced in Chapter 4 and share the evaluation results of the different optimization approaches. In this context, we demonstrate that, for the TNC example, workload-driven fine-grained optimizations allow us to reduce the memory footprint (up to 71% by equal performance) and increase the performance (up to 90% by equal memory size) compared to established rule-based heuristics. Chapter 8 presents directions for future work and summarizes the conducted research.

2

Background

In this chapter, we present background information on (i) the analysis process of spatio-temporal data, (ii) different approaches and optimizations to store spatio-temporal data, and (iii) columnar in-memory data management. As each of the presented research areas is quite complex, we made the selection of which aspects to cover depending on the knowledge that later chapters build on. Parts of the content presented here, including the trajectory data mining framework, have been published [275].

2.1 Spatio-Temporal Data Mining

To summarize and formalize the whole process of trajectory data mining, we developed a framework as shown in Figure 2.1. The framework is based on the work of Feng et al. [104], Zheng [395], and Tanuja et al. [329]. It describes the different challenges of each step in the spatio-temporal data mining process. In contrast to previous research, we focus on the data management layer to derive various aspects and requirements relevant to storing and processing spatio-temporal data.

Trajectory data is collected from various moving objects with sensors by location-acquisition technologies such as GPS. The accumulated data has to be processed to gather information for different applications. As shown in Figure 2.1, the general process can be structured into four layers, which are preprocessing, data management, query processing, and data mining. The final layer of the framework is the application layer, which classifies the different application scenarios. Based on the application-specific requirements and characteristics (e.g., positioning system, data volumes, and performance specifications), the complexity and relevance of the different framework layers vary.

In this section, we discuss the challenges of the individual process steps. The preprocessing step streamlines the data provided by various data sources (e.g., sensors) in the data collection. Additionally, mechanisms to improve the data quality and integrate data from different sources (e.g., heterogeneous positioning systems) are performed. The data management tackles the issue of storing large-scale trajectory data efficiently in a scalable manner and enabling high-performance access. In the next process step, we focus on the different spatio-temporal query types and the optimized data retrieval. Furthermore, it

includes different trajectory-based metrics (e.g., distance metrics). These access characteristics and metrics are used in various spatio-temporal data mining approaches, which are applied in different application scenarios.

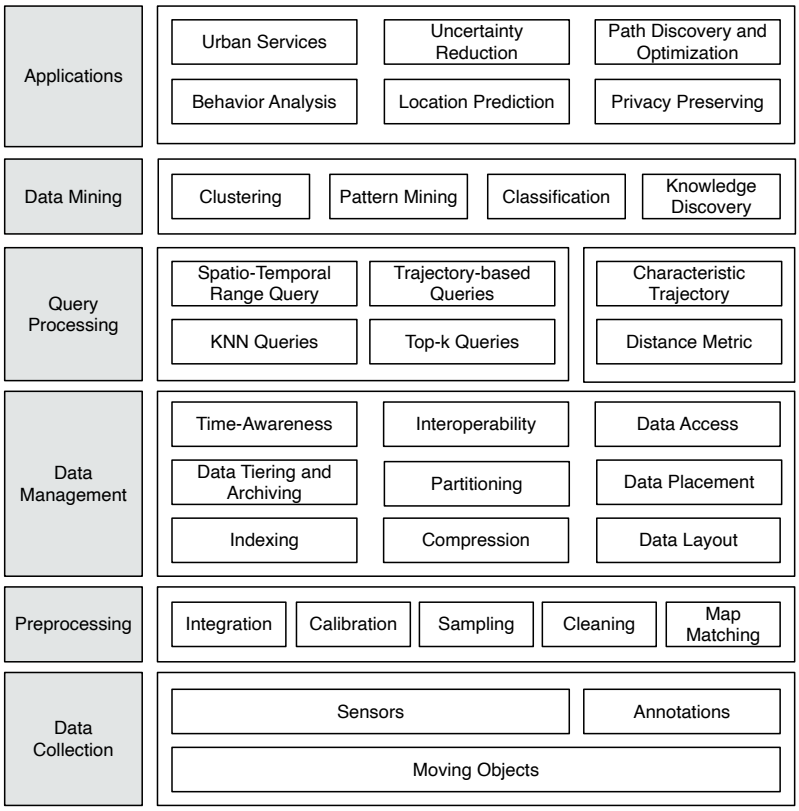

Fig. 2.1: Framework of the trajectory data mining process.

2.1.1 Data Collection and Characteristics of Spatio-Temporal Data

Nowadays, a wide range of objects (e.g., mobile phones, cars, or airplanes) are equipped with location-acquisition technologies. Various sensor-based technologies such as GPS, RFID, infrared and ultrasonic systems, or location estimation via 802.11 enable the indoor and outdoor tracking of moving objects [26, 67, 147, 305]. Moreover, there are different systems that determine the trajectory of an object based on video data [77, 134]. Additionally, social media services (e.g., Twitter, Facebook, or Swarm) provide a comprehensive source of user-generated location sequences through geotagged photos, posts, or check-ins at various locations [46, 227, 228]. Based on the used technologies and application-specific configurations, the data characteristics like the accuracy of the tracked locations, the dimension of the coordinates, and the sample rate differ [104]. Further aspects that influence the selection of a tracking technology

are the costs, requirements (e.g., weight or dimensions), and the ability to use already available resources.

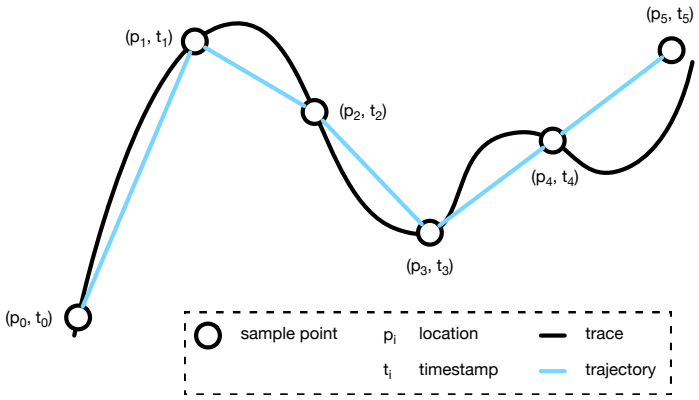

Fig. 2.2: Representation of a trajectory (blue) generated by sampling from a moving object's continuous trace (black).

As displayed in Figure 2.2, a trajectory approximates a moving object's trace in geographical space. In contrast to the continuous trace of a moving object, the captured trajectory is only a sample of locations that the moving object has passed at a specific time [104]. Usually, the trajectory is represented by a series of chronologically ordered observed locations [395]. Each sample point consists of a multi-dimensional coordinate in a geographic reference system p_i (e.g., latitude and longitude) and a temporal component t_i (e.g., timestamp) [323]. In addition, further attributes can be stored for each sample point (e.g., the operating status of a moving object).

Furthermore, the sample rate quantifies the frequency at which the sample points are gathered. Depending on the use case, it is also possible to have varying periods between sample points (e.g., geographically annotated social media data). Based on the sample rate, there is always a certain degree of uncertainty, as we have no information about the moving object's behavior between two sample points (see Figure 2.2). The amount of sample points is a trade-off between the accuracy of the approximated trajectory and the resulting costs (e.g., transmission and storage costs) that must be determined for each application. Additionally, we have inaccuracies based on technical limitations of the positioning systems as well as noise that affects the accuracy of the observed locations. In this context, accuracy is defined as the closeness between the measured value and the ground truth of the moving object's trace [12]. As displayed in Figure 2.2, the location p_5 is not located on the trace of the moving object. These devergences can range from several centimeters up to several meters depending on the applied tracking technologie [331]. As spatial annotations are often less detailed (e.g., a specific street or city), we also have to consider uncertainty introduced by imprecise location information in different use cases.

Another aspect is the used spatial reference system (e.g., World Geodetic System 84) that significantly impacts the complexity of the applied algorithms [176]. The Cartesian spatial reference system is often used for indoor

applications and applications in a limited space. This planar system allows the application of more straightforward algorithms (e.g., distance calculation) but leads to significant deviations of multiple hundreds of kilometers if applied on a global scale. For applications on a global scale (e.g., air traffic management), there are different systems (e.g., geodetic reference system) that have to apply more complex algorithms on approximations of the earth's surface (e.g., a sphere or an ellipsoid) to avoid these deviations but also consume significantly more computation time [183].

2.1.2 Preprocessing

The different technologies to track the movement of objects generate vast volumes of trajectory data. Additionally, we have to consider the quality of positional information due to sensor noise and other technical factors [367]. Consequently, it is often necessary to preprocess the data. The general intention of the techniques used in the preprocessing step is to (i) increase the data quality, (ii) integrate data from different sources (e.g., different positioning systems), and (iii) control the captured data volumes. To avoid time-consuming operations on large trajectory datasets, sampling is a standard approach to reduce the data amount by minimizing the sample points of a trajectory to the most representative points [196, 243]. Data integration aims to provide unified access to data from different data sources [81]. The integration of trajectory data from various data sources is challenging due to different sample rates, accuracies of the sample points, spatial reference systems, or data representations [27, 221, 253, 293].

Techniques to increase the data quality can be classified into the categories: data cleaning, calibration, and map matching. The data cleaning step is necessary for various applications to filter sensor noise, outliers, and abnormal behavior of objects in terms of maximum speed or unreachable constraints [102, 103, 249]. As inaccuracies of sample points can lead to deviations in various metrics of a trajectory (e.g., distance or length), the calibration of trajectory data uses algorithms to adjust the spatial information. Additionally, it aims to unify the sampling strategy and rate of heterogeneous trajectories [321, 322]. For various trajectory data mining use cases, map matching can be applied to optimize the accuracy of the data. By using additional information (e.g., road networks), map matching algorithms adjust the gathered spatial information to specific positions on the map (e.g., road segments) [80, 238, 269, 310].

2.1.3 Data Management

The large volumes of continuously accumulated data and increasing performance requirements of applications foster the development of optimized storage concepts for spatio-temporal data. Based on the specific data and access characteristics, spatio-temporal data greatly benefit from being treated specifically [126]. The most addressed research areas are compression and indexing of spatio-temporal data in this context, but also various other factors have to be considered (e.g., data layouts, partitioning, and tiering) for trajectory data management. The usage of optimized compression approaches can save costs by reducing the required storage capacities and increase performance by mitigating bandwidth bottlenecks or enabling the efficient application of data mining

algorithms [363]. Besides general compression approaches (e.g., delta encoding), lossy compression techniques exist for spatio-temporal data that use trajectory simplification or generation algorithms to reduce the number of sample points [203, 239, 310, 379]. These lossy compression approaches represent a tradeoff between compression ratio and maximum error that defines the deviation between the compressed trajectory and the uncompressed one [104].

Additional data structures like indexes are introduced to increase the query performance of different query types. These data structures are generally optimized for specific access types and characteristics and require additional memory resources. An example of an index structure optimized for trajectory data is *TrajTree*, which was developed to manage retrieval tasks like KNN queries [271]. Popa et al. introduced another trajectory index structure that optimizes the access costs of spatio-temporal range queries [292]. Other index structures for different application scenarios are *SETI*, *SEB-Trees*, or *TB-Tress* [47, 258, 311]. Moreover, further relevant aspects impact the interoperability, memory consumption, and performance of trajectory data management systems. Especially in complex systems, the data layout, placement, partitioning, and tiering concepts have to be optimized to avoid unnecessary data transfers between nodes, storage devices, or systems [267, 268]. For instance, in some systems, the sample points of a moving object are divided into several sub-trajectories and stored in different data partitions to enable pruning and improve the query processing [254, 353]. In general, all these approaches are optimized for specific access and data characteristics, which makes the application in different use cases challenging. In Section 2.2, we discuss the different aspects of spatio-temporal data management in more detail.

2.1.4 Query Processing

The query processing layer is divided into two parts. The first part focuses on classifying spatio-temporal access patterns, and the second on metrics and simplification techniques used in various data mining approaches. For spatio-temporal applications, we observed different access patterns, which have to be processed by trajectory data management systems. In this context, some optimizations developed for spatio-temporal data management are more beneficial for specific access patterns. We distinguish four query types: (i) trajectory-based queries, (ii) spatio-temporal range queries, (iii) KNN queries, and (iv) top-k queries [123, 346, 396]. Trajectory-based queries refer to the trajectory of a single moving object and return the entire trajectory, a specific segment of the trajectory, or related information like the length of the trajectory [179]. Spatio-temporal range queries include all queries which filter the dataset accordingly to specified spatial, temporal, or spatio-temporal filter criteria. These query types are the most common ones, as they are used to filter the data for different data mining techniques and visualization approaches. Other query types in spatio-temporal applications are KNN queries, which refer to similar trajectories or trajectory segments, and k-top queries, which refer to trajectories with a specific characteristic (e.g., highest average speed) [302].

Additionally, we have different metrics and techniques that are required by various data mining approaches. As various data mining approaches cannot process the data volumes of raw spatio-temporal data, there are algorithms that

summarize similar trajectories. By representing a set of trajectories via a characteristic trajectory, the amount of data that has to be processed can be significantly reduced [188]. The determination of these characteristic trajectories is a tradeoff between accuracy and the necessary memory consumption. The approach of representative trajectories is also used in trajectory compression methods. To determine sets of similar trajectories, a distance metric is required. It is challenging to define a similarity metric for comparing paths or subpaths (e.g., different sampling strategies or at different sampling rates) [187, 335, 350]. Examples of such distance metrics are the Euclidean distance [389], longest common subsequence [347], discrete Fréchet distance [92, 120], or Edit distance [56, 163].

2.1.5 Data Mining and Applications

A broad spectrum of applications is driven by trajectory data mining, enabling data-driven decision support. Based on our trajectory data mining framework (cf. Figure 2.1), we subdivide the applications layer into six parts: (i) urban services, (ii) behavior analysis, (iii) location predictions, (iv) path discovery and optimization, (v) privacy-preserving methods, and (vi) uncertainty reduction.

Urban services focus on improving several aspects in urban areas [104, 164]. In this context, spatio-temporal data analysis is used in combination with other data sources to optimize traffic flow [326, 388], infrastructure [197, 247, 287], and derive knowledge about the use of areas [375, 387]. Another application domain is the analysis of behavior patterns of groups of moving objects or individual ones. Mobility behavior patterns of various kinds of moving objects (e.g., people, animals, or ships) can be used to detect anomalies or identify reoccurring patterns [114, 141]. By recognizing different forms of mobility, we can develop a more detailed understanding of mobility and the importance of different means of transport [365, 385].

Furthermore, location and arrival time predictions are essential for various industries. For instance, transportation companies (e.g., airlines, carriers, or public transport) employ spatio-temporal data to determine estimated arrival or travel times, which are used for scheduling or passenger information systems [24, 358]. Also, location predictions are applied to predict the next point of interest a user will visit and to provide recommendations for points of interest [61, 207, 240]. Related subjects are path discovery and optimization, where historical trajectory data is leveraged to find a suitable route (e.g., fastest path problem, most frequent path problem) [60, 213, 301]. Additionally, path discovery is a technique that is used in the research area of uncertainty reduction. In this context, path discovery algorithms are applied to estimate the trace of a moving object between two consecutive observed locations, which is especially relevant for trajectory data with low sample rates [394]. The last category of applications is privacy-preserving spatio-temporal data mining [16]. Various researchers study and develop methods to analyze large amounts of trajectory data without drawing conclusions about the behavior of individual moving objects.

All these different application domains are based on trajectory data mining algorithms. Trajectory data mining algorithms can be summarized and classified into four categories similar to traditional data mining, i.e., pattern mining, clustering, classification, and knowledge discovery [187, 198]. In this context,

the algorithms of conventional data mining techniques must be adopted to incorporate the characteristics, properties, and data volumes of spatio-temporal applications.

2.2 Aspects of Spatio-Temporal Data Management

Based on the different access types and data characteristics presented in the trajectory data mining framework (cf. Section 2.1), there are various opportunities to improve the data management of spatio-temporal data for a specific use case. Consequently, different application-specific optimizations were developed to enhance the storing and processing of spatio-temporal data. This section classifies and discusses the different approaches.

2.2.1 Data Layouts for Trajectory Data

The applied data layout is a distinguishing feature of different trajectory data management systems that significantly impacts memory consumption and runtime performance. As some data layouts are more suitable for specific query characteristics and require additional overhead to process other query types, the selection should be optimized for the particular access patterns of the application domain. Besides performance requirements, another aspect that influences the decision for a data layout is whether the trajectory data is stored in a standalone system, specialized for trajectory data, or in a standard database system or framework. In the following, we describe three different data layouts for spatio-temporal data.

- *Sample Point Format:* The sample point format is the data layout that is used in the majority of spatio-temporal data management systems [66, 96, 390]. It stores each observed location as a tuple with the following attributes: (i) the moving object identifier, (ii) a location (e.g., specified by GPS coordinates), and (iii) the observation time of the location. In some applications, the tuple is extended by values that provide additional information about the observed location. The structure of the sample point format streamlines the integration of spatio-temporal data storage capabilities into existing data management systems and frameworks (e.g., relational databases). The format enables the application of optimized compression, partitioning, and sorting (e.g., space-filling curves) techniques to improve the performance, especially for spatio-temporal range queries and KNN queries [59, 290, 334].
- *Key-Value Format:* A straightforward approach to store trajectories or trajectory segments is to save them as objects in a key-value store. In this format, the moving object identifier is used as the key referencing a sequence of chronologically ordered locations. Alternatively, the observed locations could also be saved as a sequence of pairs consisting of a timestamp and location, or structured formats like GeoJson could be used [42]. This approach enables efficient access to all sample points of a specific moving object. In contrast, operations like spatial or spatio-temporal range queries are more complex to process because it is necessary to scan all stored objects. Additional data structures, which contain a minimum bounding box as well as a start and end timestamp for each trajectory, are stored to address this problem and significantly reduce the number of objects that must be processed [193].

- *Frame Format:* The frame format divides the temporal space of the raw trajectory data into frames of a fixed duration [352, 353, 393]. The frame duration depends on the application-specific characteristics of the spatio-temporal data and has to be defined by the database administrators. Afterward, each observed location is allocated to a frame based on its timestamp. As the approach stores one spatial location for each frame, we have to aggregate the data if multiple observed locations are assigned to the same frame. A drawback of this approach is that the uncertainty is increased, especially for larger frame intervals, because of the loss of information about the concrete timestamp and the potential aggregation of multiple observed locations in a time frame. In contrast, we can reduce memory consumption as the data layout's structure already reflects the temporal component, and we do not need to store a timestamp for each observed location. The different frames are stored as individual columns in a database table. To store the frames in a table with a limited number of columns, the sequence of frames is divided into sections. Each section consists of a fixed number of consecutive frames and is called a frame group. An advantage of this data layout is that we can efficiently analyze or compare the positions of different moving objects in one frame or a set of frames. Furthermore, changing the frame duration requires a time-intensive reorganization of the data structure. Also, the approach produces a lot of null values for heterogeneous trajectories with different sampling strategies and sampling rates.

Furthermore, some approaches store the data redundantly in multiple data layouts to optimize the query processing for different query types [195].

2.2.2 Data Partitioning

Data partitioning is a common approach to enable the effective parallel processing of spatio-temporal data [25, 79, 96, 127]. Based on the specified partitioning criteria, we distribute the data on several data partitions. Besides the parallel processing of partitions, an advantage of this approach is that the flexibility and scalability of systems are increased as the individual data partitions can be stored on different storage mediums or servers. In general, it is possible to partition the data vertically or horizontally [10]. As the spatio-temporal components are often accessed together, vertical partitioning is only applied in specific use cases (e.g., various additional attributes are stored for each sample point). Different approaches are conceivable to partition trajectory data horizontally: (i) temporal partitioning, (ii) spatial partitioning, (iii) object-based partitioning, (iv) parameter-based partitioning, and (v) hybrid partitioning approaches.

Based on the partitioning criteria, we are able to apply pruning strategies that skip irrelevant data partitions during query processing [34]. By maintaining statistics for each data partition (e.g., min/max values), we can determine which partitions have to be considered for processing a specific query. Consequently, we can significantly increase the performance by ignoring all other data partitions. For instance, the runtime of a spatial range query benefits if the observed locations are partitioned by a spatial partitioning criterion, as a possibly large number of partitions can be pruned during query execution. In contrast, most partitions are search candidates and must be processed for this query type if the partitioning criterion is time [79].

For the selection of an appropriate partitioning strategy, we have to consider different aspects: (i) parallel data processing, (ii) efficient pruning, (iii) partition size, and (iv) load balancing. The system should generally leverage the parallel scan of equally distributed partitions to increase the query performance. Additionally, each partition should have a proper size to minimize the overhead introduced by metadata structures [390].

- *Object-Based Partitioning:* Object-based partitioning is a strategy where the trajectory data is partitioned by the identifier of the moving object. This approach is efficient for trajectory-based access patterns that query all data points of a moving object or segments of the moving object's trajectory in a specific time interval or geographic region. Additionally, for queries that determine metrics for individual moving objects (e.g., length of a trajectory), the number of partitions that have to be processed can be reduced. A significant disadvantage is that spatio-temporal range queries have to scan a large number of data partitions since every partition could contain relevant data [179].
- *Spatial Partitioning:* The general idea of spatial partitioning is to store trajectories or trajectory segments together, which are geographically co-located. This means that all sample points included in a specific geographical region are stored in one partition. A spatial partition strategy is beneficial for coordinate-based range queries and spatial KNN queries, as various data partitions can be excluded based on the partition criteria [336]. A potential disadvantage of this partitioning approach is the sophisticated selection of the spatial partition criteria, which requires knowledge about the data distribution as well as future changes in the data distribution [66]. Sub-optimal partition criteria could lead to unbalanced data partitions, negatively affecting the query performance.
  Additionally, this approach is not well suited for trajectory-based queries, as potentially all partitions have to be scanned to reconstruct the trajectory of a specific moving object. This problem could be addressed by applying additional data structures that store for each moving object the minimal bounding box. By using this data structure, we only have to process the partitions overlapped by the bounding box.
- *Temporal Partitioning:* Temporal data partitioning is a common approach for time series data and spatio-temporal data [154, 255]. The partitioning strategy is efficient for temporal range queries and spatio-temporal range queries [376]. Similar to spatial partitioning, this approach has disadvantages concerning trajectory-based queries. As the insert order defines a kind of temporal order on the data in various applications, time-based partitioning avoids a repartitioning process for new data entries [376]. Moreover, maintaining equally sized partitions is relatively simple in this approach.
- *Parameter-Based Partitioning:* Parameter-based partitioning is a strategy where the spatio-temporal data is distributed on various partitions based on a parameter, which is additionally stored for each sample point. An example of such a parameter could be the status or type of a moving object. This approach is only relevant for selected use cases, where this specific parameter is often used to filter the trajectory data (e.g., datasets containing trajectories of different means of transport). Nevertheless, this approach has

disadvantages for all other query types compared to the other partitioning strategies.

- *Hybrid Partitioning:* Various trajectory data management systems apply hybrid partitioning approaches [66, 367, 390]. The most common is a two-dimensional partitioning strategy, which divides the data on the temporal dimension first and on a second level on the spatial dimension. This partitioning strategy allows efficient data pruning for temporal and spatio-temporal queries. Also, coordinate-based queries benefit since the partitioning scheme can limit the relevant data partitions. However, the selection of the partitioning criteria to avoid unbalanced partitions is a non-trivial task. For that reason, Zhang et al. [390] developed an adaptive approach to optimize the partitioning scheme and adapt it to changing data characteristics.

2.2.3 Compression

Modern location-acquisition technologies create vast amounts of spatio-temporal data, which cause expensive costs for storage, transmission, and query processing [392]. To lower communication loads, reduce storage requirements, and optimize data processing, there are different compression approaches for trajectory data. Besides various general compression techniques provided by modern data management systems, compression approaches developed explicitly for spatio-temporal data are applied to minimize the memory footprint. The different compression techniques are based on (i) trajectory simplification, (ii) representative trajectories, (iii) semantic compression, or (iv) leveraging spatio-temporal data characteristics. As most of these approaches are lossy compression techniques, there is always a tradeoff between the compression ratio and the added maximum error. In general, the higher the compression ratio, the less exact the accuracy of the compressed trajectory data will be [104, 216].

Trajectory simplification, also known as line generalization, reduces the number of observed locations required to represent a trajectory [83]. Based on a distance measure (e.g., perpendicular Euclidean distance or direction-aware distance), the goal is to find an approximation of the original trajectory with minimal information loss [379]. There are error-bound algorithms that minimize the number of sample points for a given maximum error and size-bound algorithms that minimize the maximum error for a given number of sample points. A well-known example of a trajectory simplification approach is the Douglas-Peucker (DP) algorithm [83]. The DP algorithm computes the sample point causing the largest perpendicular Euclidean distance for a given trajectory. Only the two endpoints are kept if the distance is smaller than a specified maximum error. Otherwise, we split the trajectory at the determined point with the greatest distance and recursively repeat the process. Several other compression techniques use trajectory simplification to reduce the number of sample points [57, 203, 229, 234, 239, 360, 368]. A comprehensive evaluation of different trajectory simplification algorithms was presented by Zhang et al. [379].

Representative trajectories are another approach to compress spatio-temporal data. Instead of storing individual trajectories, we cluster similar trajectories and store only one trajectory with additional information (e.g., the number of trajectories represented and the included moving objects) for each of these identified clusters. By applying this compression strategy, we lose knowledge about

the individual trajectories. An example of this approach is *TrajStrore* [66]. TrajStrore splits the original trajectories into sub-trajectories and clusters them into cluster groups. For each cluster group, only one representative trajectory and meta-information, like the time values and moving object identifier of the original trajectories, are stored. Zhao et al. [392] developed the *REST* framework, which also uses reference trajectories to compress trajectory data. The system addresses the computational issue of determining the reference trajectories by introducing an efficient greedy algorithm. Similar approaches based on the aggregation of values are also used for time-series data [161, 255].

Semantic compression is another approach applied in some use cases [159, 204]. It leverages additional geographical information (e.g., road networks) to compress trajectory data. A map-matching algorithm is applied to assign the observed locations to defined artifacts (e.g., road segments) [53, 399]. Instead of the observed location (e.g., GPS coordinates), we can store the trajectory as a sequence of identifiers (e.g., road segment identifier).

Additionally, other approaches leverage the characteristics of spatio-temporal data. For example, delta-encoding is applied as the coordinates of successive points usually only vary slightly [66]. Storing the offset between points requires less memory but introduces additional data processing overhead.

2.2.4 Time Awareness

For trajectory data management systems, the required time awareness of spatio-temporal applications impacts memory consumption and query performance. There are different approaches to storing the temporal component of observed locations. The selection of a suitable storage concept strongly depends on the specific use case and the used location-acquisition approach (e.g., sample rate or precision of the temporal information). In the following, we discuss the three concepts: sequential order, time interval, and timestamp.

- *Sequential Order of Observed Locations:* Instead of storing a timestamp for each sample point, the temporal information in this approach is defined by the sequential order of the sample points. Consequently, we are able to reduce the memory consumption (e.g., 4 to 8 bytes to store a timestamp) as we do not explicitly store the temporal component. This approach is only suitable for use cases where the specific time is less important (e.g., path reconstruction). For applications with a fixed sample rate, it is also possible to save only the timestamp of the first sample point and determine the timestamps of the subsequent sample points based on the initial timestamp and the position of the sample point in the sequence of observed locations.
- *Time Frames:* Another approach is to store the temporal data in fixed-length time intervals. This concept enables uniform data access for trajectories with different sample rates. Nevertheless, it increases uncertainty as we lose information about the exact timestamp. As mentioned in the frame format (cf. Section 2.2.1), we can leverage that aspect and use the data layout to store the temporal information implicitly. Consequently, we could reduce the required memory resources to store a sample point, but we increase the processing complexity of the several query types.
- *Timestamps:* The most common approach for spatio-temporal data is to store the temporal component as a timestamp for each sample point. It

enables accurate temporal analysis and has high flexibility concerning sample rates, accuracy, and transmission errors. In this context, the precision of the timestamp and the selected data format have an impact on the memory footprint as well as the performance.

2.2.5 Data Placement and Tiering

Data placement is a topic that is strongly related to data partitioning. Especially in distributed trajectory data management systems, the distribution of data partitions on different nodes or storage devices is a complex task. For minimizing NUMA effects, storing and processing data frequently accessed together on the same node [267, 268] is optimal. Furthermore, effective data placement strategies must consider the workload characteristics and the corresponding utilization of computational resources [179]. In this context, the optimization objectives of approaches that distribute data partitions on different nodes are (i) to enable efficient parallel data processing, (ii) to balance the load on the computational resources, and (iii) to reduce the data transfer between nodes. Moreover, different approaches consider data replication or optimized additional data structures on various nodes [192, 201]. Xie et al. [367] presented *Elite*, an elastic storage infrastructure for trajectory data. The system enables on-demand provisioning of storage and computational resources. Also, it introduced a mechanism to allocate computational resources based on periodically collected statistics from nodes and a workload estimator.

Another approach that can improve the capabilities of spatio-temporal data management systems to handle large amounts of trajectory data is data tiering. In contrast to traditional data management systems and data management for time-series data, optimized data tiering approaches are less common for trajectory data [339]. Especially, trajectory storage systems that use main memory as primary persistency face the challenges posed by the cost of DRAM and the inherent problems in scaling DRAM capacities [88]. By transferring data to more cost-efficient storage devices with higher latencies, the required DRAM resources can be reduced significantly [35]. In this context, optimized data tiering strategies based on the access frequency of data partitions can significantly reduce operating costs. However, various trajectory data management systems apply only simple rule-based approaches that tier data based on specified thresholds (e.g., data size or timeframes).

2.2.6 Index Structures

To improve the performance of storage systems for spatio-temporal data, a broad spectrum of indexing approaches was developed [215]. As indexes are auxiliary data structures, they consume further memory resources, which makes selecting whether an index should be applied and which one challenging [166]. Modern database systems enable the application of various standard index structures, such as B^+-trees, to improve the performance of data retrieval operations. Moreover, there are optimized approaches for main memory-optimized database systems [97, 99]. These general-purpose index structures can be used for various data types and are not designed for the characteristics of spatio-temporal data. An approach to optimize the B^+-tree for trajectory data is the B^x-tree proposed

by Jensen et al., which uses a space-filling curve as part of the linearization function [145, 146]. Chen et al. [58] introduced the ST^2-tree, a self-tunable index that can be reconfigured automatically to enable B^+-trees to adapt to changing data distributions in spatio-temporal data.

Another method is to apply well-known index structures for spatial data, such as *R*-trees, quadtrees, or different optimizations of these approaches [28, 122, 124]. As these data structures are designed to index spatial points and geometric shapes, they do not consider trajectory data's temporal component. Similar to the B^+-tree, different index structures for spatio-temporal data have been developed based on the *R*-tree [3, 258, 272, 291, 311, 330]. Furthermore, various other indexing approaches were developed explicitly for spatio-temporal data or to enhance existing index structures to address the properties of trajectory data [232, 246]. In this context, the indexes are often optimized for specific access characteristics and application domains. For instance, *TrajTree* [271] and *Dita* [302] are spatio-temporal index structures that are optimized for similarity search between trajectories.

Additionally, different systems apply hybrid index structures. For instance, *TrajSpark* [390] and *Elite* [367] use a distributed multi-level hybrid index structure. Both index structures have three levels: the first level is a temporal index, the second is a spatial index, and the third is a traditional B^+ tree. Both systems also use the index structure to partition the data and divide the trajectory of a moving object into different trajectory segments. A similar segmentation-based approach is also used by *TrajStore* [66] and *SETI* [47]. *SETI* uses a two-level index structure to decouple the spatial and temporal dimensions.

Furthermore, Kreska et al. [173] showed that learned index structures could be an efficient alternative to traditional indexing concepts for conventional database systems. Pandey et al. demonstrated the applicability of learned index structures for the spatial component of trajectory data [245].

2.2.7 Data Access and Interoperability

Another aspect of spatio-temporal data management is to support efficient query mechanisms for spatio-temporal data and enable interoperability with other systems or applications. There are different approaches to providing query interfaces for trajectory data. For instance, Koubarakis and Kyzirakos developed *stSPARQL* [170], an extension of *SPARQL* to query spatio-temporal RDF data. Furthermore, there are different enhancements for the SQL standard to improve the handling of temporal, spatial, and spatio-temporal data. *TSQL2* (Temporal Structured Query Language) is a temporal extension for SQL [309]. Additionally, there are different approaches to extending SQL to store and manipulate spatial data structures [18, 91, 182, 289]. Specially designed for spatio-temporal data are the language extensions *STQL* (Spatio-Temporal Query Language) [95] and SQL^{ST} [54]. Both approaches add a set of temporal and spatial operators (e.g., a meet operator that identifies moving objects that were in the same place at the same time) to facilitate the querying of trajectory data. Furthermore, the integration of spatial and spatio-temporal capabilities into relational database systems simplifies the integration of other data sources. Therefore, different database vendors include spatio-temporal data management capabilities in their systems [117, 217, 352, 391].

2.3 Columnar In-Memory Data Management Systems

The workload-driven optimization approaches for spatio-temporal data introduced in this thesis are built for relational, column-oriented in-memory database systems. As the concepts of column orientation and in-memory computing have an impact on different aspects of our implementation, we explain these concepts in Section 2.3.1. Furthermore, we describe the research database *Hyrise* and depict how our optimization approaches leverage the storage concepts of *Hyrise*.

2.3.1 Storage Concepts of In-Memory Column Stores

By storing data on fast dynamic random access memory (DRAM) instead of relatively slow solid-state drives (SSDs) or hard disk drives (HDDs), in-memory database systems outperform traditional database systems. In contrast to conventional buffer-cache-based databases built under the assumption that not all data fits into main memory, in-memory databases keep the entire data in DRAM [264]. For decades, database systems were forced to store the data on slower storage mediums with higher capacities (e.g., hard drives) based on limited DRAM resources. To mitigate the orders of magnitude access latency differences between DRAM and disk (cf. Figure 2.3), the data was loaded into the main memory buffer cache (also known as buffer pool) when it was accessed and kept there until it was evicted by loading other data into DRAM. Almost all data in such database systems are mapped to database pages, which are fixed-sized blocks of several Kilobytes (e.g., 8KB or larger is typical) and have the same representation both on disk and in main memory to avoid translation overheads [97]. When the database accesses data, it first performs a lookup to determine whether the corresponding database page is located in the buffer cache. If the page is not in the main memory, an I/O operation to load the page into the buffer cache is triggered. As DRAM accesses are significantly less expensive than disk accesses, an efficient buffer cache management, including optimized prefetching and eviction strategies, was needed to benefit from the faster DRAM performance as often as possible [324, 325]. Based on different drawbacks of operating systems caches for database workloads, the implementation of these caches was integrated into the DBMS, which further increases the complexity of DBMS architectures [131, 313].

With increasing capacities and decreasing prices of DRAM, it became viable to store all data in the main memory and avoid page-based indirections [261]. Färber et al. [97] stated that avoiding the buffer cache indirection can boost the performance of database systems up to an order of magnitude by bypassing indirections to resolve physical record pointers and page-level latch contention in the buffer cache. Instead of data access via database pages, most main memory-optimized database systems use pointers for direct memory access.

The first in-memory database widely used in productive systems was SAP HANA, introduced in 2010 [98]. In the following years, all major vendors introduced their own in-memory database systems, including Oracle TimesTen [177] and Microsoft SQL Server Hekaton [78]. Additionally, there is a wide range of in-memory research databases, such as HyPer [160] or H-Store [156]. A more detailed overview of main memory database systems is provided by Färber et al. [97] as well as Zhang et al. [381].

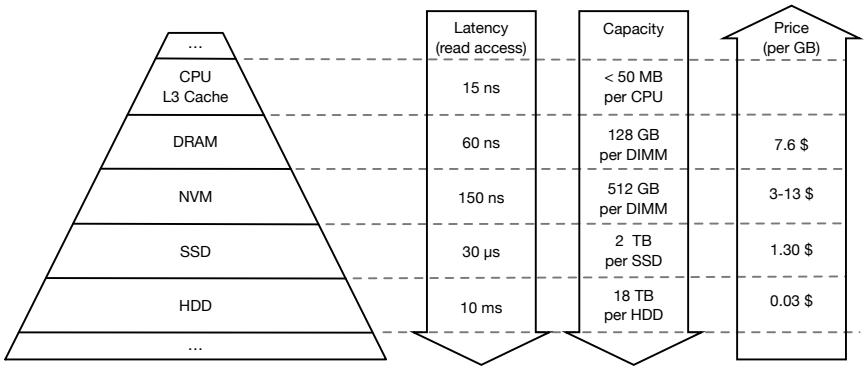

Fig. 2.3: The memory and storage hierarchy, including key performance figures. Numbers are only given for reference and vary from product to product. Figure partially based on data taken from [17, 19, 139, 299].

As displayed in Figure 2.3, CPU caches have a significantly faster access latency compared to DRAM. Conversely, these caches have a limited capacity, which requires efficient utilization of the available cache slots and sophisticated caching strategies. In this context, DBMSs have to optimize the utilization of CPU cycles and memory bandwidth [260]. Based on modern CPUs' increased performance that exceeds the advances in memory latency, DRAM access represents a performance bottleneck for database systems [36]. The cache management is performed on the granularity of cache lines. Based on the cache size, the cache can hold a limited number of cache lines, which represent a fixed-size block of the memory. The size of the cache lines depends on the CPU architecture (e.g., 32, 64, or 128 Bytes). In the context of this discussion, we stick to the 64 Bytes of current Intel Xeon architectures. Based on this concept, an entire cache line is transferred even if only a single 8-Byte word is accessed. Consequently, 56 additional Bytes have to be transferred. To increase efficiency, we have to optimize the data organization to ensure that all data of a cache line can be used.

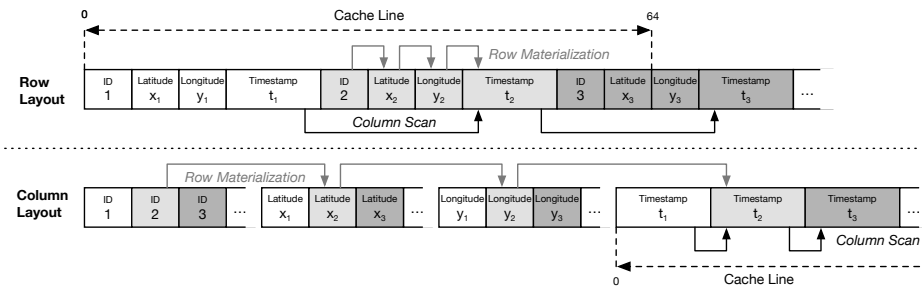

Fig. 2.4: Visualization of a row-oriented and column-oriented storage layout in the case of the exemplary table from Figure 1.1 with the four columns: ID, Longitude, Latitude, and Timestamp.

In relational database systems, the data is organized in two-dimensional tables with rows and columns. To store the data of a table on conceptually one-dimensional memory, there are two approaches, the row-oriented and the column-oriented storage format. As an example, we use the table introduced in Figure 1.1 with four columns. As shown in Figure 2.4, in a row-oriented format (top), all values of a row are stored contiguously. Furthermore, the tuples of different rows are stored successively. This storage format is beneficial for single tuple access (e.g., row materialization) as only a limited number of cache lines have to be transferred. In the example, all values of the observed location with the identifier 2 fit into a single cache line. In contrast, we have a less efficient cache line utilization for column scan operations. The specific values of the entries are stored one tuple size apart for each column. If we perform a scan operation on the *Timestamp* column, the visualized 64 Byte cache line only contains 16 Bytes of relevant information (i.e., two timestamp entries of 8-Byte). Here, we only use 25% of the cache line and correspondingly waste 75% of the memory bandwidth. This ratio can become even worse through additional columns so that only a single byte is used per 64 Byte cache line.

In the column-oriented layout (bottom), all values of a column are stored contiguously and the different columns are stored successively. Based on this layout, a scan operation on the *Timestamp* column is able to utilize the entire cache line. Especially for large tables, a high cache line efficiency can be achieved as no memory bandwidth is wasted. A drawback of the columnar layout is that the materialization of a row is significantly more expensive. Generally, one cache line must be transmitted for each table column. Due to the clear advantages and disadvantages of both storage formats for specific workload characteristics, there a different hybrid approaches that try to find an optimal combination of row- and column-oriented storage formats [20, 35, 116, 260]. The tradeoff between row-oriented and column-oriented layouts has been studied in more detail by Abadi et al. [2] and Zukowski et al. [404]. Already in 1985, Copeland and Koshafian performed a comprehensive analysis of the two storage formats. The columnar layout is referred to as the decomposition storage model in their work.

2.3.2 Hyrise: A Relational Columnar In-Memory Research Databases

For the evaluation of our workload-driven optimization approaches for spatio-temporal data, we use the relational research database *Hyrise*. *Hyrise* is a columnar main memory-optimized database. Each table in *Hyrise* is implicitly divided into horizontal partitions with a predefined maximum size (see Figure 2.5). A partition, called chunk, contains fragments of all columns of a table, whereby the section of a column stored in a chunk is referred to as a segment. There are two types of chunks, mutable and immutable chunks. Only the most recent chunk is mutable, and consequently, all insertions, as well as MVCC-enabled updates, are appended to this unencoded chunk. When this write-optimized chunk's capacity is reached, it becomes immutable, and a new mutable chunk is created. A disadvantage of this approach is that we increase the memory footprint by additionally storing per-chunk metadata and redundant information (e.g., per-segment dictionaries for dictionary-encoded segments). In exchange, the database system can benefit from pruning during query execution, distributing the workload more efficiently, and applying fine-grained optimizations of

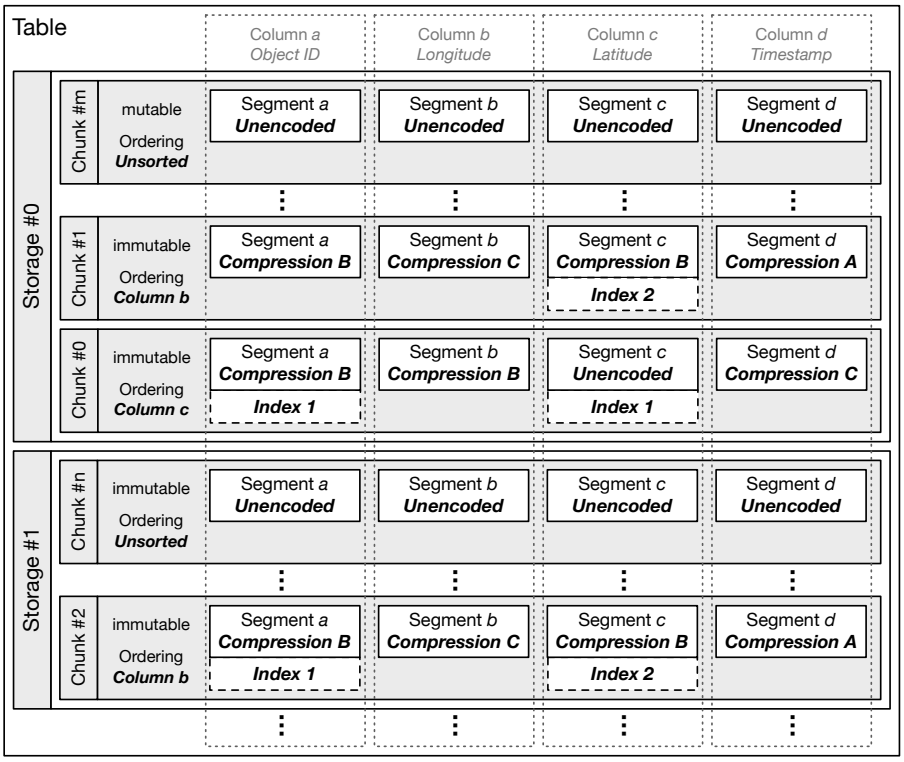

Fig. 2.5: Depiction of the storage layout for an exemplary table configuration for two storage devices. Besides the allocation decision, we are able to select for each segment a compression, indexing, and sorting tuning option.

table configurations. For these reasons, similar concepts are applied by various databases [178, 248, 252].

Fine-grained table configurations enable the application of different tuning approaches for various data chunks and segments of a table. Consequently, the configuration of different parts of the data can be optimized for specific workloads and data characteristics. *Hyrise* provides different tuning options, such as the sorting, indexing, and compression configuration of a chunk [282]. Also, *Hyrise* supports the tiering of entire chunks or segments to more cost-effective storage devices (e.g., NVRAM or SSD), cf, Figure 2.3. *Hyrise* uses C++'s polymorphic memory resources to provide a uniform interface that allows allocating data in DRAM, NVRAM, or on block devices using *UMap* [84]. *UMap* [256, 257] is a user-space page fault handler that allows – in contrast to *mmap* – limiting the buffer size of cached pages. This implementation enables *Hyrise* to store data on multiple storage devices.

All these configuration decisions have an impact on the system's performance and memory consumption. For in-memory databases, the used DRAM capacities are an important cost factor [31, 237]. Concerning spatio-temporal data volumes, minimizing the memory footprint can significantly reduce the system's operating costs. Different tuning options reduce the memory consumption but also have implications on the performance, which are difficult to estimate for database

administrators [13, 38, 58]. Consequently, various vendors apply relatively simple threshold-based approaches. Based on a defined threshold (e.g., data volume), data partitions are transferred to lower-cost storage mediums.

The developed optimization approaches introduced in this thesis are not limited to the application in *Hyrise*. All concepts can be adapted and applied to various relational data management systems. Furthermore, the different optimizations are particularly efficient for databases that partition the data and enable the application of different tuning options for various data partitions.

2.4 Summary

In this chapter, we introduced a framework (cf. Section 2.1) that summarizes and formalizes the entire process of trajectory data mining. Based on the trajectory data mining framework, we explained the challenges of the different process steps and described the specific characteristics of spatio-temporal data and applications. Afterward, we focused on the data management of spatio-temporal data (cf. Section 2.2). We classified and discussed different optimization approaches to enhance the storing and processing of trajectory data. In this context, we identified that there is a broad spectrum of optimizations (e.g., compression techniques or index structures) designed for specific application domains and that it is challenging to estimate the effectiveness of these application-specific approaches for other application characteristics (e.g., sample rate or access patterns). Furthermore, we explained the theoretical foundations of columnar in-memory databases (cf. Section 2.3) as these have an impact on different aspects of our data management optimizations for spatio-temporal data. Finally, we described the research database *Hyrise* and depicted how our optimization approaches leverage the storage concepts of *Hyrise*.

3

Related Work

In this chapter, we discuss existing approaches for spatio-temporal data management systems (Section 3.1) as well as workload-based and data-aware database optimizations (Section 3.2). In this context, we discuss how existing concepts and systems relate to our approaches.

3.1 Spatio-Temporal Data Management Systems

In recent years, several systems have been developed to store and process large amounts of spatio-temporal data, which are accumulated in different applications (cf. Section 1.1). In this context, the design of the various systems strongly depends on the characteristics of the addressed use case and different corresponding optimization objectives: (i) scalability, (ii) memory footprint reduction, (iii) elasticity, (iv) efficiency, or (v) query performance.

An approach to storing trajectory data is to use data management systems for spatial data, which provide support for different geospatial operations. Various systems, such as *SpatialHadoop* [94], *SpatialSpark* [390], *Simba* [366], or *LoacationSpark* [328], were primarily designed to store spatial data points, but also used for trajectory data. For the object-rational database PostgreSQL, the spatial extension PostGIS adds support for geographic objects allowing users to formulate geospatial queries in SQL [383]. Based on PostgreSQL and PostGIS, Zimányi et al. [403] developed *MobilityDB*, a moving objects database that extends the existing implementation by time-varying data types. As the system uses the existing operations, indexing, and optimizations, it benefits from the enhancements done to PostgreSQL and PostGIS. Our optimization approach follows a similar concept by improving the data management of general-purpose databases (e.g., *Hyrise*) for trajectory data. The spatial extensions of disk-based databases, such as PostGIS, MySQL, or Oracle Spatial, provide rich features and functions for geospatial data and queries but fall behind when performance is considered [244]. This is especially the case if interactive analysis on the latest transactional geographical data is a requirement. Pandey et al. [244] presented *HyPerSpace*, a spatial extension to the main-memory database HyPer to enable high-performance geospatial processing in a general-purpose database system.

Furthermore, with the increased availability of spatio-temporal data, specialized data management systems were built for different system infrastructures to address the specific characteristics of trajectory data further. *TrajStore* is a

disk-based dynamic storage system for spatio-temporal trajectory data developed by Cudre-Mauroux et al. [66]. The approach slices trajectories in multiple sub-trajectories and co-locates trajectory segments that are geographically and temporally near to each other. The system uses a quadtree as a spatial index to divide the trajectory segments into spatio-temporal regions. Afterward, the data of each region is densely packed in a block on disk, called a cell. Each cell is represented by one or more disk pages. Within a cell, the sample points of the sub-trajectories are chronologically ordered, and delta encoding is used to reduce the data footprint. Additionally, the system uses cluster-based compression to cluster similar sub-trajectories in each cell into cluster groups and store only one representative trajectory per group. Instead of keeping the individual sample points of the trajectories, only the representative trajectory and the time intervals of the original trajectories are stored for each cluster group. The storage structure of *TrajStore* is optimized for spatial queries over specific regions with relatively large time bounds. Since the data is stored on disk, the query execution time is dominated by disk access. Therefore, the primary evaluation and optimization criterion is the number of disk seeks.

Wang et al. developed *SharkDB*, an in-memory column-oriented trajectory store [353]. The system uses a hierarchical I/P frame approach to store and compress the raw trajectory data in a column-oriented data structure. Based on the characteristics of the trajectory data, the temporal space is divided into frames with a fixed time interval. Afterward, each sample point is exactly allocated to one frame based on its timestamp. The information about the accurate timestamp gets lost by assigning sample points to defined frames. This increases the uncertainty, especially for larger frame intervals. Additionally, the approach produces a lot of null values for heterogeneous trajectories with different sampling strategies and sampling rates. Furthermore, adopting the time intervals based on changing application requirements or data characteristics requires a time-intensive reorganization of the data structure. To reduce memory consumption, the sequence of frames representing a trajectory is divided into frame groups, which consist of a fixed number of consecutive frames. The first frame of each frame group is defined as an I-frame and is stored uncompressed. All other frames of a frame group are P-frames, which are stored delta encoded. Based on the frame groups, the system uses a hierarchical storage structure to increase the scan performance. The frame groups of the raw trajectory represent the lowest level of the hierarchical storage structure. The higher levels of the structure use the same I/P frame storage approach but only include the data of the I-frames of the lower level. The approach of *SharkDB* was adopted to a columnar table scheme and integrated into *SAP HANA* [352]. Due to the storage concept, finding a specific sample point or a set of trajectories in a particular area can be time-intensive as the system has to scan several columns.

Elite is an elastic infrastructure for spatio-temporal trajectory data developed by Xie et al. [367]. It is a peer-to-peer storage and index schema for elastic storage and query processing. The system's design focuses on efficiently processing spatio-temporal range queries and nearest-neighbor queries. The approach combines online transactional data processing and online analytical processing for large trajectory datasets and supports the processing of uncertain trajectory data. One of the key aspects is the on-demand provisioning of computational resources to accelerate query processing. In general, the indexing concept consists of three layers distributed across all system nodes. The first layer is represented

by a skip list that partitions the temporal domain into a set of time intervals. Each time interval is assigned to a torus, which is the second index layer. In the system, a torus corresponds to a cluster of nodes, whereby each node stores a specific subset of the data and routing table about its neighboring nodes. The routing table includes the IP addresses of the neighbors and their date ranges. For the communication between the different nodes of a torus, the CAN routing schema is used [273]. The third layer consists of an oct-tree and a hash table. The oct-tree is used to store the sample points of the trajectory and the hash table is used to map a trajectory's ID to its observed locations. The flexibility of the distributed system enables the on-demand allocation of storage and computational resources but also leads to additional routing overhead and data transfer between different nodes. Additionally, the system is a standalone solution to analyze large amounts of trajectory data, which complicates the integration of other data sources.

TrajSpark is a Spark-based trajectory data management system developed by Zhang et al. [390]. In contrast to LocationSpark [328], GeoSpark [373], or SpatialSpark [372], which use point-based partitioning and indexing strategies optimized for independent points, *TrajSpark* attempts to leverage the chronological order and characteristics of trajectories to increase the query performance. The system stores the trajectory data in segments, delta encoded and ordered by moving object identifiers in TRDDs, which is an adapted version of Spark's Resilient Distributed Datasets (RDDs). While TRDD only supports sequential scans, a multi-level IndexTRDD is introduced to improve the query performance. The IndexTRDD consists of a local and a global index structure. The first level of the global index is a temporal range index. The second level is a spatial quadtree and the last level is a traditional B+ tree. To adjust the applied data partitioning strategy to changes in the data distribution, the system uses a time-decay model. By monitoring the data distribution, the system creates a new spatial index to partition the data if the distribution of recently loaded data has greatly changed and exceeds a given threshold. In this context, the JSD distance is used to measure the difference between two data distributions [242].

UlTraMan developed by Ding et al. [79] is another data management system for trajectory data based on the distributed computing framework Spark. To avoid the usage of heterogeneous systems for data storage, processing, and analysis, *UlTraMan* provides a unified engine for a holistic trajectory data management and analytics solution. It has a flexible application interface, which supports customizable data organization, preprocessing, and analysis pipelines. By integrating pluggable components (e.g., index structures or processing techniques), the systems can be adapted to the requirements of different analysis scenarios. As many important techniques and optimizations are realized based on random data access (e.g., hash-maps or indexes), the authors enhanced Spark's RDDs, which facilitate sequential operations on the data, and integrated an abstraction called TrajDataset. This abstraction enables random access at both local and global levels. The flexible system design allows *UlTraMan* to be used in diverse application scenarios, but the implications of the different pluggable components on the memory footprint and performance are hard to estimate. Based on *UlTraMan*, Fang et al. [96] developed *Dragoon*, a hybrid system for integrated offline and online scalable trajectory data management and analytics.

Another distributed in-memory trajectory analytics system based on Spark is *DITA* [302]. In contrast to other systems, *DITA* is optimized for trajectory

similarity search and join operations. It can support the most widely adopted similarity functions (e.g., Fréchet distance) to quantify the similarity between trajectories. Based on a set of selected representative points, called pivots, the system creates a trie-like structure to index the pivots and effectively prune irrelevant partitions and dissimilar trajectories.

CloST is a spatio-temporal data storage system to support data analytics using Hadoop [327]. The main design objective of the system is to avoid scanning the entire dataset to process spatio-temporal range queries. Therefore, *CloST* stores the data in a table and partitions it hierarchically into blocks using all the core attributes: (i) object identifier, (ii) spatial location, and (iii) time. First, the data is partitioned based on the hash values of the object identifier and coarse time ranges. Second, the resulting partitions are further divided into second-level partitions according to a spatial index. Finally, each second-level partition is divided into a number of third-level partitions based on finer time ranges. In this context, the third-level partitions contain the actual records and higher-level partitions serve as indexes. Each third-level partition, called a block, is stored in a block file layout as a regular file on HDFS. The block file layout groups the records of a moving object and stores each group in a file section. The values are sorted by time for each section and organized in a column-store fashion. Each column is compressed with delta or run-length encoding for numeric values or gzip for non-numeric values to reduce the memory footprint. Additionally, each section is compressed again using gzip to reduce the data size further. To optimize the data distribution on blocks, *CloST* introduces a storage optimizer that periodically tunes the spatial and temporal partitioning based on the query log.

Li et al. [194] developed *TrajMesa* a distributed NoSQL trajectory query engine based on GeoMesa, an open-source indexing toolkit for spatio-temporal data [143]. Based on the precondition that disk storage costs are significantly cheaper than computing costs, *TrajMesa* stores two copies of trajectory data with different carefully designed keys in different tables. By using two storage approaches and indexes optimized for different access patterns, the system improves query processing by selecting the best-suited storage configuration for a given query. Based on the higher DRAM storage costs, our approach optimizes different partitions of the data for the occurring access patterns instead of storing multiple versions of the data.

3.2 Database Optimizations Based on Data and Workload Characteristics

This section focuses on approaches to optimize the selection of database tuning options based on workload and data characteristics. In this context, we briefly discuss (i) data compression, (ii) indexing, and (iii) data tiering. Furthermore, we analyze (iv) tuning approaches to jointly optimize multiple tuning configurations and (v) data type-specific optimizations for storing timestamps in databases.

3.2.1 Compression Scheme Selection

For DBMS, data compression is essential to reduce the used storage resources and, consequently, the operation costs of such systems [270]. Moreover, it allows efficient processing (e.g., vectorization using SIMD instructions) and mitigates bandwidth bottlenecks [32, 184]. On the negative side, data compression can lead to reduced performance for CPU-bound applications as additional CPU cycles are used to compress and uncompress the data [162, 362]. Based on the applied compression technique and implementation details of compression approaches, the performance and compression ratio can vary significantly [1, 68, 136]. Moreover, application-specific data characteristics and access patterns strongly influence the performance of compression approaches, making selecting efficient compression schemes complex [33]. As these implications are hard to estimate, DBAs apply data encoding only conservatively to avoid a negative impact on the runtime performance [32].

Based on the column-oriented database C-Store [315], Abadi et al. [1] presented an extension, which supports various compression schemes allowing operations directly on compressed data. Moreover, the authors introduced a decision tree-based approach to aid DBAs in deciding how to compress a specific column. The selection is based on workload and data properties but does not adapt to changing environments nor consider memory budgets. A similar approach was proposed by Lemire et al. [190] for integer data. They defined a set of rules to choose between different encodings.

For the database HyPer [160], Lang et al. [178] presented the concept of data blocks. Similar to our research database *Hyrise* (cf. Section 2.3.2), the system divides the data of a table into fixed-size chunks. Each data block is a self-contained container that stores one or more attribute chunks in a compressed format. The system supports different byte-addressable compression techniques (single value compression, ordered dictionary encoding, and truncation) to enable efficient point access. For each data block, it selects a compression scheme statically concerning the resulting memory consumption based on data characteristics. As each data block is self-contained, the chosen compression per attribute is limited to the corresponding data block. This blockwise compression is similar to our approach, where we independently apply different chunk-based or segment-based tuning options (e.g., encodings or indexes). A drawback of this approach is that we have an increased memory consumption caused by redundancy (e.g., identical entries in multiple dictionaries). However, it enables the selection of an optimized compression scheme for each column in each block, which can amortize the overhead [178].

Cen et al. [45] introduced the Learned Encoding Advisor (LEA), a learned approach to column encoding selection. LEA can optimize the compression scheme for encoded size, end-to-end scan performance on the target system, or a combination of the two. To leverage localized correlations in data, it enables fine-grained optimizations by selecting an encoding per column and block. LEA has two phases, a training phase and an inference phase. In the training phase, it trains three internal models for each encoding type to predict the encoded size (one model) and the overall scan time for the column (two models that work together) based on synthetic training data. In the inference phase, the system uses data statistics (e.g., cardinality, min/max) and sample statistics to predict the optimal encoding using random forest and linear regression. CodecDB [150]

models the encoding selection as a learning-to-rank problem. Based on data characteristics (e.g., value length, cardinality ratio, and sortedness), the system trains a model to estimate the compression ratio of a given encoding scheme on a dataset. For the training, different real-world datasets are used. CodecDB selects the compression scheme with the best estimated compression ratio. In contrast to LEA, CodecDB does not consider workload characteristics. Both approaches do not take into account memory budgets, robustness, or reconfiguration costs.

By analyzing light-weight integer compression, Damme et al. [69] found that sophisticated compression techniques can have significant impacts on both performance and compression ratios. The authors indicate that consideration of multiple dimensions is necessary to determine which technique is the best for a specific scenario. Also, Raman and Swart [270] presented a method based on a mix of column and tuple coding that leverages successive similar values and correlations of sorted relations to improve data compression for row-oriented databases. Based on similar observations and the fact that different decisions influence each other, we propose a joint optimization. For example, the sorting decision has a major impact on the data characteristics and, consequently, on the compression ratio of different approaches (e.g., run-length encoding). Additionally, the sorting of a column has an impact on the runtime, which can lead to obsolete auxiliary data structures.

Boissier and Jendruk [33] introduced a workload-driven selection of compression configurations with memory constraints for columnar databases. The approach uses a greedy heuristic to determine configurations based on data characteristics and estimated runtime performances via regression models. Furthermore, Boissier [32] presented a linear integer programming approach that yields optimal configurations and a greedy hybrid heuristic as an efficient and scalable solution for the compression scheme selection problem. The author presented two enhancements of the linear programming approach to achieve robustness by avoiding unexpected performance regressions. The first enhancement enables DBAs to define constraints on single query runtime changes. As the presented LP-based approach optimizes the cumulative workload runtime, several determined configurations can improve the overall runtime but increase the runtime for a specific query. To avoid these scenarios, the DBA can define a set of queries and a maximal runtime factor for these queries. By including these constraints in the model, performance requirements for individual queries can be guaranteed. The second enhancement focuses on equally distributed performance gains. The purpose of this adaption is to enforce that all workload queries have a similar relative runtime change compared to their optimal performance. To lower the chances of unexpected performance regressions, we also included an extension in our approach to determine robust configurations. In contrast to individual query runtimes, we focus on robustness concerning different potential workload scenarios.

In practice, many DBMSs solve the problem by defining a default encoding per data type with acceptable performance for different data and workload characteristics [150]. Although various commercial databases provide the means to estimate the impact of compression, they do not apply a compression selection mechanism that is fully exploiting the potential of automated workload-driven column compression [32]. Furthermore, none of the presented approaches considers the impact of other tuning options on the compression scheme selection. For instance, additional data structures (e.g., indexes) significantly affect memory

consumption and the performance of database operations, which can significantly impact the compression scheme selection process.

3.2.2 Index Tuning

The performance of certain database operations can be improved by applying auxiliary data structures such as secondary indexes. Accordingly, various indexing approaches optimized for specific access patterns or data types have been developed to reduce the runtime. For instance, there are different spatio-temporal index structures (cf. Section 2.2.6). In general, there is a tradeoff between performance improvements and the necessary memory consumption. While the performance of a DBMS is an important aspect for DBAs, the operating costs caused by the used memory resources represent a significant factor of the operation costs [380]. The efficiency of an applied index structure strongly depends on the workload and data characteristics. Therefore, indexes do not always accelerate performance. Due to expensive maintenance operations for indexes as well as specific access types (e.g., filter operations with low selectivity), the usage of additional indexes can even increase the workload processing time [115, 245].

Zhang et al. [390] observed that the characteristics of spatio-temporal data change over time, which leads to a decreased efficiency of data structures (e.g., indexes). They proposed a time-decay model to monitor the data distribution and adopt the used indexing schema accordingly. In the proposed multi-level index structure, they first divide the data into time ranges according to the temporal component and maintain for each time range an independent spatial index. To adapt to density and distribution changes over time and consider different densities between regions, Chen et al. [58] developed the Self-Tunable Spatio-Temporal B^+-Tree index (ST^2B-Tree). The index is based on a traditional B^+-Tree and requires no modification of the basic structure. The keys of the B^+-Tree are composed of a temporal and a spatial component. In this context, a Voronoi cell approach with a configurable number of reference points is used for the data partitioning. The grid granularity is determined by the object density around a reference point. Based on frequent updates to subtrees, the system can rebuild and optimize a subtree using a different set of reference points and grid size without significant overhead. In contrast to static index structures, these approaches are able to adjust and optimize the indexing concerning dataset and workload changes. However, they are not designed to decide whether it is beneficial to apply an index for a given workload nor to consider any memory limitations.

An approach to reduce the memory footprint of indexes are partial index structures [300]. Instead of indexing a complete set of tuples of a table, partial indexes include only a subset of tuples [314]. Partial indexes are especially beneficial if a set of tuples is accessed in a high percentage of requests and have a relatively low update rate [108]. Considering the often tremendous memory requirements of full table indexes, an advantage of partial index structures is that they are suitable for self-tuning and can grow or even shrink on demand [294]. Fuentes et al. [108] propose a tuning process that considers solutions containing partial and full table indexes. For the selection of partial index structures, the approach determines the set of indexable attributes and the set of possible

restrictions by analyzing a given workload. Based on a profit function, only partial indexes that have a greater benefit than a defined threshold compared to a full index are considered. The final configuration is determined based on an algorithm that stepwise adds index candidates to the final configuration that minimizes the execution costs for the given workload.

Based on horizontal partitioning, different systems enable the creation of partition-wise index structures [48, 86]. Wu and Madden [364] developed Shinobi, a system that determines an optimal set of non-overlapping range partitions and chooses indexes for each partition to maximize workload performance. It only indexes frequently accessed partitions. By applying indexes partition-wise, the costs of creating and maintaining an index on a single partition become cheaper, which enables the system to perform fine-grained optimizations. In contrast to the traditional full table indexes, the index must not always be updated when a tuple is inserted, updated, or deleted. Instead, only the index of the corresponding partition has to be adopted by data modifications. Additionally, they propose an algorithm to maintain the partitions as workloads change. To avoid a lookup operation for each partition's index to be executed for specific search conditions, Weisgut [361] proposed a partial index that is created table-wise but that stores only index entries for frequently accessed partitions. Thus, partition indexes are more efficient in terms of memory consumption as they can selectively index data, whereas a full table index contains index entries for all tuples of a table. Especially for trajectory data, partial index structures can be beneficial as full table spatio-temporal index structures can have a high memory footprint. Moreover, index structures optimized for the specific access patterns of different partitions can improve performance. For example, Ray et al. described a spatio-temporal index in which the spatial domain is organized as grid cells and a partial temporal index is maintained for each grid cell. The partial temporal index is applied with finer intervals for more recent data and with coarser granular intervals for older data.

Based on the high number of possible index candidates, the selection of an efficient configuration is challenging. Different database systems support only a limited set of index structures and have varying implementations that impact the performance. Moreover, based on the application-specific data and workload characteristics, the efficiency of index structures can vary significantly. Schlosser et al. [295] introduced a workload-driven index selection approach that builds on a recursive mechanism and accounts for index interaction. In contrast to other approaches, the presented strategy also allows the scalable computation of index configurations for large workloads. Kimura et al. [162] presented an index selection approach that selects viable secondary indexes and considers compressed alternatives for each index based on a given memory budget. In this context, they focus on accurate index size estimations. This index size-aware approach uses a heuristic to prune index candidates and configurations of candidates. A greedy selection of indexes was developed for IBM DB2 by Valentin et al. [342]. Based on a ratio between the runtime improvement and memory consumption, the index configuration is determined by a greedy heuristic. To account for index interactions, the determined configuration is randomly shuffled in search of potentially better configurations. SWIRL [167] is an index selection algorithm that applies reinforcement learning and requires a preliminary training procedure. The training effort is then traded off against runtime performance, which is especially relevant for cloud-based applications. In contrast to other solutions

that have to interactively test multiple configurations to find a suitable configuration, SWIRL learns which indexes are beneficial under what circumstances in a training phase.

Caprara et al. [44] formulated the index selection problem as an integer linear programming model. For a given workload, a set of index candidates, and a total amount of memory available for secondary indexes, it selects a set of indexes that minimizes the execution time under consideration of index candidate-specific maintenance costs. To achieve a manageable model complexity, they restrict that for every query, at most, one index is used to retrieve the tuples. Thus, opportunities that arise when multiple indexes exist simultaneously are not taken into account. Another linear programming-based index advisor is CoPhy, developed by Dash et al. [73]. In contrast to the approach presented by Caprara et al., CoPhy supports the utilization of multiple indexes per query and incorporates multiple query plans potentially chosen by the query optimizer depending on different index configurations. What-if cost estimations are used to determine the costs of executing a query using a given index configuration and maintaining the index configuration. Furthermore, the model allows additional constraints, such as index constraints (e.g., limiting the size, contained columns, and column width) or performance constraints (e.g., a specific speed up compared to an initial index configuration). An exhaustive evaluation of different index selection approaches was done by Kossmann et al. [166].

All these approaches determine optimized index configurations, but they do not consider the possibility of different compression techniques to minimize the data footprint and create further space for auxiliary data structures. In this context, the definition of memory budgets reserved for auxiliary data structures is challenging for DBAs. Additionally, various approaches use heuristics based on the large solution space (the number of attributes and attributes per index), where further optimization approaches can be considered that leverage the limited number of index candidates for spatio-temporal data.

3.2.3 Data Tiering Decisions

This section focuses on data placement strategies for modern multi-tier DBMS architectures. In this context, we distinguish between caching and tiering strategies. The difference between these approaches is that caching strategies maintain a buffer cache to overcome capacity limitations by loading temporary copies of the data stored on secondary storage devices (e.g., HDD or SSD) into DRAM if accessed. In contrast, tiering strategies migrate the original data between multiple tiers [139].

In general, relational database systems are designed for a specific storage class and do not cope well with multitudes of storage devices [348]. For instance, traditional DBMSs such as PostgresSQL [316] and MySQL are optimized for HDD as the primary storage class and maintain only a small buffer cache in DRAM to mitigate the HDD latency and capacity limitations of DRAM. There are different eviction strategies to efficiently maintain the buffer cache and minimize cost-intensive page misses [63, 185]. Based on the workload characteristics of modern database systems (e.g., OLAP workloads with large, sequential table scans), buffer management approaches applied by operating systems (e.g., last recently used) are insufficient, as operation system-specific access characteristics differ significantly [62]. Consequently, different DBMSs choose to im-

plement a DBMS-managed buffer pool in user space [313]. Furthermore, various approaches have been developed and optimized for specific storage classes to address the characteristics of the storage classes (e.g., capacity limitations and latency differences). For instance, there are approaches to cache data stored on SSD in DRAM [151, 152, 340] or to avoid time-consuming data access on HDD by maintaining a cache on SSD [22, 142]. Additionally, there are hybrid strategies that support multiple storage classes [206, 212]. With increasing DRAM capacities, in-memory databases, such as SAP HANA and HyPer, were developed that store the entire data in main memory. These database systems abolished the buffer cache as existing implementations cause an unneglectable overhead even for data stored in DRAM [132]. By removing the buffer cache, the in-memory databases lose the capability to store data that exceed the available main memory resources and to store data on cheaper storage devices. Consequently, new approaches have to be developed to be cost-efficient [208].

In contrast to caching strategies, DeBrabant et al. [76] introduced the concept of anti-caching. The major difference between these approaches is the primary storage location. Rather than storing data on slow SSDs or HDDs and maintaining a buffer cache to profit from faster access latencies to cached data, anti-caching stores the data primarily on DRAM and evicts less frequently accessed data, called cold data, to slower storage tiers. This approach enables the access of memory-resident data without the buffer cache abstraction layer. However, the problem that arises is which data should be kept in DRAM and which data should be evicted.

The original implementation of the anti-caching concept of DeBrabant et al. was implemented for the distributed main memory-optimized row-oriented research database H-Store [156], which is the foundation for the commercial database VoltDB [317]. To decide which data is classified as cold data and consequently evicted, H-Store maintains an LRU (least recently used) list on row granularity. Every time a row is accessed, it is removed from its current position and appended at the end of the list. Based on a user-defined memory budget, the least recently used rows of the list are evicted to HDD. In contrast to row-oriented databases, it is significantly more complex in the columnar data layout to migrate data on the row level. Instead, we focus our data tiering decisions on segment granularity (cf. Section 2.3.2).

Based on distinguishing between hot and cold data, Stoica and Ailamaki [312] developed a strategy to reorganize cold data so that the OS can efficiently page it out. Another implementation of the anti-caching strategy was presented by Eldawy et al. [93]. The system, called Siberia, adds a framework for managing cold data in the Microsoft Hekaton main-memory database engine [78]. Siberia provides the capabilities to migrate cold data to secondary storage as well as to access and manipulate hot and cold data. By analyzing log data, it uses an exponential smoothing method to estimate record access frequencies. Based on this data, a backward algorithm classifies a number of records as hot, which have to be stored in main memory [191]. Furthermore, there are different systems that enable database administrators to choose the storage location on the table level. Table-granular data placement decisions often waste fast storage space as also less frequently accessed parts of the table occupy memory [348].

SAP HANA [98, 225, 263] is a commercial in-memory database system that uses a main/delta architecture to achieve high throughput rates for analytical

as well as transactional workloads. In the main/delta architecture, each table is horizontally divided into a read-optimized main storage and a write-optimized delta storage [307]. Periodically, the delta storage is merged into the main storage [264]. By default, all data stored in SAP HANA is memory-resident and can be directly accessed without an abstraction layer. To substantially increase the database capacity and allow the system to scale beyond the available DRAM resources, SAP HANA supports different data eviction approaches [303]. The first one is called column-loadable data. Based on a maintained LRU list, the database migrates the least recently used columns from DRAM in cases where are predefined memory consumption limit is exceeded. As this setting applies to entire columns, it does not allow for fine-grained tiering decisions [84]. With the Native Storage Extension (NSE), Sherkat et al. introduce a second one, called page-loadable data. This approach extends the database by an elastic DRAM buffer cache with a configurable size limit and modifies the underlying data structures to make them partially accessible on a per-page basis [303, 304]. In contrast to traditional buffer cache implementations, the buffer cache is only used for data that is marked as page-loadable. All data that is not classified by the DBA as page-loadable remains in the main memory and is accessed without the buffer cache indirection. This is possible based on a unified persistence format for in-memory processing and paged access [303]. Additionally, Andrei et al. [15] developed another tiering approach in SAP HANA for non-volatile memory (NVM). In this approach, the read-optimized main storage is stored by default on NVM and the delta storage on DRAM. For individual tables, columns, or partitions, the DBA can define exceptions and force the database to keep the data structures in main memory.

A drawback of these three approaches is that the tiering decisions have to be made manually by the DBA. For the classification of page-loadable data, the NSE provides a Load Unit Advisor [303]. Based on a metric called scan density, the advisor should help the DBA to identify objects (tables, partitions, or columns) that are suitable to be marked as page-loadable or column-loadable. The scan density is determined for an object based on the ratio of the access scan count and the object's memory consumption. In this context, the recommendations are based on thresholds defined by system parameters.

Lasch et al. [180] presented an approach based on SAP HANA to determine data placement configurations for the two storage classes, DRAM and persistent memory. The method uses a mixed-integer linear model to select a placement configuration on column granularity. In this context, each column consists of three data structures: data vector, dictionary, and index. Each data structure of a column can be placed independently on DRAM or persistent memory. The optimization approach provides two different modes (budget and target) with different objectives. In the budget mode, the DBA defines a maximal additional CPU time (compared to all data stored in DRAM) for a given workload and the model maximizes the data volume placed on persistent. In the target mode, the DBA defines the data size that should be stored on persistent memory and the model minimizes the additional CPU time. To model the performance differences between DRAM and persistent memory, a cost function is used based on the estimated number of cache misses and the total additional CPU time determined individually for each data structure and different access types. For the additional CPU time calculation, the system executes a set of microbenchmark queries. A disadvantage of the approach is that other configuration decisions

have been made before the execution of the microbenchmark (e.g., selected compression technique) or the model execution (e.g., if an index is applied or not). As spatio-temporal access characteristics vary for different partitions of a column, optimizations on column level are less effective.

Leanstore [189] is a storage engine developed by Leis et al. that implements an optimized buffer cache that overcomes the inefficiencies of traditional buffer managers. For cache eviction, Leanstore uses a speculative second chance algorithm. Based on the Leanstore's storage manager, Neumann and Freitag developed Umbra [237], a DBMS that stores data primarily on SSD and maintains a large in-memory buffer cache. In contrast to Leanstore, Umbra supports variable-size pages, which makes buffer management more complex. An alternative storage engine for Umbra to handle multiple storage devices, called Mosaic, is presented by Vogel et al. [348]. Mosaic is a storage engine for scan-heavy workloads on relational database systems. It introduces a linear optimization approach to find data placement configurations that maximize I/O throughput for a given workload and budget. The *capacity mode* of the presented linear optimization strategy (LOPT), which is an LP-based data placement strategy, places data on SSD and HDD with column granularity and considers different compression algorithms for each device. In this approach, the optimization is done on columnar granularity and does not consider indexing or sorting decisions. Furthermore, Mosaic requires that for each device, only one encoding can be applied. This encoding has to be defined by the DBA in advance. As it is optimized for data placement decisions for SSD and HDD devices, the cost model is based on the throughput and accessed data size. An advantage of this cost model is that less detailed information about the workload (e.g., selectivity of queries) is required, and no benchmarking or calibration queries have to be executed. In *budget* mode, Mosaic provides purchase recommendations to find the best configuration subject to specific performance requirements. Like Mosaic, Guerra et al. [119] developed a general-purpose framework for dynamic tier management that includes an advisor for purchase recommendations.

Boissier et al. [35] present a linear programming-based model and a heuristic approach that evicts cold data to secondary storage. For a given DRAM budget and workload, the system determines a Pareto-optimal selection of columns that are stored in DRAM. To represent the costs to access a column in DRAM or on secondary storage, the authors use a bandwidth-centric cost model that models all accesses as scan operations with a particular selectivity. The scan costs are determined based on the column's size, the scan operation's selectivity, and a fixed cost parameter for the storage class. In contrast to other approaches and similar to our approach, detailed workload information (e.g., selectivity of scans) is used to improve the cost estimations. Moreover, an extension of the linear programming model is presented that considers reconfiguration costs. Here, a fixed cost factor is integrated to represent the migration costs of a column. Like other approaches [15, 75], Boissier et al. support only two tiers. In contrast, our approach supports multiple devices or device classes that users can add or remove without reengineering the models.

Dreseler [84] introduces an automatic tiering approach for the research database *Hyrise*. In this context, the author integrates a memory management framework that enables transparent data allocation and access on different tiers and the migration of data structures between different storage devices. Furthermore, a lightweight access tracking approach was introduced to provide access

statistics on segment granularity. The developed concepts in this work are also used in this thesis (cf. Section 2.3.2). To determine data placement decisions for each segment and a given memory budget for each tier, Dreseler used a greedy knapsack algorithm. Similar to the scan density used by NSE, a profit density is used to determine the profit of a segment based on the access frequency and the segment's memory consumption.

Due to the growth of data volumes, query engines (e.g., Apache Spark [378] or MapReduce [74]) use distributed file systems such as the Hadoop Distributed File System (HDFS) [306] or Google File System (GFS) [112] to data on different nodes. In this context, files are split into blocks, which are distributed and replicated across various nodes. Multiple approaches have been implemented to improve the placement of these blocks on nodes based on storage and workload characteristics [62, 137, 174]. OctopusFS [155] is a distributed file system based on HDFS developed by Kakoulli and Herodotou that considers a fixed set of heterogeneous storage tiers with different capacities and performance characteristics. The system applies multi-objective optimization techniques for intelligent data management decisions considering fault tolerance, data and load balancing, and throughput maximization. Based on OctopusFS, the same authors developed a machine learning-based approach to automatically tier data based on file access predictions [137]. OctopusFS and other approaches based on HDFS blocks use blocks as atomic storage units. Similar to general-purpose data placement systems [144, 351], they cannot perform domain-specific optimizations as they have no detailed information about the blocks (e.g., access types). Additionally, these approaches consider further aspects for data placement decisions (e.g., replication, load balancing, or data locality). For our approach, we focus on a single machine.

3.2.4 Joint Tuning Approaches

As workload and data variations can significantly impact the performance of the database system, database tuning and continuous adoptions are crucial. However, it is impractical for the DBAs to keep track of the system's performance all the time [58]. As the costs for DBAs are a not negligible factor in the TCO of database systems [49], self-managing systems that optimize different configurations automatically came to recent popularity [169, 252]. In this context, there are different approaches that jointly optimize various aspects of database systems (e.g., materialized views, indexes, or storage layouts). For instance, different approaches use machine learning to tune multiple knobs of database systems with the goal of optimizing an expected reward (e.g., increasing performance) [11, 87, 382, 386].

Zilo et al. [402] addressed the tuning order problem and classified the pairwise dependencies between tuning options as non-dependent, unidirectional dependent, and mutually dependent. Furthermore, the authors described a hybrid approach to define an order for the tuning process based on these dependencies. Non-dependent features can be sequentially optimized in any order. For unidirectional dependent ones, the most efficient ordering should be selected. Moreover, the authors stated that mutually dependent ones should be optimized simultaneously as long as the problem complexity allows a joint optimization. A disadvantage of this approach is that it is required to know the dependencies

between various features in advance, which is challenging and requires expensive calculations [169]. Kossmann and Schlosser [169] presented an approach to tune multiple mutually dependent features. The authors demonstrated the interdependencies of different feature tunings and presented a linear programming approach to determine an optimized tuning order for a given set of features.

The Database Engine Tuning Advisor (DTA) for Microsoft SQL Server was developed by Agrawal et al. [9]. For a given workload, the DTA determines a physical design recommendation consisting of indexes and materialized views. Moreover, the approach suggests a vertical and horizontal partitioning configuration for tables, indexes, and materialized views [10]. To estimate the effects of different tuning options, the system leverages the "what-if" analysis interface of SQL Server [51], which uses the cost model of the query optimizer to determine the performance of a query on a given configuration. To minimize the overhead on the production system, DTA provides a metadata-based approach to determine a physical database design on a test server without copying the data. Afterward, the determined configuration can be applied to the production system.

Abebe et al. [4] developed the distributed database system Proteus. Proteus is able to store data in multiple formats and storage tiers. Based on learned costs models, an adaptive storage advisor (ASA) autonomously decides on suitable row- and column-storage formats, storage tier, and whether to employ optimizations such as sorting or compression in addition to data replication and partitioning schemes. Proteus groups data items into partitions with varying sizes, which can reach from a single data cell up to an entire table, and selects for each partition a storage layout. The ASA decides whether to apply a layout change based on a calculated net benefit. The net benefit considers the estimated costs to execute the storage reconfiguration as well as the expected cost effect, which is determined based on the predicted latency difference between the current and proposed layout. To predict the latency of different operations and layouts for ongoing and predicted workloads, the system's cost functions use linear regression, non-linear regression, and neural network models. Moreover, Abebe et al. [5] presented Tiresias, a predictor designed to make decisions to optimize storage configurations in database systems. Like Proteus, Tiresias drives storage and indexing adaptation decisions by computing the expected benefit of a change under consideration of the reconfiguration costs. For the calculation of the benefit, Tiresias determines for each ongoing and predicted request the relative change in the execution latency weighted with the probability that the request arrives. It makes these predictions by collecting observed latencies and access histories. As a data layout change is based on the net benefit, which is mainly determined by the latency improvement and does not consider the memory consumption of different optimizations, a risk of this method is that a large amount of the available memory resource is used for a limited number of optimizations.

3.2.5 Storage Concepts for Timestamps

There are different approaches to improve the storing of timestamps for spatio-temporal and time series data. For applications with a fixed sample rate, methods exist that calculate the timestamp of an observed location based on the trajectory's start timestamp and the position of the observed location in the

sequence of chronologically ordered locations [275]. For this approach, only the start timestamp for each trajectory is stored, which significantly reduces the memory footprint of spatio-temporal data. However, this approach is mainly suitable for applications where the temporal component is less important. Additionally, it has problems concerning transmission errors and lossy compression approaches [239]. Another approach is delta encoding, which is applied by systems such as TrajStore [66]. Instead of storing for each point a timestamp, only the delta between two timestamps is stored. Based on the assumption that the timeframe between two observations of a moving object varies only slightly, Pelkonen et al. [255] introduced a concept to store timestamps for time series data as the delta of deltas. In this approach, the offset to the delta between two data points is stored with a variable amount of bits depending on the offset. Timestamps with the same delta can be stored with one bit, which makes the approach memory efficient. However, this storage concept is optimized for analyzing and visualizing time series data. For spatio-temporal workloads with trajectory-based and spatio-temporal range queries, additional overhead is introduced due to the computation of an observed location's timestamp based on the deltas, which has to be performed for moving objects. Wang et al. [352, 353] presented an I/P frame-based structure (cf. Section 3.1) to store trajectory data in in-memory column stores. The authors divide the data into frames of a fixed period (e.g., one minute) and create for each of these frames two table columns, which store the two-dimensional coordinates of one representative location for each moving object in this timeframe. In this approach, the determination of the timespan of one frame is challenging for use cases with varying sample rates. A drawback of this data layout is that the timeframes introduce additional uncertainty and that many columns have to be scanned for spatial range queries.

Furthermore, various algorithms optimize the compression scheme of a table, which can also be applied for timestamp columns (cf. Section 3.2.1). These approaches focus on the optimized selection of compression schemes based on data and workload characteristics but do not consider that different data layouts can be applied to store specific data types. Based on the applied data layout, columns' data characteristics can significantly change and create further optimization potentials. Jiang et al. [149] presented the PIDS, Pattern Inference Decomposed Storage, a storage method for decomposing string attributes in columnar databases. The system identifies patterns in string attributes and splits the attribute correspondingly in sub-attributes based on an unsupervised approach. Additionally, PIDS can rewrite query operations on the logical view to operate on sub-attributes to minimize I/O costs and speed up the execution time. The proposed pattern inference algorithm consists of two phases. In the splitting phase, PIDS applies three rules iteratively to discover patterns. The rules identify common symbols, sequences of the same length, and common sequences. In the second step, the algorithm removes redundant structures and merges adjacent identified sequences. PIDS stores and compress each sub-attribute independently as a column, using a dictionary and bit-pack-run-length hybrid encoding. All entries that the determined pattern could not match are stored separately. A limitation of the presented system is that the splitting of attributes into sub-attributes takes only into account the column's stored entries and general data properties (e.g., the ratio of distinct values). Other factors, such as workload characteristics or the impact of different compression approaches, are not considered. Furthermore, all sub-attributes are compressed

with the same encoding approach. Based on the resulting access pattern and data characteristics of the sub-attribute, further optimizations of the storage layout could be applied.

3.3 Summary

In this chapter, we briefly discussed different existing approaches for data management systems optimized for spatio-temporal data as well as workload-driven and data-aware database optimizations. We investigated various standalone data management systems for spatial and spatio-temporal data (cf. Section 3.1). Afterward, we presented that different general-purpose database systems developed specialized extensions for spatial and spatio-temporal data management. Based on the large data volumes and response times requirements in various application scenarios, the operating costs of such systems can be significantly reduced by optimizing different tuning options provided by modern database systems. Consequently, we analyzed different approaches to optimizing the selection of database tuning options based on workload and data characteristics. We briefly discussed data compression scheme selection (cf. Section 3.2.1), index selection (cf. Section 3.2.2), and data placement (cf. Section 3.2.3) approaches. Furthermore, we analyzed tuning approaches to jointly optimize multiple tuning configurations (cf. Section 3.2.4) and data type-specific optimizations for storing timestamps in databases (cf. Section 3.2.5). To the best of our knowledge, no spatio-temporal storage system applies a joint optimization approach to determine workload-driven fine-grained storage configurations that consider the given memory budget of different storage devices.

4

Optimizing Passenger Dispatch Decisions of Transportation Network Companies

In this chapter, we introduce an approach to optimize the dispatch decisions of transportation network companies (TNCs) based on the analysis of spatio-temporal data. First, we describe the limitations of state-of-the-art dispatch processes based on the last observed location of a driver in Section 4.1. Afterward, we present the concept of probabilistic location predictions to improve the dispatch decisions. In Section 4.2, we briefly describe the location prediction algorithm that we developed and present an evaluation of the approach based on a real-world dataset of a TNC. The evaluation in Section 4.3 shows that the algorithm is able to predict the directions of drivers accurately. In Section 4.4, we explain different dispatch strategies based on the determined probabilistic locations before we discuss and summarize the results in Section 4.5. Parts of the presented content have been published [279, 284, 285].

4.1 Improving Dispatch Decisions by Probabilistic Location Predictions

The demand for ride-hailing and ride-sharing services has rapidly increased over the last few years. TNCs such as Uber or Lyft offer a peer-to-peer ride-hailing service by connecting vehicle drivers with passengers to provide flexible and on-demand transportation [222]. The application of efficient dispatching strategies that assign drivers to passenger requests in real-time is crucial for these service providers [7]. To dispatch incoming passenger requests, different policies are applied to optimize specific objective functions (e.g., minimizing the overall travel time, reducing the overall waiting time, optimizing the utilization of available resources, or increasing customer and driver satisfaction) [266, 369]. Besides other criteria (e.g., customer status), spatio-temporal cost functions are an integral part of state-of-the-art dispatching algorithms. Based on the current position of all available drivers and the passenger, geographical distance or estimated time of arrival metrics are calculated [199]. Consequently, the quality of the decisions is strongly affected by the accuracy of the observed and transmitted locations of the drivers. For that reason, it is necessary to have exact location information or accurate location predictions.

Additionally, the passenger's travel and expected waiting times are determined based on the location of the selected driver by traffic-adjusted routing

services. The estimated waiting time, which is communicated initially to the customer, has to be accurate as the cancellation rate strongly increases with the waiting time. High cancellation rates reflect unsatisfied passengers leading to a drop in passenger retention rate, as the ride-hailing industry is characterized by fierce competition. Ultimately, high cancellation rates reduce the revenue of a TNC. Hence, the communicated waiting time has to be accurate. Otherwise, the passenger has to wait longer than initially communicated, which leads to an increase in the cancellation rate. By analyzing the TNC's data, we observed that passengers do not tolerate delays, as more than 50% of all delay-related cancellations happen within the first two minutes of waiting time.

Based on a manual analysis of dispatch decisions of a globally operating TNC, we discover that over one-fifth of delayed pickups are due to unplanned detours that were caused by inaccurate information about the driver's position [279]. As one cause for delayed pickup times and suboptimal dispatch decisions, we identified the inaccuracy and uncertainty of the driver's location. There are different factors like noise or technical limitations of the GPS, which influence the accuracy of the observed locations [359]. Additionally, the given sampling rate, data transfer problems, and the time consumed by the entire dispatch process affect the actuality of the spatio-temporal information. Consequently, the correct position of a driver at the time of the order assignment can deviate significantly from the last observed location, which is currently used as input to calculate the estimated travel time or distance.

4.1.1 Limitations of Status Quo Dispatch Processes

The assignment of available drivers to passenger requests is a dynamic vehicle routing problem or dial-a-ride problem. As described by Psaraftis et al. [265], the vehicle routing problem is characterized as dynamic if requests are received and updated concurrently with the determination of routes. In the setup of TNCs, incoming bookings of passengers have to be continuously assigned to an available driver considering further information, such as the current traffic situation or the availability of drivers, which are unknown in advance. To optimize the time-critical dispatch process, TNCs usually define policies that use specific cost functions (e.g., minimum travel time or distance) to improve a specific objective function (e.g., minimize the overall waiting time) [266]. As inputs, most of these cost functions use the location of the passenger and the locations of a set of available drivers. A common approach is the nearest vehicle dispatch used by various ride-hailing services [153]. Here, the driver with the shortest travel time to the pickup location is assigned. Based on the locations, the travel time is determined by using services that offer traffic-adjusted routing services (e.g., Google Maps) to incorporate the current traffic situation.

Consequently, accurate calculations require precise and up-to-date location information. To demonstrate the limitations of dispatch decisions based on the last observed location, we use the dispatching example depicted in Figure 4.1 to exemplify the implications of the inaccuracy and uncertainty of a driver's current location at the time of dispatch. The example shows a dispatching scenario on a highway, where the upper-right user pin represents the passenger's pickup location and the car pins represent a single driver's potential GPS location. While the dotted marker represents the driver's last observed location (which

the dispatching algorithm uses), the solid markers represent the driver's possible locations at the time of dispatch.

There are different factors like noise or technical limitations of GPS-based systems [359], which affect the recorded location of a driver and can lead to coordinates that are not directly assignable to position on a road segment. In the example, the driver's last recorded position is between the two highway lanes. Based on the last observed location, it is difficult for the dispatching algorithm to determine the driver's direction. Here we can assume that the driver is in the right lane. However, suppose the driver's correct location is A. In that case, the actual travel time can be much higher than its estimated counterpart, as turns on highways are impossible and the next exit may be far away.

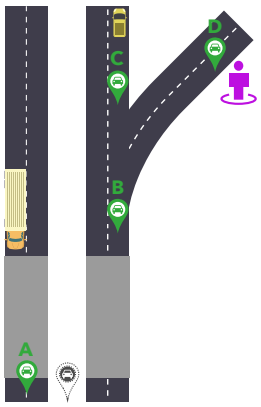

Fig. 4.1: An example highlighting the implications of the driver's current location's inaccuracy and uncertainty. The dotted location marker between the two highway lanes depicts the last recorded location. The other markers indicate a driver's possible current locations on the two roads.

Even when on the right side of the street, the driver's location at the time of dispatch relative to the necessary highway exit is unknown: the driver may have or may not have taken the exit (location D and C) or the driver may not have reached the exit (location B). The actual travel time varies significantly with locations B, C, or D, as missed exists on highways are costly in terms of time. Consequently, there is a high risk of delay. In addition, the driver's last recorded location may be older than the sampling rate indicates, as urban effects such as tunnels may prevent the transmission of GPS signals. Also, the entire process of assigning a driver and the acknowledgment of the driver takes several seconds, where the position of the driver is continuously changing. As shown by the example, the actual position of a driver at the time of the order assignment can deviate significantly from the last observed location and consequently influence the determined value of the cost function. Therefore, the dispatching algorithm has to decide based on incorrect information, for which reason it may not assign the optimal driver to a requesting passenger, and also, the driver could arrive delayed at the pickup location.

In this context, we introduce the concept of *detoured dispatches* and *risky dispatches*. We classify a dispatch as a detoured dispatch if the assigned driver's

arrival at the pickup location is delayed due to an initial detour of the driver. Based on this definition, we define a dispatch as risky if the dispatched driver's arrival at the pickup location is likely to be delayed due to uncertainty about the current position of a driver, which may lead to an initial detour or a sub-optimal route. To evaluate the share of delays caused by detoured dispatches, we manually analyzed a sample of 500 dispatch decisions. The dispatch processes were randomly selected from a real-world dataset of a TNC, which includes the bookings and the spatio-temporal data of Dubai, spanning from November 2018 to February 2019. Further, we limited the analysis to dispatch processes where the driver arrived at the pickup location between one and five minutes delayed. We identified that in about 20 percent of the delayed arrivals, the driver performed an initial detour based on the random sample.

For that reason, it is necessary to develop new strategies to improve the accuracy of the applied spatio-temporal cost functions by precise predictions of the exact drivers' locations. Furthermore, the dispatching algorithms should be enhanced and optimized by incorporating the risk factor for delays based on inaccurate positional information.

4.1.2 Dispatch Decisions Based on Probabilistic Location Predictions

To overcome the limitations of drivers' assignments based on the last observed locations, we propose a novel approach that uses probabilistic location predictions. Here, we determine the potential locations of a driver with the corresponding probabilities at the time the order is assigned based on previously observed spatio-temporal data and the current trajectory of a driver. Here, we can leverage that traffic repeats itself [338]. Additionally, drivers of TNCs often try to stay in specific profitable areas to increase the chances of getting further trips [286]. To incorporate the location probabilities into the dispatch process, the cost functions must be adopted consequently.

An example of how probabilistic location prediction can influence assignment decisions is shown in Figure 4.2. The black marker represents the pick-up location and the blue, green, and orange markers the last observed map-matched location of three available drivers. A traditional dispatching algorithm that uses a specific cost function (e.g., shortest distance or shortest travel time) would assign the booking request to the blue driver based on the last observed locations. By analyzing the predicted potential positions of the drivers, we can see that the blue and green drivers are likely to move away from the passenger's location. In contrast, the orange driver is directly driving in the direction of the passenger. Therefore, it is highly likely that the orange driver would be the best option for the dispatch algorithm. In this example, we demonstrate that by including the driving behavior and direction of drivers, the result of the dispatch algorithm can change. Additionally, we can immediately detect whether the estimated arrival time of a particular driver (e.g., the blue driver in Figure 4.2) would be too optimistic and whether detours and, in turn, critical delays are likely.

In the second example (see Figure 4.3), we demonstrate the impact of the probabilities calculated based on observed patterns in past drives. The size of the dots represents the probability of the corresponding location. The larger a dot is, the higher the probability of the location. Similar to the first example (see Figure 4.3), the blue driver has the shortest distance and seemingly the

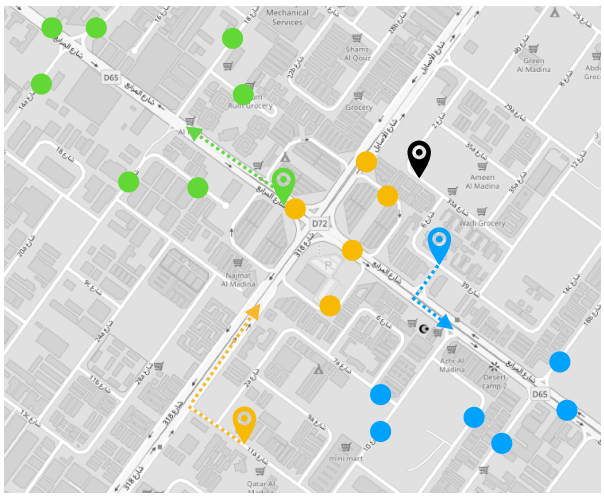

Fig. 4.2: Predicting potential current locations of candidate drivers to be assigned to a waiting customer (black marker): Example of three different drivers (green, blue, orange marker). The dots represent the predicted potential next locations of each driver based on their driving behavior.

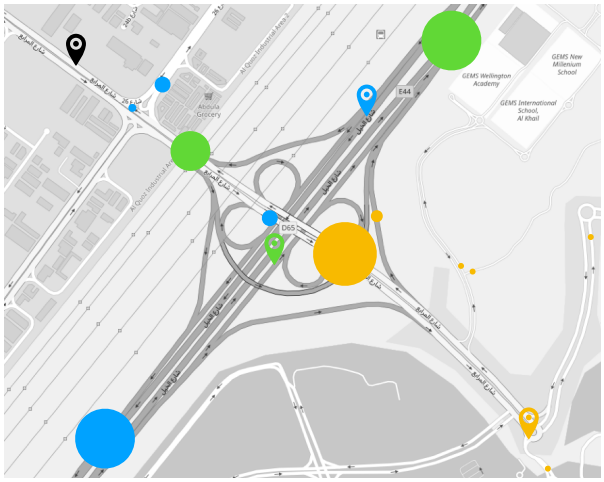

Fig. 4.3: Improving dispatch decisions using probability distributions for the current locations of potential drivers: Comparing the likelihood of a driver reaching the customer (black marker) without critical delays. Example of three different drivers (green, blue, orange marker). The dots represent the predicted next locations of each driver (the larger the dot is, the higher the probability of the location).

shortest travel time to the pick-up location. But the big dot in the left-bottom corner indicates that there is a high chance that the blue driver misses the exit. Consequently, it may be preferable to assign the trip to another driver. The green driver has a higher probability of being on the shortest route to the pick-up location, but also, there is a not negligible probability that the driver stays

on the highway and needs to perform a costly detour to reach the location of the passenger. Based on the last observed location, the orange driver has the longest distance to the pick-up location, but the predicted probabilities show that she is highly likely to drive in the direction of the pick-up location. Consequently, assigning the order to the orange driver is potentially not the optimal decision, but the one with the lower risk of delays.

Our proposed approach enables TNCs to apply dispatching strategies that take risk considerations into account. Whether to optimize expected arrival times, worst-case scenarios, or other risk-aware criteria can be strategically determined by the companies. However, accurate location probabilities are key for such risk-aware dispatching strategies. By applying risk-averse dispatching policies, TNCs can optimize the request dispatching and avoid unnecessary detours of their drivers.

4.1.3 Approaches to Predict the Locations of Drivers

Route prediction algorithms can be separated into long-term and short-term route prediction algorithms. Long-term route prediction approaches forecast drivers' entire route to their final destination, whereas short-term route prediction algorithms predict only a fraction of the remaining route a driver can drive within a provided prediction time. Various long-term route prediction algorithms use Hidden Markov Models (HMMs) that model a driver's intended route as a sequence of hidden states since drivers' intentions can only be observed indirectly by the driven routes [181, 308, 371]. Simmons et al. [308] use an HMM that models the road segment, destination pairs as hidden states, and the GPS data as observable states. While Simmons et al. do not require a separate map-matching step, Ye et al. [371] require one, as their HMM models the driven road segment as observable states, while clusters of routes serve as hidden states. Other approaches use clustering techniques to group similar trajectories into clusters so that deviations of the current trajectories to past trajectories are more tolerated [107, 181]. Lassoued et al. [181] hierarchically cluster trajectories via two different similarity metrics: same destination or route similarity. In this context, the route similarity metric is defined as the fraction of the shared road segments. Froehlich and Krumm [107] predict the intended route by using an elaborate route similarity function to compare the current route to a representative combination of routes of each cluster. The similarity metric depicts the distance differences between the GPS recordings of trajectory without pre-requiring a map-matching step. Further approaches use machine learning techniques, such as reinforcement learning [400], neural networks [230], and methods of social media analysis [371].

While long-term route prediction algorithms are helpful for the prediction of an entire route, their predictions are bound to previously observed routes. In our problem, however, the pick-up routes of individual drivers are rarely identical, as pick-up locations are not stationary, but various aspects can be used in short-term prediction. Trasarti et al. [337] use clustering techniques to extract fractions the driver is expected to be able to drive within the provided prediction time. These approaches, however, still lack the support for new unseen routes. Karimi et al. [157] predict the most probable short-term route by mining the driver's turning behavior at intersections and using the trajectories' underlying road network. They traverse the road network in depth-first fashion to find the

maximum reachable locations from the driver's current location. Additionally, they determine the traversal time of road segments by using the corresponding speed limits. This approach was extended by Jeung et al. [148] by mining the road segments' traversal time from trajectories. Both approaches require the trajectories to be map-matched, as the turn probabilities are calculated on road segment level. In contrast, Patterson et al. [251] avoid map-matching by using practical filters that incorporate the error of all random variables into one model. Additionally, dynamic short-term route algorithms exist that reconstruct their models on the fly on data changes. These approaches acknowledge the dynamic nature of traffic and moving objects, whose environment changes aperiodically. Zhou et al. [397] continuously evict patterns from outdated observed trajectories so that the applied models only consider data from the most recent trajectories.

In addition, there are different turning behavior predictions, which model drivers' turning behavior as a Markov process [148, 157, 175, 205, 251, 401]. These approaches are similar in the way they model the turning behavior at intersections as Markov chains, in which the states represent road segments, and drivers' decisions indicate their transitions at intersections. They differ, however, in the order of the Markov chain, i.e., the number of past road segments they consider. While some consider only the last driven road segment to be an indicator for the next turn [148, 157, 205, 251], Krumm [175] proposes the usage of an n^{th}-order Markov chain, in which the next road segment is predicted by following the last n driven road segments as states in the Markov chain. The evaluation shows that the accuracy of the determined turning behavior predictions increases with the number of road segments n considered in the forecast. However, with the increasing order of the Markov chain, fewer sequences of driven road segments are observed as the state space (cf. number of segment combinations) increases exponentially. Also, they experimented with inferring if the result's accuracy is sensitive to context information, such as time of day or day of the week. However, they did not find such sensitivity, as the fraction of matched road segment sequences of the given context was small due to the training dataset's size.

Ziebart et al. [401] model the turning behavior of drivers via a Markov decision process whose cost weights of actions are learned via inverse reinforcement learning using context-aware and road-specific features. Further approaches analyze the speed and acceleration profiles of drivers to predict the turning behavior at an upcoming intersection. Liebner et al. [200] cluster speeding profiles using k-means to predict a driver's turning behavior at a single intersection. Phillips et al. [259] and Zyner et al. [405] use short-term memory neural networks to predict the turning behavior. These approaches use different context-sensitive turning probabilities to improve the prediction accuracy but do not incorporate the determined probabilities into further analysis or decision processes.

4.2 Probabilistic Location Prediction Algorithm

To minimize detoured dispatches and enable risk-aware decisions, we propose an approach to predict probabilities of future driver positions based on patterns observed in past trajectories. In this context, we aim to predict the possible locations of dispatching candidates at the time of the trip assignment. Based on the applied dispatching policy, the algorithm calculates the cost function for the

candidates (e.g., estimated travel time) as a combination of the cost function values for various possible locations (see Section 4.4). Mining driving patterns and predicting possible locations allows more precise estimation of pickup times, leading to shorter waits, despite the inherent uncertainty and inaccuracy of a driver's current position. The algorithm uses repeating driving patterns from all drivers that can be generalized to apply them to forecast upcoming driving behaviors.

As we forecast a driver's next locations around the time of dispatch, we constrain the analysis to free-time trajectories. In free-time trajectories, drivers are generally not influenced by external factors and drive freely around to get assigned to incoming passenger requests. Drivers are unaware of a request until it is communicated to them. Consequently, at the time of dispatch, drivers drive freely around, and hence, their decisions are similar to those they have made in past free-time trajectories. The analysis of trajectories also allows us to extract information on the dynamic characteristics of the road network, such as traffic. Traffic affects drivers' traversal times on road segments, and hence we need to incorporate this into the location prediction to ensure accuracy. As traffic repeats itself [338], we can consequently use historical traffic patterns to forecast future traversal times on road segments.

Overall, the prediction algorithm consists of four parts: (i) data preprocessing, (ii) road segment candidates determination, (iii) route probability calculation, and (iv) location prediction.

4.2.1 Spatio-Temporal Data Preprocessing

The process starts with the data preprocessing step, in which we segment the trajectories in sub-trajectories representing distinct driving sessions, classify the sub-trajectories based on the occupancy state, and map-match the observed locations. To divide the spatio-temporal data of a driver into different driving sessions, we apply a time-based approach. We defined a time limit of ten minutes. The trajectory is segmented into sub-trajectories if (i) the driver's position does not change in the time limit, (ii) no further data is tracked in the time limit, or (iii) the occupancy state of the driver changes.

Afterward, we classify each sub-trajectory based on the occupancy state. Here, we distinguish between the two states, free and occupied. Depending on the occupancy state, a driver's driving behavior changes significantly. Suppose the driver is transporting passengers or is on the way to the pickup location of a passenger. In that case, drivers use the shortest route based on the current position, the destination, and the traffic situation. Routing services often suggest these routes. In contrast, drivers with the occupancy state free are freely driving around intending to get incoming bookings. Their paths depend on personal experience and individual preferences, as well as external circumstances. For that reason, we distinguish trajectories based on the occupancy state for our use case.

The transmitted coordinates of the GPS sensors have not to be located on a specific road as the data can be affected by various factors (see Section 4.1.1). Noise, technology-based limitations, and measurement errors can cause a discrepancy between the observed and the driver's exact location [359]. These inaccuracies can result in an observed location being off-road or assigned to an incorrect road segment. For that reason, a map-matching step is necessary to

ensure that each observed location is assigned to a road segment on the one hand and, on the other hand, to reduce inaccuracies and filter measurement errors [211]. In this step, we map each tracked GPS location to a specific point on a road segment of a road network. A road network is a directed multigraph that represents the real-world traffic infrastructure of a specified area along with the corresponding metadata [29]. In the graph, each node represents an intersection between at least two road segments represented by edges. These road network maps are created and maintained by humans or automatically updated by trajectory-based algorithms [135]. A road segment is a directed edge that is defined by a source and target intersection. It is associated with a list of intermediate GPS coordinates describing the segment's geography. For each road segment, meta-information such as the length and a speed limit is stored [29]. A set of connected road segments composes a road. Here, we used the data of OpenStreetMap[1].

The obvious algorithm is to match each observed location with the nearest road segment. Due to measurement noise and the dense structure of road segments in urban areas, this algorithm is prone to error. State-of-the-art map matching algorithms take into account sequences of points before deciding on a match to increase the physical plausibility of the resulting pings and the corresponding trajectory [238]. Therefore, we decided to apply an algorithm published by Newson and Krumm [238]. Their approach demonstrates high accuracy despite high noise in GPS measurement as well as a low sampling rate and is widely used. The map-matching algorithm utilizes an HMM, in which a system's hidden state is only observable indirectly via measurements over time. In the scenario, a driver's movement in the road network, i.e., a sequence of road segments, is implicitly observed via a sequence of GPS measurements. The used HMM is configured to model this characteristic. Consequently, the HMM derives the most likely sequence of hidden states, i.e., driven road segments, from observed measurements. As suggested in related work [107, 238], we applied filters to remove outliers based on traversal speed and acceleration thresholds. We set the thresholds according to physical plausibility, i.e., it is doubtful to surpass the thresholds in the given road network via street-legal cars. The speed threshold is set to 200 $\mathrm{km\,h^{-1}}$ because the used road networks have maximum speed limits of 120 $\mathrm{km\,h^{-1}}$. Additionally, we set the acceleration threshold to 12 $\mathrm{m\,s^{-2}}$.

Based on the map-matched locations and the assigned route segments, we define a route as a temporally ordered sequence of connected road segments of a driver in a limited time interval. Due to the GPS sensors' sample rate, it can occur that the assigned road segments of two consecutive locations are discontiguous. If the resulting road segments are not connected, the corresponding road segments to connect the segments by the shortest path are added to the route.

4.2.2 Identification of Potential Locations

After the data preprocessing step, we determine a set of road segment candidates that includes all reachable road segments of a driver based on the driver's last observed position and the prediction frame (see Figure 4.2). As prediction frame

[1] `https://www.openstreetmap.org`

f, we define the time difference between the timestamp of the last observed location and the expected time of the trip assignment. Instead of considering just all road segments in the neighborhood, this approach restricts the set of candidates that include all potential reachable ones to a small number of road segments. Due to this step, the number of candidates can be reduced for subsequent parts of the prediction algorithm, allowing for faster predictions.

Determination of Road Segment Candidates

We partially analyze the given road network to determine the relevant road segments on which the driver is estimated to be after the defined prediction frame f. Therefore, we use the road segment of the last observed location of an available driver as the starting point. First, we determine the time required for the driver to leave the current road segment based on the driver's position and the traversal time of the segment. Each road segment has an associated cost value, which depicts its traversal time (i.e., the time a driver needs to traverse it completely). In Section 4.2.2, we briefly discuss the algorithms to determine the traversal time of road segments.

Following, we determine all possible paths of the driver based on the starting point and the road network structure by summing up the traversal times of adjacent road segments until the prediction frame is exceeded. Here, we simulate all potential routes a driver could drive in reality by analyzing all potential sequences of road segments. By summing up the traversal times of the segments of a path, the algorithm calculates the time it would take the driver to drive the associated route. Since the prediction frame restricts the overall traversal time, the end of a path is reached if we can add no further road segments without exceeding the prediction frame. By definition, the algorithm expects drivers to reach the last road segment of a path.

As we assume that for all road segments of a valid path, except the last one, there is still enough time to leave the road segment in the prediction frame, we expect drivers to be located on one of these candidate road segments. Consequently, we define for a given road network's set of road segments E, the set of road segment candidates $M \subseteq E$, which contains all reachable road segments of a driver within the prediction frame $f \in \mathbb{R}_0^+$. Given the traversal time $t_{route} \colon E \to \mathbb{R}_0^+$ of the fastest route from a driver's current location to a road segment $r \in E$ and the traversal time $t_{rs} \colon E \to \mathbb{R}_0^+$, we define the set of road segment candidates $M = \{r \in E \mid t_{route}(r) \leq f \leq t_{route}(r) + t_{rs}(r)\}$.

Traversal Time of Road Segments

As already mentioned, each road segment has an associated cost value, which depicts the estimated time a driver needs to traverse a road segment completely. We calculate the traversal time of a road segment based on its length and the driver's forecasted traversal speed on it. This approach is used as we cannot derive a driver's spent time on a road segment solely from tracked GPS data since we do not know the exact time the driver enters and leaves the road segment. However, with two consecutive observed locations on the same road segment, we can calculate a driver's average traversal speed between the two observations. Therefore, we translate the problem of road segment traversal time

forecasting to the problem of forecasting traversal speeds on road segments. Some algorithms use speed limits as an estimation of a road segment's traversal speed [157, 205]. However, these fail to reflect the dynamic nature of traffic conditions on a road segment (e.g., traffic lights or congestions).

In our approach, we mine the traversal speed from past trajectories. The mined traversal speed is the average speed of all drivers on the road segment in a given timeframe (e.g., of a given hour). By mining the traversal speed on a road segment level, we implicitly incorporate traffic light phases and special traffic situations (e.g., traffic jams and slow-moving traffic). The algorithm calculates and averages the speed between consecutive locations on the same road segment and does not have a route interpolated between them. Without these constraints, a driver could have left the road segment so that the actual time spent on the road segment is unknown. As the traffic volume changes in specific areas over the day (e.g., rush hour), the daytime has an impact on the traversal speed of a road segment. In this context, we mine the traversal speed for different hours to address temporal traffic phenomena. We define a driver's traversal speed of a given time to be the traveled distance of the path between the associated observed location and its successor in the road network divided by their time difference. Unlike related work, we do not apply an approximate distance metric, such as the great circle distance [82], that neglects the underlying road network and thus fails to represent the actual traveled distance between two observed locations. Instead, we leverage the underlying road network to calculate the exact distance traveled between consecutive tracked locations. To calculate the traveled distance, we first add the fractions of the two locations that indicate what percentage of the current road segment has already been traversed by the driver, multiplied by the length of the respective road segments to the distance. As consecutive locations' road segments can be nonadjacent, we also add the length of the interpolated route to the distance.

To ensure that the mined traversal speed values are representative of the correct traversal speeds, we filter mined traversal speeds that have been observed as fewer than a defined observation support threshold. The selection of a small time frame enables the consideration of detailed traffic patterns but also strongly limits the number of observed drivers in the time frame. For that reason, we selected a time frame of one hour, which allows us to distinguish between different traffic situations and delivers a sufficient number of observations for the majority of road segments. Due to the high number of road segments, road segments exist for which we either only observe a few or no observed locations for a given hour. In cases where we have no or only a minimal number of observations, we use the driver's current speed if all road segments in the path have the same speed limit. The last fallback option is to use the speed limits.

We define the traversal speed of a driver at the observed location m_t with the timestamp t to be the driver's average speed between the observed locations m_t and the next location m_{t+1}. To calculate the average speed between the observed locations m_t and $t+1$, we divide the traveled distance $tdist(m_t, m_{t+1})$ between the two locations by their time difference. In contrast, we define the previous traversal speed at the time of the current location m_t to be the driver's average traversal speed between the driver's previous location m_{t-1} and m_t.

4.2.3 Route Probability Calculation

To determine the probability of a route, we analyze drivers' turning behavior at intersections to assess drivers' probability of taking a specific turn. These turning patterns allow the computation of the overall probability of different potential routes. By definition, intersections connect at least two road segments. Consequently, to model drivers' turning behavior at intersections, we can count the co-occurrences of road segment pairs and calculate the corresponding probabilities [157, 175, 205, 357].

We model the turn probabilities by a Markov chain of n^{th}-order, as a driver's behavior at intersections, can be represented by a sequence of events, in which the probability of each event, i.e., the decision at the current intersection, depends only on the state attained in the previous event, i.e., the decision at the previous intersection. For instance, in a 1^{st}-order Markov chain, in which the turning behavior at the current intersection is independent of the decisions made at previous intersections. In contrast, in an n^{st}-order Markov chain, the decision depends on n previous decisions at intersections. In the model, the road segments serve as states and the turn probabilities as transition probabilities between the states. Let H be a set of past free-time trajectories and E be a set of road segments. Given road segments $r_i \in E$ and $r_j \in E$, we define the support $\tau(H, r_i \succ r_j)$ as the number of trajectories in H that contain $r_i \succ r_j$. Based on that, we define the transition probability from r_i to r_j as:

$$P(r_i \succ r_j) = \frac{\tau(H, r_i \succ r_j)}{\sum_{r_k \in E} \tau(H, r_i \succ r_k)}. \tag{4.1}$$

Let R be a driver's current route with $i \in \mathbb{N}^+$ road segments, where $R(1)$ is the first and $R(i)$ is the driver's current road segment of R. We calculate the probability of a road segment $r_j \in E$ to be the next road segment given the current route's previous $n \in \mathbb{N}$, $n \leq i$, road segments by

$$P_n[R \succ r_j] = \\ P[R(i-n+1) \succ R(i-n+2) \succ \ldots \succ R(i) \succ r_j]. \tag{4.2}$$

We define the transition probabilities of the n^{th}-order Markov chain in the following by extending the support measure to be the number of trajectories that contain the route $R(i-n+1) \succ \ldots \succ R(i) \succ r_j$.

The selection of the Markov chain order is a tradeoff. Markov chains of a higher order allow us to represent the behavior of drivers better to drive around a specific area. This behavior is typical for drivers of TNCs, as particular regions are more profitable compared to others [286]. Using n^{th}-order Markov chains, the number of routes, i.e., states of the Markov chain, that do not have many observations increases since the number of possible states increases exponentially with the order. A road network contains a large number of road segments. Therefore, we may observe some road segments of the road network infrequently. States with few observations are inexpressive for drivers' turn behavior. To avoid inexpressive states, we filter states whose support is below a minimum support threshold $s_{min} \in \mathbb{N}$. Additionally, with decreasing observations per state (or a higher order of the Markov chain), the likelihood of overfitting increases. While we can better model the turning probabilities of past free-time trajectories with

higher order, we perform worse when applying the higher-order Markov chain to new trajectories because we may not have observed some states due to the exponentially increasing number of Markov states.

We calculate the probability of each location by calculating the probability of its associate route. We compute the probability $P_{route}[\hat{R}]$ of a predicted route $\hat{R}$ by applying the turn probabilities P_n of an n^{th}-order Markov chain that we mined from past trajectories to the turns of the predicted route

$$P_{route}[\hat{R}] = \prod_{i \in \mathbb{N}: 0 < i \leq |\hat{R}|} P_n[R(i-n-1) \succ \ldots \succ R(i-1) \succ R(i)]. \tag{4.3}$$

Similar to the construction of the Markov chain, we analyze each route with a sliding window size equivalent to the chains order. In each step, we match the state, extract the next road segments transition probability, and multiply it with the previously obtained probabilities. The first window starts with the last $n \in \mathbb{N}^+$ road segments directly preceding the predicted route. Due to the size of possible co-occurrences and the state support filter, we may be unable to match a state of the predicted route to the constructed Markov chain. We need to resolve these missing states in order to calculate the probability of the predicted route.

There are a couple of strategies in research to resolve missing states. Jeung et al. [148] introduce a concept called reverse mobility statistics, which assumes that drivers follow the new route back to their origin after driving a new route due to drivers tendency to drive familiar routes. Therefore, they assign missing states the same probabilities as their observed reversed counterparts. While this may be true for drivers with a fixed origin, e.g., someone leaving their home to visit a new place, this is not necessarily the case for drivers in the context of a TNC since they do not drive back to their origin after a booking. Because we only double the number of observed states at max, we may still not resolve many missing states. Thus, we apply a different strategy.

We resolve the missing state by distributing its transition probabilities uniformly. This strategy is both neutral and universally applicable to all missing states. Therefore, the algorithm can also handle missing states and thus can also predict locations on unprecedented road segments that it would have otherwise filtered out because their associated probability is zero. To predict turning behaviors at intersections precisely, we need to know the historical sequence of connected road segments between consecutive observed locations. In the context of higher sampling rates, it is possible that a driver has traversed a road segment entirely between two locations and the turn probability calculation algorithm wrongly attributes the latter's road segment to be adjacent to the former's road segment. Consequently, we count the occurrence of the nonadjacent road segment instead of the traversed road segment leading to mistakenly lower turn probabilities.

During the route interpolation of two consecutive locations, we are confronted with an abundance of possible routes the driver could have taken. We use Dijkstra's shortest path algorithm to identify the route with the shortest traversal times between two consecutive locations. Traffic situations and the demand vary for specific regions depending on the daytime [286]. For that rea-

son, we distinguish between different contexts (e.g., rush hour) to increase the accuracy of the turn probability prediction.

4.2.4 Location Extrapolation on Road Segment Candidates

In the final step, we extrapolate a driver's specific location on the determined road segments, as the estimated time to the passenger can vary based on the particular location on a road segment. During the short-term route prediction (cf. Section 4.2.2), we calculate a set of road segment candidates that a driver is expected to reach within the prediction frame f. We determine for each candidate road segment a driver's required traversal time $t_{path} \in \mathbb{R}_0^+$ to reach it. As the remaining time $t_{remaining}$ of each candidate road segment, i.e., $t_{remaining} = f - t_{path}$, is not large enough to traverse it completely, we expected the driver to be located on it. Given each candidate's remaining traversal time $t_{remaining}$, we estimate the driver's precise position via the fraction of the road segment the driver is expected to have traversed within the remaining time of the prediction frame.

4.3 Evaluation of Location Prediction Algorithm

This section evaluates the applicability of the presented location prediction algorithm based on the real-world dataset of a TNC. First, we introduce the dataset and explain the data characteristics (Section 4.3.1). Afterward, we analyze the accuracy (Section 4.3.2) and the computation time (Section 4.3.3) of the approach.

4.3.1 Dataset

For the evaluation, we use a real-world dataset of a TNC. The dataset consists of the observed locations of drivers and booking information in the City of Dubai. It includes the data for three consecutive months, from November 2018 to February 2019. Moreover, the trajectory data have a raw size of about 15.9 GB and store 400 million GPS-tracked location information. Due to privacy concerns, the data involve only dispatch process-related positional information and no details about the routes of passengers. Compared to other publicly available passenger transportation datasets (e.g., NYC Taxi Rides [332]), the dataset has a significantly higher granularity as a driver's position is tracked around every five seconds. Besides the timestamp, latitude, longitude, and the driver's identifier, a status attribute is tracked for each observed location. This status attribute represents the driver's status (free or occupied). In this context, the status free indicates that a driver is available for incoming passenger requests, and the status occupied defines that the driver is on the way to the pickup location of an assigned passenger request. All attributes are stored as integers. Based on the insertion order, a certain temporal ordering of the sample points exists. Still, we cannot guarantee that the timestamp column is sorted due to transmission problems and delayed transmissions (cf. Section 4.1.1).

The corresponding OpenStreetMap road network of the City of Dubai has 139 117 road segments with an average length of 115m. The length of road segments can vary based on the road type (e.g., highway).

4.3.2 Accuracy of the Predicted Locations

To evaluate the overall quality of the location prediction algorithm (cf. Section 4.2), we use 1 000 pings located on the same road segment to predict the road segments the associated drivers could be on after the prediction frame, i.e., the road segment candidates, along with their respective probabilities. As ground truth serves the driver's correct road segment after the prediction frame f, which defines the period, we try to predict based on the timestamp of the last observed location. We compare the discrete probability distribution of these predicted road segments with the discrete relative frequency distribution of the drivers' correct road segments. We model the turning behavior via 2^{nd}-order Markov chains. For the experiments, we set the prediction frame f to 5 seconds, 10 seconds, and 20 seconds and compare the results of a highly frequented road segment, i.e., one with many observations, to a less frequented road segment to evaluate how the algorithm performs with a smaller set of observations.

We perform out-of-sample four-fold cross-validation for all experiments and report the average score over all four runs. In Figure 4.4, we illustrate the results of our location prediction algorithm for drivers that are currently on a *highly frequented* road segment. The training dataset for the algorithm includes 21 751 traversal speed and 9 224 turn observations for the respective road segments.

The predicted probability density over the set of road segment candidates is similar to the distribution of the relative frequencies of the individual road segments of the ground truth. The average absolute difference between the probability of a predicted road segment and its relative frequency in the ground truth for the three prediction frames are 0.012 (5 seconds), 0.033 (10 seconds), and 0.075 (20 seconds). For a prediction frame of 5 seconds, the predicted probability deviates on average by 1.2% from the actual relative frequency. The difference proves that the location prediction algorithm is accurate for frequently observed road segments. As the prediction frame increases, the difference between the predicted probabilities of the road segments and their actual relative frequencies increases. The reason for this is that with an increasing prediction frame, the impact of the estimated traversal speeds' inaccuracies increases. The imprecision of the estimation may be caused by temporary traffic conditions that the mined traversal speed estimations do not capture in full detail.

We conduct the same experiments on a *less frequented* road segment with fewer observations. Figure 4.4 shows our algorithm's results for a representative *unfrequented* road segment. For the road segment, we obtained 2 210 traversal speed and 3 977 turn observations during the trajectory data analysis process.

The results show that the set of road segment candidates the location prediction algorithm predicts overlaps on average with 63.5% of the ground truth across prediction times. Although the ground truth contains every member of the set of candidates, we miss on average 36.5% of the ground truth's road segments because we did not observe two turns during the trajectory analysis process. We may have missed these turns during the observation due to temporary road conditions, e.g., road constructions blocking the turns. However, the relative frequencies of the turn's following road segments are low, for which reason these turns are generally infrequent. Finally, the missed turns account on average for 0.4% of road segments within the ground truth, which is a negligible amount. The average absolute difference of the predicted road segments' prob-

abilities to their relative frequencies in the ground truth for the less frequented road segment is 0.039 (5 seconds), 0.053 (10 seconds), and 0.031 (20 seconds).

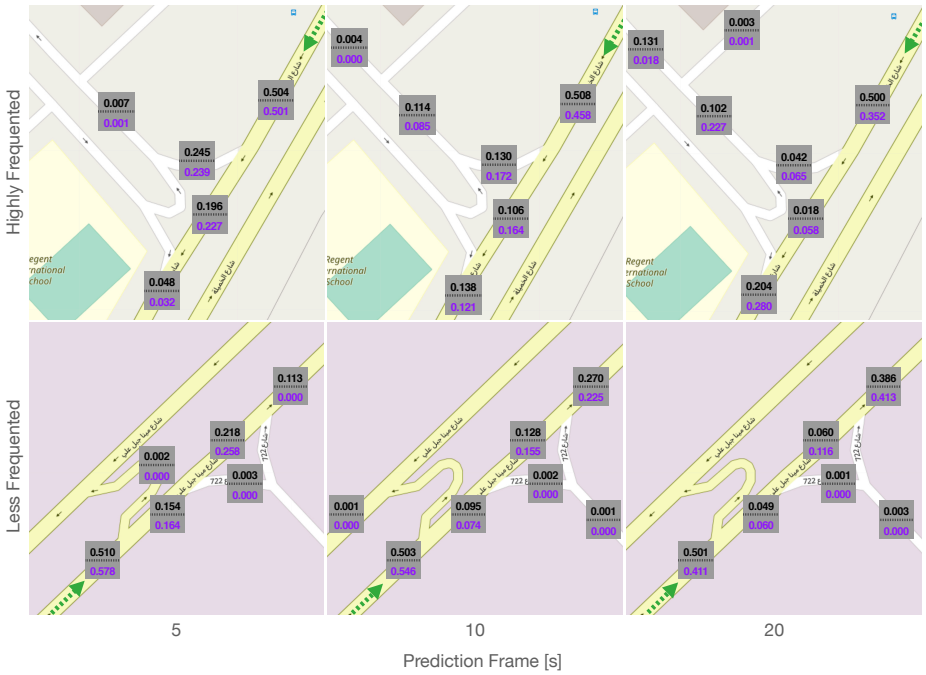

Fig. 4.4: Results of the next location prediction algorithm for a *highly frequented* and a *less frequented* road segment. We run the experiment on 1 000 out-of-sample observed locations that share the same road segment indicated by the dashed green arrow. The upper value in the box denotes the relative frequency of drivers that are on the respective road segment after the prediction time. In contrast, the value below depicts the probability we predict for drivers to be on that road segment after the prediction time.

Furthermore, we conducted additional location predictions for different examples. Naturally, we found that the results depend on the specific setting considered (road segment, time, individual driving behavior, etc.). However, overall, we obtained similar accuracy results as in the shown example, see Figure 4.4. Further, we observed that the most critical factor is the amount of data associated with a specific setting.

Moreover, we evaluated if the turning behavior at intersections changes with the time of the day (e.g., rush hour). For that reason, we construct one Markov chain that models the turning behavior of drivers during rush hour and one during the evening hours. We select both time frames (rush hour and evening) so that both Markov chains cover the same number of observations.

Further, to assess if the context-specific modeling boosts prediction accuracy, we assess if Markov chains of the same context have more similar turn probabilities than Markov chains of different contexts, considered as *Both*, cf. Figure 4.5. We measure the similarity via the average absolute difference of turn probabilities of the same Markov state. We constrain the comparison to inter-

sections with at least 50 observations for each context. The restriction results in 2 485 intersections, for which we compare the turn probabilities. Here, we observe that considering different contexts or time intervals can improve the accuracy of the turn probabilities.

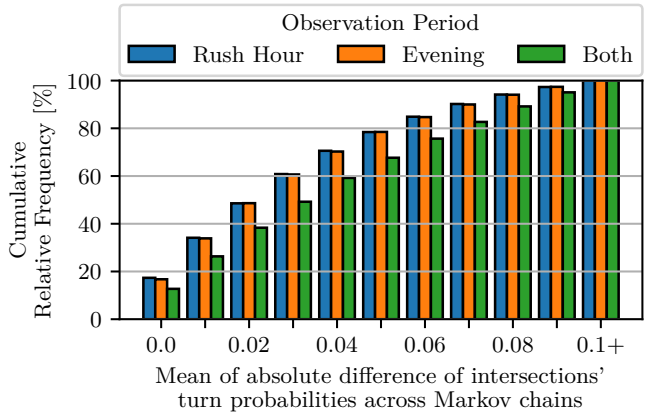

Fig. 4.5: Sensitivity to context: The histogram shows the cumulative distribution of the mean absolute differences of intersections' turn probabilities of different times of the day.

4.3.3 Runtime of the Prediction Algorithm

We conduct experiments to evaluate if the algorithms fulfill real-time runtime requirements. This is especially interesting in the context of dispatching. Dispatching algorithms have to decide in an adequate time, as with increasing passenger waiting time, the cancellation rate increases and the discrepancy between drivers' last recorded locations used for dispatching and their current location increases with execution time. We concentrate on the runtime analysis of the short-term route prediction because it has the highest runtime complexity of the three subcomponents. Therefore, the route prediction should be the most time-consuming subcomponent of the location prediction algorithm. During the experiment, we focus on drivers that leave their current road segment. Otherwise, the algorithm does not traverse the road network, and hence, only the estimation of a driver's current traversal speed and the retrieval of the current road segment's length contribute to the runtime.

We investigate the effect of an increasing prediction frame on the performance of the short-term route prediction algorithm. For this reason, we execute the algorithm with a prediction frame f of 5, 10, 30, and 60 seconds on the dataset. Figure 4.6 visualizes the results of the experiment measured on a consumer notebook. The results depict the median execution time to increase with an increasing prediction frame. This is due to the fact that with a larger prediction frame, the algorithm visits more road segments during traversal as the overall traversal time threshold, i.e., the traversal's early exit criterion, is larger.

Additionally, the number of road segments that must be analyzed for different intersections is responsible for the high variance of the results.

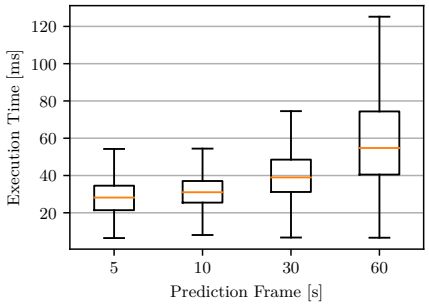

Fig. 4.6: Execution times for increasing prediction frames: The box-plot diagram shows the execution times of the short-term route prediction algorithm for different prediction frames. We run 10 000 predictions for each of the four folds.

The results of the experiment show that the execution times of the short-term route prediction algorithm fulfill real-time runtime performance requirements. Although with an increasing prediction frame, the algorithm's execution time increases as well, the algorithm's median execution time of 54 ms for a prediction frame of 60 s demonstrates that the algorithm also executes in real-time for larger prediction frames. Therefore, live dispatching algorithms can utilize the algorithm without experiencing losses in their runtime performance.

4.4 Risk-Averse Dispatch Strategies

Based on the probabilistic location predictions (cf. Section 4.1.2), we propose different risk-aware dispatch strategies. When choosing a driver, the goal is to not only minimize the (expected) arrival time but also account for the risk of critical delays. By $p_k^{(d)}$, we denote the probability that a driver d's current location is k (where the set of potential locations $K^{(d)}$ is derived as described in Section 4.2); by $t_k^{(d)}$ we denote the (estimated) time to reach the customer from location k. A driver d's random arrival time is denoted by $T^{(d)}$. Next, we define five different dispatch assignment strategies:

- **Best Expected Arrival Time:** Select the driver d with the smallest expected arrival time $E(T^{(d)}) = \sum_{k \in K^{(d)}} p_k^{(d)} \cdot t_k^{(d)}$, i.e.,

$$\min_d E(T^{(d)}).$$

  The approach optimizes the mean arrival time based on potential current locations instead of the outdated last known location as used in many status quo strategies, see Section 4.1.1. Based on the example in Figure 4.3, this strategy would avoid assigning the passenger request to the blue or green

driver as both have a high probability of a time-consuming detour resulting in a significantly longer mean estimated travel time.

- **Worst Case Optimization:** Select the driver d with the smallest worst case arrival time:
$$\min_d \left\{ \max_{k \in K^{(d)}} t_k^{(d)} \right\}.$$
The approach seeks to keep potential delays as small as possible. The approach allows to determine an upper bound for the arrival time.

- **Mean-Variance Optimization:** Select the driver d with the best balance of mean arrival time and associated variance penalized by a chosen factor α, e.g., $\alpha := 0.1$:
$$\min_d \left\{ E(T^{(d)}) + \alpha \cdot \sum\nolimits_{k \in K^{(d)}} p_k^{(d)} \cdot \left(t_k^{(d)} - E(T^{(d)}) \right)^2 \right\}.$$
The criterion combines the risk-neutral approach A with an incentive to choose drivers with low deviations of potential arrival times according to their current uncertain location, which in turn, yields more predictable arrival times.

- **Best Expected Utility:** Select the driver d with the best expected utility (i.e., smallest expected penalty) using a suitably chosen convex penalty function u, e.g., $u(x) := x^2$:
$$\min_d \left\{ \sum\nolimits_{k \in K^{(d)}} p_k^{(d)} \cdot u(t_k^{(d)}) \right\}.$$
The criterion looks for short arrival times while penalizing delays in a progressive way (as intended). Similar techniques are used in [140] for risk-sensitive path selections under uncertain travel costs.

- **Probability Constraints & Quantile-based Criteria:** Select the driver d with the best arrival time that can be realized with at least probability z (smallest z-quantile):
$$\min_d \left\{ q \,\middle|\, P\left(T^{(d)} \leq q \right) \geq z \right\}.$$
The criterion allows to formulate performance measures that are easy to interpret and to communicate, e.g., with $z = 90\%$ probability a driver will arrive within time q.

For all proposed approaches, the computation time is not an issue, as each objective can be easily evaluated numerically for a couple of potential drivers. The proposed approaches allow the optimization of different risk preferences, which may resemble a company's strategic goals. In this context, also further criteria can be used (e.g., conditional value at risk, etc.). Additionally, other factors as the customer-specific cancelation rate based on previous bookings and the current market situation, can be included in the decision process. Note, the key for all of those approaches is our probabilistic location prediction approach. Naturally, the suitability and effectiveness of the proposed risk-aware dispatch strategies have to be evaluated in practice.

4.5 Summary

We studied the limitations of established dispatching strategies and presented a novel approach to improve the dispatch processes of TNCs by using a location prediction algorithm. In particular, we identified inaccurate positional information as one aspect of drivers' late arrivals at pickup locations (cf. Section 4.1.1). These inaccuracies are produced by various circumstances (e.g., outdated data, noise, or technical limitations). Current state-of-the-art dispatch strategies have limitations, which are caused by using a driver's last observation. Using those systematically outdated locations can lead to wrong dispatch decisions, as drivers may have to take unexpected detours. Our approach seeks to attack the problem of outdated locations by predicting a probability distribution for a driver's actual location (cf. Section 4.1.2). Our algorithm uses patterns observed in past trajectory data to determine sets of potential locations and their corresponding probabilities (cf. Section 4.2). We demonstrate the applicability and accuracy of our prediction approach using numerical experiments based on a real-world dataset (cf. Section 4.3). The evaluation showed that the algorithm is quite accurate in predicting the possible locations of a driver. As the prediction of the correct road segment is quite hard, especially at the transitions between two road segments, the determination of the driving directions of a driver has high accuracy. In Section 4.1.2, we showed that accurate directions could significantly improve dispatching decisions.

Additionally, the accuracy of the turn probabilities and traversal speeds of individual road segments depends on the number of observations. In this context, we present fallback strategies (e.g., speed limits) for areas with only a few observations. The impact of these inaccuracies on dispatch decisions in less frequented areas is often negotiable as the communicated estimated arrival times are higher and there are wider variations between the drivers. Moreover, we verify that the algorithm's runtime is sufficiently fast to be applied in practice. In order to further improve traditional dispatch strategies, we propose different concepts to minimize the probability of large waiting times. This way, the risk of critical delays can be addressed, and thus, helps to minimize the customers' cancellation rates. Here, an overall evaluation of the impact on the cancellation rate was not possible due to the traffic-adjusted calculation of arrival times. Consequently, it was impossible to determine realistic estimated arrival times for different routes for past dispatch decisions. Our results reveal possibilities to improve the dispatch processes of TNCs as the risk of critical delays can be quantified and endogenized. Additionally, it enables these companies to use given resources more efficiently and increase customer satisfaction by communicating precise arrival time information.

5

Workload-Aware Joint Table Configuration Optimizations for Spatio-Temporal Data

Based on the performance requirements of modern spatio-temporal data mining applications, in-memory database systems are frequently used to store and process the data. To efficiently utilize the available DRAM capacities, modern database systems apply various optimizations (e.g., data compression or secondary indexes) to increase performance or lower the operation costs by reducing the memory footprint. However, the selection of cost and performance-balancing configurations is challenging due to the vast number of possible setups consisting of mutually dependent individual decisions. In this chapter, we introduce an approach to optimize the data management of columnar in-memory databases for spatio-temporal applications.

In Section 5.1, we use the research database *Hyrise* (cf. Section 2.3.2) to demonstrate the implications of different configuration decisions on the memory consumption and query performance of a system. We use the dataset of a transportation network company (TNC), introduced in Chapter 4, to visualize the impact and dependencies of various configuration decisions in a real-world example. In Section 5.2, we present the concept of leveraging fine-grained database optimizations to reflect spatio-temporal access patterns and describe the entire process to determine optimized table configurations for spatio-temporal workloads. In Section 5.3, we introduce our linear programming (LP) approach to determine application-specific fine-grained configuration decisions for a given memory budget. In Section 5.4, we extend our LP models to include data tiering decisions in the joint optimization process. To yield maintainable and robust configurations, we present two enhancements of the LP-based approach to incorporate reconfiguration costs and a worst-case optimization for potential workload scenarios.

Parts of the presented content, including the LP models for the joint compression, sorting, tiering, and indexing configuration optimization as well as the extensions to consider reconfiguration costs and robustness, have been published [282, 283].

5.1 Implications of Configuration Decisions on Query Performance and Memory Footprint

The selection of tuning configurations is a trade-off between performance (e.g., runtime) and costs (e.g., memory consumption). In Figure 5.1, we visualize the

implications of different tuning configurations measured in the research database *Hyrise*, which has comparable performance to other database systems [85]. To demonstrate the impact of different table configurations on the memory consumption and the runtime performance of database operations, we use the real-world dataset of a TNC, introduced in Section 4.3.1, as a running example. The used subset of the dataset consists of ten million observed locations of drivers for three consecutive days. We stored all attributes as integers. Based on the insertion order, a particular temporal ordering of the sample points exists, but we cannot guarantee that the timestamp column is sorted due to transmission problems. We compare the memory consumption and runtime performance of isolated executed table scan operations on a single column for the different chunk ordering options (incl. unsorted) and encoding configurations (incl. unencoded). The chunk ordering options include an unsorted option, which is determined by the insertion order of the table, and a sorting option for each column. The set of analyzed encodings consists of an unencoded option and the four compression approaches (i) LZ4 encoding, (ii) dictionary encoding, (iii) frame-of-reference encoding, and (iv) run-length encoding. Additionally, we include an index configuration, which is represented by a group key index [100] that can be applied to dictionary-encoded segments.

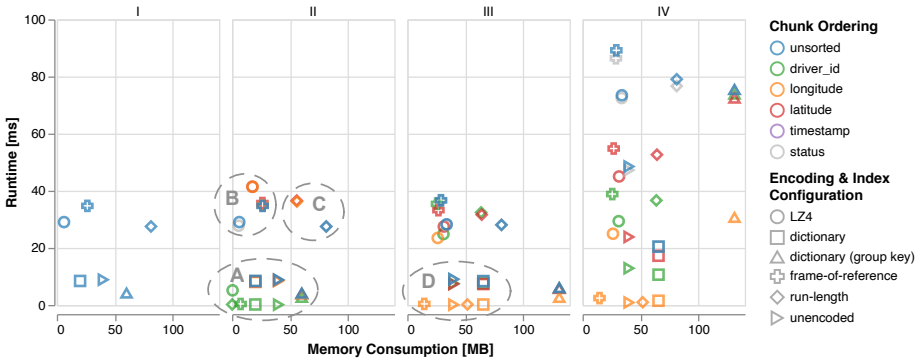

Fig. 5.1: Impact of different tuning decisions on memory consumption and performance for ten million observed locations partitioned into ten chunks and specific scan operations: a *LessThanEquals* scan (selectivity: 1%) on (I, II) the driver id column, (III) the longitude column, and (IV) a *Between* scan on the longitude column with a selectivity of 10%.

In Subfigure I, the runtimes of the encoding tuning options on an unsorted table for a *LessThanEquals* scan operation with a selectivity of 1% are visualized. Here, we can see that the different configuration options have a significant impact on performance as well as memory consumption. In this case, dictionary encoding is the compression approach with the lowest runtime (8.4 ms), which is over four times faster than the runtime of frame-of-reference encoding (37.7 ms). We can further reduce the runtime to 4.2 ms by using additional memory and applying a group key index structure [100].

In Subfigure II, we include sorting effects. The colors of the symbols indicate the sorting column of the table. If the scan operation is performed on a sorted

column, the runtime of the different configurations is significantly lower Ⓐ. For instance, the scan operation with applied run-length encoding executed on a sorted column takes only 0.17 ms compared to 27.5 ms on an unsorted column, which is over 160 times faster. Also, the memory consumption for this configuration (0.15 MB) is over 540 times lower than the value to store an unsorted column (81.25 MB). Moreover, in Ⓑ, we can observe that there is a correlation between the data of different columns and that the table's specific sorting decision can also impact the memory footprint and performance for various encodings (e.g., LZ4). LZ4 encoding stores data as a sequence of sequences and benefits from repeating sequences of literals [65]. Based on the resulting data characteristics, the ordering of the table by longitude or latitude leads to an increased memory consumption (280%) and runtime performance (140%) compared to the values for an unsorted table or a table sorted by timestamp. In contrast, the sorting decision has a significantly lower impact on the data footprint and runtime for frame-of-reference encoding. In Ⓒ, we can observe that ordering by one of the coordinate columns leads to a lower memory consumption for run-length encoding. Although the memory consumption is smaller, the runtime of the scan operation is about 30% higher for this tuning option compared to ordering by timestamp. In the benchmark, we partitioned the ten million observed locations into ten chunks with a chunk size of one million and applied the same configuration for each segment. Based on the specified queries' selectivities, some chunks can be pruned during query execution. As the data characteristics vary between chunks (e.g., during nighttime, we can observe that there is a significantly reduced number of active drivers and that the average trip length is increased), the impact of the sorting decision also varies between chunks. In this case, the sorting by timestamp leads to an increased memory consumption of some chunks that are not accessed by the query. Furthermore, other effects (e.g., fewer branch mispredictions and caching) based on the data characteristics impact the runtime performance of the tuning options.

In Subfigure III, we execute a scan operation with the same selectivity on the longitude column. Based on the changed data characteristics, the runtime performance and memory consumption of several tuning options changed Ⓓ. For instance, as the longitude column has a higher number of distinct values, the memory consumption to apply an index significantly increases. In Subfigure IV, we can observe that the given workload influences the effectiveness of the different tuning options. Based on the measurements, we can summarize that various aspects determine the efficacy of specific tuning options. Overall, Figure 5.1 motivates that the selection of suitable tuning options for a particular application context can be highly beneficial for memory consumption and performance. Further, the selection of performance and cost-balancing configurations consisting of mutually dependent tuning options is challenging due to the large number of potential setups. Thus, it is difficult for database administrators to estimate the impact of specific tuning decisions [58]. Especially for workloads with a mix of different types of queries, the impact of individual tuning decisions is hard to predict, and hence, the overall tuning is not easy to optimize.

5.2 Optimizing Table Configurations for Spatio-Temporal Workloads

As the workload and data characteristics particularly impact the effectiveness of various tuning options (cf. Section 5.1), we argue that the selection of application-specific tuning options can significantly improve the operating costs (e.g., memory consumption) and the performance of database systems. In Section 5.2.1, we explain our approach to optimize the data management of spatio-temporal data in general-purpose database systems by applying fine-grained database optimizations. In Section 5.2.2, we describe the process to determine optimized table configurations.

5.2.1 Leveraging Fine-Grained Database Optimizations to Reflect Spatio-Temporal Access Patterns in the Data Management Layer

Due to the high volumes of continuously accumulated trajectory data and cost-related limited main memory resources, database optimizations that leverage the characteristics of spatio-temporal applications are valuable. The used DRAM capacities are an important cost factor for in-memory databases [33, 180]. Concerning spatio-temporal data volumes, minimizing the data footprint can significantly reduce the system's operating costs. One aspect of spatio-temporal data management is that the specific data access patterns are often implemented in the application layer but are not reflected in the storage layer. The access frequency and access characteristics of spatio-temporal data points change over time. Complexity increases as several applications with different access patterns commonly work on the same spatio-temporal data. Based on the running example of a TNC, we observed applications with high selectivity queries on the most current data (e.g., passenger request dispatching). These applications partially ignore data after a specific timeframe, as the data do not reflect the current situation (e.g., traffic circumstances or order situation) anymore. Additionally, there are queries with a low selectivity on mostly older data for sophisticated analytical applications (e.g., demand prediction). Another aspect is that the spatio-temporal data characteristics can vary between different timeframes strongly (e.g., seasons or day and night) [390]. For instance, we can observe, for the TNC example, a reduced number of active drivers during nighttime. Also, different areas are more frequented based on the time. By considering these application-specific properties in the selection of tuning options, we can achieve the best performance based on the often cost-constrained hardware resources of spatio-temporal applications.

Modern database systems partition the data of a table by different criteria (e.g., time) to enable various optimizations (e.g., pruning). In *Hyrise*, we implicitly divide the data into horizontal partitions with a predefined maximum size (cf. Section 2.3.2). Similar concepts are applied by other databases [178, 248, 252]. This concept enables the application of different tuning options provided by data management systems for various partitions and segments of a table. Consequently, database administrators can apply fine-grained table configurations consisting of multiple tuning options (e.g., sorting, indexing, and compression configuration) to optimize the storage layer for the given data and workload characteristics. The implications of different configuration decisions

on the runtime performance are difficult to estimate for a database administrator [13, 38]. Consequently, various vendors apply simple threshold-based data-driven approaches. Based on a defined threshold (e.g., data volume or timeframe), data partitions are transferred to lower-cost storage mediums with higher latencies or more heavyweight compression techniques are applied. However, the selection of cost and performance-balancing configurations is challenging due to the vast number of possible setups consisting of mutually dependent individual decisions.

We introduce an approach that determines fine-grained table configurations optimizing multiple tuning dimensions as we leverage the characteristics of spatio-temporal applications. First, we exploit that the number of attributes in spatio-temporal applications is typically small, preventing the number of variables in our models from becoming unmanageable large. Second, the relatively small number of attributes limits the number of potential index candidates. Real-world datasets such as SAP's enterprise resource planning system include tables with hundreds of attributes. Faust et al. [99] showed that large indexes with up to 16 attributes are used in such systems. Based on the number of index candidates in such systems, it would be too complex to determine joint table configurations. Furthermore, the number of multi-column index candidates can be reduced in advance as in the spatio-temporal domain indexes that are potentially beneficial are typically known or can be restricted to a small set, i.e., the number of options. Third, we exploit that trajectory-based queries and spatio-temporal range scans dominate various spatio-temporal workloads [275]. Trajectory-based queries return all sample points of the trajectory or a sub-trajectory of a specific moving object. In contrast, a spatio-temporal range query determines all sample points of various moving objects in the defined temporal and spatial bounds (e.g., data visualization or acquiring data as input for data mining applications). For these query types, we can build relatively accurate cost models to estimate the operating costs for each segment. Additionally, to avoid random accesses, which are significantly slower than sequential accesses, *Hyrise* materializes the intermediate results into an uncompressed format once non-sequentially processing operators occur (e.g., join). Once an intermediate result is materialized, no encoded or tiered data is accessed any longer.

5.2.2 Process Overview

The optimization process consists of a controller, benchmark engine, and configuration optimizer. The controller operates the process and acts as an intermediary between the target database management system (DBMS) and the configuration optimizer. It collects the runtime and benchmark data from the target DBMS for the configuration optimizer, which uses the data to recommend a new table configuration based on the internal tuning models. Based on the selected tuning model and workload, the benchmark engine executes a set of benchmark queries on different table configurations.

The optimization of a table configuration can be triggered based on specific metrics (e.g., time intervals, performance constraints) or after the specified maximum size of a mutable chunk is reached. As shown in Figure 5.2, in the first step ①, the controller collects the workload data from the target DBMS. During runtime, modern database systems track various parameters to optimize the performance of a DBMS autonomously [169]. The SQL plan cache is used to extract

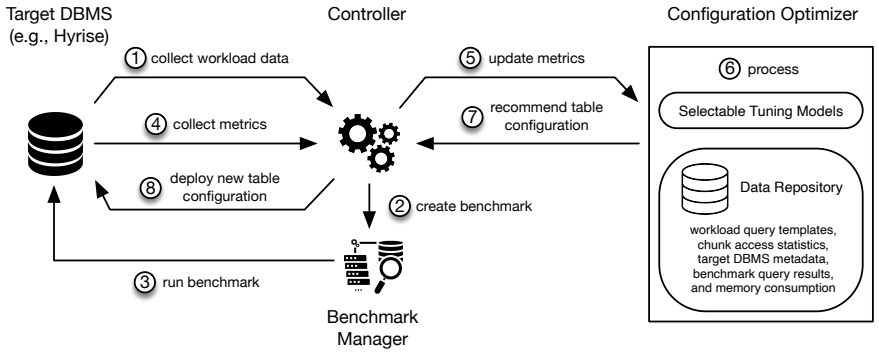

Fig. 5.2: Overview of the *optimization process* - The controller collects the workload information of the target DBMS. Based on the workload data, the benchmark manager executes a set of benchmark queries. The workload data and the benchmark metrics are transferred to the configuration optimizer, which processes the updated data and uses the selected algorithm to determine an optimized table configuration. Finally, the new table configuration is applied to the target DBMS.

the query templates of the workload. Each template describes a set of similar executed queries. Additionally, the segment access statistics captured by *Hyrise* are collected [84]. Moreover, we can use min/max statistics of each segment to determine the relevant chunks for a specific query template. These workload statistics are used in various physical database tuning tools [38, 169, 231]. Based on the query templates and the selected tuning model, the benchmark engine ② creates a set of benchmark queries. In step ③, the benchmark consisting of isolated executed single-column scan operations is conducted on different configurations to get information about the runtime performance and memory consumption of different encoding types for each column.

Due to the limited number of attributes in spatio-temporal data tables, the number of queries is manageable. Alternatively, estimated cost models [33, 219] or what-if analysis [8, 50] could be used to predict the runtime performance and memory consumption. By determining the input parameters for the LP models based on the stored trajectory data and queries, we can consider the application-specific characteristics in the optimization process. The controller ④ collects the benchmark results and ⑤ transfers the results as well as the workload data to the configuration optimizer. The configuration optimizer uses the data and the tuning models to ⑥ calculate an optimized configuration. Based on the recommended new table configuration ⑦, the controller ⑧ applies the configuration on the target DBMS. Here, each chunk's determined configuration can be applied asynchronously to reduce the overhead [86].

5.3 A Linear Programming Approach to Compute Joint Index, Sorting, and Compression Configurations

To efficiently utilize the available DRAM capacities, we introduce a joint LP-based approach to determine fine-grained configurations for specific spatio-temporal applications. As the different configuration decisions mutually influence each other, we seek to jointly optimize the compression, index, and ordering configuration to determine the best runtime performance for a given workload and memory budget. Note that each of those individual tuning problems is, in general, already challenging. We are still able to address a joint optimization of these dimensions as we exploit the specific characteristics of spatio-temporal data and applications, i.e., a limited number of columns and few query types. Further, we present different LP variants with differences regarding accuracy and numerical complexity, which allows for balancing multiple aspects (e.g., solving time) for the specific use case. For instance, to obtain a manageable problem complexity, we focus on single-attribute indexes and discuss the use of specific selected multi-attribute indexes based on domain knowledge.

Linear programming is an optimization technique for finding a solution for a given linear problem based on a mathematical model. In this context, a solution is defined by a set of variables, referred to as decision variables, whose values have to be decided in some optimal fashion [344]. The objective of linear programming is to minimize or maximize a linear function of these decision variables subject to linear constraints [70, 105]. The linear function is called the objective function and describes the optimization criteria (e.g., minimal runtime). The constraints are a set of equalities or inequalities that restrict the solution space for the value assignment of the decision variables. A solver determines the solution based on the mathematical model consisting of the objective function, a set of constraints, and a set of input parameters using standard algorithms (e.g., simplex or branch & bound, etc.).

In this section, we formalize the problem of database systems to determine memory-efficient table configurations for spatio-temporal data (Section 5.3.1). In Section 5.3.2, we introduce a general LP model that solves the specified problem. Additionally, we present heuristic solutions based on segment-specific costs (Section 5.3.3) using specialized LP models with (Section 5.3.4) and without (Section 5.3.5) sorting dependencies. In Section 5.3.6, we show how our models can be adapted to include database-specific restrictions.

5.3.1 Problem Definition

We consider a table with a set of attributes N and a set of chunks M (cf. Section 2.3.2). The problem is to find a valid table configuration for an available memory budget B and a given workload Q consisting of different query templates q, which occur with frequency f_q, such that the overall performance is maximized by minimizing the workload's total execution time. A valid table configuration consists of a (i) sorting, (ii) compression, and (iii) index configuration for all columns n within each *chunk* m. Consequently, for each segment (n, m) with $n \in N, m \in M$, we have to select a configuration from sets of available compression E, indexing I, and sorting O options. Note, these sets also include basic options, i.e., data can be unsorted, unencoded, or not indexed.

For the different sets, parameters, and variables of the LP approach, we provide a notation table (cf. Table 5.1).

Sets	N	set of attributes n
	M	set of chunks m
	E	set of encoding types e
	I	set of indexes i
	O	set of ordering options o
	Q	set of query templates q
	S	all scan operations s of a given query set Q
	K	set of configuration options k (CCD model)
Parameters	$n_{q,s}$	scan column of the scan operation q, s
	$\omega_{q,s}$	scan factor of the scan operation q, s
	f_q	scan frequency of query q
	$u_{q,s,e}$	successive scan penalty (for scan operation q, s given e)
	$a_{m,q,s}$	proportional size of the segment $(m, n_{q,s})$
	$v_{e,i}$	defines whether index i valid for encoding e
	$c_{m,n,e,o,i}$	scan costs for segment n_s of chunk m
	$c_{q,m,s,k}$	scan costs for chunk m (CCD model)
	$\phi_{m,n,e,o,i}$	memory consumption for segment (n, m)
	B	main memory budget
	$p_{q,s,e,o,i}$	execution time of a scan operation s
Variables	$x_{m,n,e,o,i}$	decision if a configuration is active (SSD, ISE)
	$y_{m,o}$	decision (a chunk m's sorting, SSD)
	$z_{m,n,e,i}$	decision (a segment n's configuration, SSD)
	$x_{m,k}$	decision (a chunk m's configuration, CCD)

Table 5.1: Notation table for the LP approach to determine table configuration optimizations.

As the required DRAM capacities of spatio-temporal data represent a significant cost factor of modern systems, the available memory resources have to be leveraged efficiently. The size of a segment (m, n) with configuration e, o, i (with $e \in E, o \in O, i \in I$) is described by the parameters $\phi_{m,n,e,o,i}$. Note, $\phi_{m,n,e,o,i}$ also includes the memory consumed by the index (if an index is applied on the segment). A valid table configuration must not exceed the given memory budget B. For a chunk $m \in M$ we consider potential joint configurations k from a given set of feasible options K. An option k characterizes *combinations* of configurations on a segment level, i.e., by $e_{m,n,k} \in E$, $o_{m,n,k} \in O$, and $i_{m,n,k} \in I$, we denote encoding, sort, and index decisions for column $n \in N$. Further, as we are focusing on spatio-temporal range queries and trajectory-based queries, each query template can be described as a composition of various *scan operations*, where the set S_q returns all scan operations s of a query template q, $q \in Q$. For scan operation s of template q the corresponding costs for chunk m and under a specific tuning configuration k are denoted by parameters $c_{q,m,s,k}$, $s \in S_q$, $q \in Q$, $m \in M$, $k \in K$.

5.3.2 General Model with Chunk-Based Configuration Dependencies

First, we consider a general model with chunk-based configuration dependencies (CCD), which represents a solution approach accounting for full cost dependencies within a chunk. In this model, the costs associated with a segment can depend on *all* specific configuration decisions of all *other* segments. This enables the model to particularly include multi-attribute indexes (e.g., k-d tree on latitude and longitude) or multi-attribute sorting options (e.g., space-filling curves) as long as (i) the number of considered configurations is tractable and (ii) the necessary data is at hand.

In the CCD model, we use the binary variables $x_{m,k}$, to express whether for a chunk $m \in M$ the joint configuration $k \in K$ is chosen. The objective of the CCD model is to minimize the cost (in this case, the runtime) for a given workload, cf. Q and f, over all x variables (denoted by $\vec{x}$)

$$min_{\vec{x}} \sum\nolimits_{m \in M, k \in K} x_{m,k} \cdot \sum\nolimits_{q \in Q, s \in S_q} f_q \cdot c_{q,m,s,k} \tag{5.1}$$

subject to the B budget constraint, which guarantees that the accumulated memory consumption of all segments (m, n) with their selected configurations e, o, i, b (cf. $\phi_{m,n,e,o,i}$) does not exceed the memory budget B, we use,

$$\sum_{m \in M, k \in K} x_{m,k} \cdot \sum_{n \in N} \phi_{m,n,e_{m,n,k},o_{m,n,k},i_{m,n,k}} \leq B. \tag{5.2}$$

To ensure a unique configuration option for each chunk m we use

$$\sum\nolimits_{k \in K} x_{m,k} = 1 \quad \forall m \in M. \tag{5.3}$$

The CCD model, (5.1) to (5.3), is linear and can be optimally solved using standard solvers. Naturally, the model's complexity and the required input is driven by the size of K, which can quickly become large when exhaustive combinations of tuning options are used. In this context, we recall that the options within K should be chosen by taking domain knowledge into account such that only *reasonable* configurations are considered. Moreover, we note that within these options for a certain chunk, we can exclude all options that are *dominated* by another option (with smaller required memory and better scan costs). This can reduce the number of options $|K|$ dramatically. The CCD model can be, e.g., used in specialized domain settings, where it is crucial to be able to account for complex tuning dependencies. In applications with less complex dependencies, basic models may be more suitable as they can be directly characterized by segment-based costs, which are discussed next.

5.3.3 Segment-Based Cost Estimation

Cost estimations for different configurations are a crucial aspect. To determine them on a segment-level, we consider the scan operations of a query and their execution order, which is defined by the query optimizer. Based on the *Hyrise* query optimizer implementation, the order of the scan operations is determined

by the operations' selectivity value, starting with the lowest selectivity value [35]. To consider that a scan operation s of a query template q (executed after a previous scan operation of the same query template) operates only on a subset of the data, we introduce a scan factor $\omega_{q,s}$. This factor $\omega_{q,s}$ is determined by the ordered sequence of (consecutively executed) scan operations of a query template. To determine $\omega_{q,s}$, we consider the selectivity factor of the j-th operation of a query template q denoted by $\tilde{\omega}_{q,j}$. By default, the selectivity factor of the *first* scan operation of a query template is defined as $\tilde{\omega}_{q,1} = 1$. Accounting for the combined selectivities of consecutive operations within a query template q for its scan operation s with operation order $J_{q,s} \in \{1, ..., |S_q|\}$ we obtain the scan factor, $s \in S_q$,

$$\omega_{q,s} = \prod_{j=1,...,J_{q,s}} \tilde{\omega}_{q,j}. \tag{5.4}$$

Besides the selectivity, each scan operation s of query template q, $s \in S_q$, $q \in Q$, has the following attributes: (i) the scanned column $n_{q,s}$, and (ii) the type of the scan operation (e.g., between scan, less than equal scan, equal scan). The costs of the scan operations on segment n of chunk m (aggregated over all scan operations that access n, $s \in S_q : n_{q,s} = n$ and weighted by f_q) are denoted by $c_{m,n,e,o,i}$ and determined by the segment's encoding $e \in E$ and index decision $i \in I$ as well as the ordering decision $o \in O := \{0\} \cup N$, where O includes all columns of the table plus the unsorted option ('0'). For $m \in M, n \in N, e \in E, o \in O, i \in I$, we define:

$$c_{m,n,e,o,i} := \sum_{\substack{q \in Q, s \in S_q: \\ n_{q,s} = n}} f_q \cdot p_{q,s,e,o,i} \cdot a_{m,q,s} \cdot \omega_{q,s} \cdot u_{q,s,e}. \tag{5.5}$$

The parameter $p_{q,s,e,o,i}$ defines the measured performance of the scan operation $s \in S_q$ of query $q \in Q$ executed as isolated scan operation on column $n_{q,s}$ stored in main memory if for the entire column encoding $e \in E$, index decision $i \in I$, and for all chunks the ordering decision $o \in O$ are applied. Further, in (5.5), we use the successive scan penalty $u_{q,s,e}$ as we observed that consecutive scans are slower than single scan operations, depending on the applied compression technique e. To reflect this observation and to adopt the measured isolated scan performance $p_{q,s,e,o,i}$ of the benchmark queries (cf. Section 5.2.2), we multiply $p_{q,s,e,o,i}$ of all consecutive scan operations with the fixed parameter $u_{q,s,e}$ for each value $e \in E$. This penalty value u is database-specific and can be measured with a simple set of benchmark queries.

Based on statistics and filters maintained by database systems, entire chunks can be pruned during query execution to increase the scan performance [86]. This is especially the case for temporal range queries, which only scan specific sections of the data. For that reason, we introduce the parameter $a_{m,q,s}$, cf. (5.5), which describes the proportional size of the segment $(m, n_{q,s})$ in relation to the amount of data scanned within a complete column scan on column $n_{q,s}$. As the costs for pruned chunks are neglectable, for not accessed chunks we let $a_{m,q,s} := 0$. For accessed chunks m, we define $a_{m,q,s}$ by their relative share of actually scanned chunks, i.e., by 1 divided by the number of not pruned chunks. This approximation is sufficient for our approach, although it has some inaccuracies if the values are unequally distributed over the different chunks.

5.3.4 Special Case: Segment-Based Model with Sorting Dependencies

The segment-based model with sorting dependencies (SMS) allows solving the configuration problem with costs determined per segment, cf. Section 5.3.3. It still allows to include intra-chunk dependencies between segments with regard to the chunk-based ordering decision. The corresponding segment's costs are computed based on (5.5) under consideration of the specified sorting column of the chunk. This enables the model to reflect the implications of different ordering configurations on memory usage and scan performance described in Section 5.1.

For the specialized SMS model, we use an adapted LP formulation, (5.1) to (5.3). The objective to minimize the costs is given by

$$min_{\vec{x},\vec{y},\vec{z}} \sum_{m\in M,n\in N,e\in E,o\in O,i\in I} x_{m,n,e,o,i} \cdot c_{m,n,e,o,i}, \tag{5.6}$$

where the binary variables $x_{m,n,e,o,i}$ describe whether a certain tuning configuration, cf. $e \in E, o \in O, i \in I$, for segment $n \in N$ of chunk $m \in M$ is used ('1') or not ('0'). Similar to (5.1), the overall cost is calculated as the sum of the costs c of all selected segment configurations, cf. (5.6).

To ensure valid table configurations, we define different sets of constraints. We distinguish between model-specific and database-specific constraints. The model-specific constraints define general requirements for the determined table configurations. Database-specific constraints to incorporate technical restrictions and limitations of different database systems are discussed in Section 5.3.6.

For the SMS model, we define three types of model-specific constraints. The first one is the memory budget rule, cp. (5.2)), that defines that the accumulated memory consumption of all segments (m, n) with their selected configurations e, o, i (cf. $\phi_{m,n,e,o,i}$) does not exceed the memory budget B,

$$\sum_{m\in M,n\in N,e\in E,o\in O,i\in I} x_{m,n,e,o,i} \cdot \phi_{m,n,e,o,i} \leq B. \tag{5.7}$$

Secondly, to guarantee that for each chunk m a unique ordering option is chosen, we use the binary variables $y_{m,o}$, which describe whether ordering o is used for chunk m, i.e.,

$$\sum\nolimits_{o\in O} y_{m,o} = 1 \quad \forall m \in M. \tag{5.8}$$

Thirdly, we use binary variables $z_{m,n,e,i}$ to ensure a unique index and encoding combination for chunk m's segment n,

$$\sum\nolimits_{e\in E,i\in I} z_{m,n,e,i} = 1 \quad \forall m \in M, n \in N. \tag{5.9}$$

The chunk variable y and segment variable z together specify the configuration $x_{m,n,e,o,i} = y_{m,o} \cdot z_{m,n,e,i}$. To express x linearly we use the following auxiliary coupling constraints for all $m \in M$, $n \in N$, $e \in E$, $o \in O$, $i \in I$,

$$x_{m,n,e,o,i} \geq y_{m,o} + z_{m,n,e,i} - 1 \tag{5.10}$$

$$x_{m,n,e,o,i} \leq y_{m,o} \tag{5.11}$$

$$x_{m,n,e,o,i} \leq z_{m,n,e,i} \tag{5.12}$$

5.3.5 Heuristic Solution: Independent Segment Effects

To heuristically solve the SMS model, we use a relaxation regarding the ordering dependencies of the cost effects between segments. In this simplified model with independent segment effects (ISE), we only account for whether a certain chunk's segment is sorted ('1') or not ('0'). Hence, instead of the full set of ordering options $O = \{0\} \cup N$ for each chunk, we use the *simplified* binary set $O := \{0, 1\}$ of available ordering options for each chunk's segment. For the unsorted option ('0'), the rows' order is set by the insert sequence and we use the costs $c_{m,s,e,0,i}$, cf. (5.5). If a segment (m, n) is sorted, we use $c_{m,s,e,n,i}$. With this formulation, we reduce the complexity by abstracting the sorting decision's intra-chunk effects. Thus, the model approximates the exact implications on the memory footprint and scan performance caused by sorting a chunk by column n (see Section 5.1).

Compared to the SMS model, the relaxed ISE model uses less variables and constraints. Specifically, we use a smaller family of binary decision variables $x_{m,n,e,o,i}$, where the ordering option only reflects the binary set $o \in \{0, 1\}$. The variables y and z, cf. (5.8)-(5.12), are not required. The objective of the ISE model is, cp. (5.6),

$$min_{\vec{x}} \sum_{m \in M, n \in N, e \in E, o \in \{0,1\}, i \in I} x_{m,n,e,o,i} \cdot c_{m,n,e,o \cdot n,i} \tag{5.13}$$

where we use $o \cdot n \in O$ to include the costs defined in (5.5) via $c_{m,s,e,o \cdot n,i}$. Similar to (5.7), we define the budget constraint,

$$\sum_{m \in M, n \in N, e \in E, o \in \{0,1\}, i \in I} x_{m,n,e,o,i} \cdot \phi_{m,n,e,o \cdot n,i} \ \leq B. \tag{5.14}$$

Note, the relaxed use of c and ϕ in (5.13) and (5.14) only approximates the exact values. Further, we directly use x to ensure that for each chunk m at most one column is sorted, cp. (5.8),

$$\sum\nolimits_{n \in N, e \in E, i \in I} x_{m,n,e,1,i} \leq 1 \quad \forall m \in M \tag{5.15}$$

and that for each segment, a unique configuration of compression e, sorting o, and indexing i, is chosen, i.e.,

$$\sum\nolimits_{e \in E, o \in \{0,1\}, i \in I,} x_{m,n,e,o,i} = 1 \quad \forall m \in M, n \in N. \tag{5.16}$$

5.3.6 Database-Specific Configuration Constraints

Additionally, we introduce database-specific constraints to the model-specific constraints, which enable the models to reflect certain properties of various database systems. The values for these constraints vary between databases and define combinations of incompatible indexing and encoding decisions. For *Hyrise*, secondary indexes require dictionary-encoded segments as they exploit the dictionary in order to improve space efficiency [100]. Consequently, indexes on all non-dictionary segments are forbidden. Regarding the ISE (cf. Section 5.3.5) and SMS (cf. Section 5.3.4) model, this is realized via the constraint

$$x_{m,n,e,s,i} \leq v_{e,i} \quad \forall m \in M, n \in N, e \in E, s \in S, i \in I, \tag{5.17}$$

where the binary parameters $v_{e,i}$, $e \in E$, $i \in I$, describe whether an index i is valid (=1) for a specific encoding e or not (=0). For the CCD model (cf. Section 5.3.2), the set of constraints (5.17) can be directly satisfied by considering only corresponding valid configuration options within the set K. Similar database-specific restrictions can be treated in the same manner. For instance, to ensure that specific index structures are only applied to specific data types (e.g., optimized spatial or spatio-temporal index structures).

5.4 Integrating Data Tiering Decisions into the Table Configuration Optimization Process

In-memory databases outperform traditional disk-based database systems by storing data on faster DRAM compared to solid-state drives (SSDs) or hard disk drives (HDDs). Based on the volumes of spatio-temporal data accumulated in different applications, the required DRAM capacities to store the data represent a significant cost factor or even exceed the available resources of modern systems. Furthermore, the scalability of single-node in-memory databases is also limited by the stagnation of DRAM capacities [218]. For this reason, modern database systems apply data tiering strategies that transfer less frequently used partitions of the data to slower and less expensive storage mediums [35, 348].

In this section, we enhance the problem definition to include data placement decisions and describe the necessary adoptions of the three LP models introduced in the previous sections to incorporate data tiering decisions.

5.4.1 Problem Definition

As described in Section 5.3.1, we consider a table with a set of attributes N and a set of chunks M. Instead of selecting an optimal table configuration for a given DRAM budget B, we determine a valid table configuration for a given set of available storage devices D with individual memory budgets. Consequently, for each segment (n, m) with $n \in N, m \in M$, we have to select a configuration from sets of available (i) compression E, (ii) indexing I, (iii) storage device D, and (iv) sorting O options (cf. Figure 2.5).

To improve the cost-efficiency of spatio-temporal data management systems, infrequently accessed data partitions are often transferred to slower and less expensive storage locations (cf. Figure 2.3). To reflect these properties in the model, we assume a given storage budget B_d for each storage medium $d \in D$. The memory consumption of a segment (m, n) with an applied tuning option e, o, i (with $e \in E, o \in O, i \in I$) is described by the parameters $\phi_{m,n,e,o,i}$ under the assumption that the used storage medium has no impact on the needed amount of bytes to store a segment. A valid table configuration must not exceed the given storage budgets B_d for all available storage devices $d \in D$. Additionally, we have to consider that the selected storage device d has a significant impact on the runtime performance of the given workload Q. Based on the notation table of the base models (cf. Table 5.1), we provide an adjusted version for the LP approach with integrated data tiering decisions (cf. Table 5.2).

Sets	N	set of attributes n
	M	set of chunks m
	E	set of encoding types e
	I	set of indexes i
	O	set of ordering options o
	D	set of storage devices d
	Q	set of queries q
	S_q	set of all scan operations s of a query q
	K	set of configuration options k (CCD model)
	W	set of potential workload scenarios w (robust model)
Parameters	$n_{q,s}$	scan column of the scan operation q, s
	$\omega_{q,s}$	scan factor of the scan operation q, s
	f_q	scan frequency of query q
	$u_{q,s,e}$	successive scan penalty (for scan q, s given e)
	$a_{m,q,s}$	proportional size of the segment $(m, n_{q,s})$
	$\tau_{e,i,d}$	penalty for storage medium d
	$v_{e,i}$	defines whether index i valid for encoding e
	$c_{m,n,e,o,i,d}$	scan costs for segment n of chunk m
	$\phi_{m,n,e,o,i}$	memory consumption for segment (n, m)
	B_d	memory budget for storage device d
	$p_{q,s,e,o,i}$	execution time of a scan operation q, s
	$\Delta_{m,n,e,o,i,d}(\bar{\eta}_{m,n})$	reconfiguration cost based on a given state $\bar{\eta}$
Variables	$x_{m,n,e,o,i,d}$	decision if a configuration is active (SSD, ISE)
	$y_{m,o}$	decision (a chunk m's sorting, SSD)
	$z_{m,n,e,i,d}$	decision (a segment n's configuration, SSD)
	$x_{m,k}$	decision (a chunk m's configuration, CCD)

Table 5.2: Adjusted notation table for the LP approach with integrated data tiering decisions and the enhancements of the segment-based models (cf. Section 5.5).

5.4.2 General Model with Chunk-Based Configuration Dependencies

To include data tiering decisions in the general model (CCD), we have to enhance the joint configuration k with $k \in K$. The set of possible options K includes, therefore, valid combinations of configurations on a segment level, i.e., by $e_{m,n,k} \in E$, $o_{m,n,k} \in O$, $i_{m,n,k} \in I$, and $d_{m,n,k} \in D$ we denote encoding, sort, index, and data placement decisions for column $n \in N$. In the memory constraint, we have to guarantee that the accumulated memory consumption of all segments (m, n) with their selected configurations e, o, i, b (cf. $\phi_{m,n,e,o,i}$) does not exceed a tier's budget B_d, i.e., $\forall d \in D$ we use (5.19). The adopted CCD model is described as follows:

$$min_{\vec{x}} \sum\nolimits_{m \in M, k \in K} x_{m,k} \cdot \sum\nolimits_{q \in Q, s \in S_q} f_q \cdot c_{q,m,s,k} \tag{5.18}$$

$$\sum_{m \in M, k \in K} x_{m,k} \cdot \sum_{n \in N : d_{m,n,k} = d} \phi_{m,n,e_{m,n,k},o_{m,n,k},i_{m,n,k}} \leq B_d. \tag{5.19}$$

$$\sum\nolimits_{k \in K} x_{m,k} = 1 \quad \forall m \in M. \tag{5.20}$$

5.4.3 Segment-Based Cost Estimation

To estimate the costs of a scan operation on a segment, we use the cost function (5.5) with the parameters described in Section 5.3.3. Moreover, we have to incorporate a factor that approximates the impact on the performance of a segment that is not stored in DRAM. The costs of the scan operations on segment n of chunk m (aggregated over all scan operations that access n, $s \in S_q : n_{q,s} = n$ and weighted by f_q) are denoted by $c_{m,n,e,o,i,d}$ and determined by the segment's encoding $e \in E$ and index decision $i \in I$ as well as the data placement decision $d \in D$ and ordering decision $o \in O := \{0\} \cup N$, where O includes all columns of the table plus the unsorted option ('0'). For $m \in M, n \in N, e \in E, o \in O, i \in I, d \in D$, we define:

$$c_{m,n,e,o,i,d} := \sum_{\substack{q \in Q, s \in S_q: \\ n_{q,s}=n}} f_q \cdot p_{q,s,e,o,i} \cdot a_{m,q,s} \cdot \omega_{q,s} \cdot u_{q,s,e} \cdot \tau_{e,i,d}. \tag{5.21}$$

Here, the parameter $p_{q,s,e,o,i}$ defines the measured performance of the scan operation $s \in S_q$ of query $q \in Q$ executed as an isolated scan operation on column $n_{q,s}$ stored in main memory d_0 if for the entire column encoding $e \in E$, index decision $i \in I$, and for all chunks, the ordering decision $o \in O$ are applied. Additionally, each storage medium has a penalty $\tau_{e,i,d}$, which reflects the difference between the measured access performance on DRAM and the access times on storage medium d, which can also depend on the index and encoding decision. Correspondingly, we multiply the estimated costs for an operation on a specified segment with the storage penalty $\tau_{e,i,d}$. The parameter $\tau_{e,i,d}$ is used to reduce the number of benchmark queries (see Section 5.2.2). In this context, an estimation of the storage penalty $\tau_{e,i,d}$ is suitable as the differences between storage mediums are generally higher (cf. Figure 2.3).

5.4.4 Segment-Based Model with Sorting Dependencies

Based on the adopted equation for the segment-based costs $c_{m,n,e,o,i,d}$, cf. (5.21), we have to adopt the binary decision variable $x_{m,n,e,o,i,d}$ and the objective of the model correspondingly. For the specialized SMS model with data tiering, we use an adapted LP formulation,

$$min_{\vec{x},\vec{y},\vec{z}} \sum_{m \in M, n \in N, e \in E, o \in O, i \in I, d \in D} x_{m,n,e,o,i,d} \cdot c_{m,n,e,o,i,d}, \tag{5.22}$$

where the binary variable $x_{m,n,e,o,i,d}$ describe whether a certain tuning configuration, cf. $e \in E, o \in O, i \in I, d \in D$, for segment $n \in N$ of chunk $m \in M$ is used ('1') or not ('0'). To ensure valid table configurations, we define different sets of constraints. The first one describes tiering-specific budget constraints, cp. (5.19), that defines that the accumulated memory consumption of all segments (m, n) with their selected configurations e, o, i on storage device d (cf. $\phi_{m,n,e,o,i}$) does not exceed a tier's budget B_d, i.e.,

$$\sum_{m \in M, n \in N, e \in E, o \in O, i \in I} x_{m,n,e,o,i,d} \cdot \phi_{m,n,e,o,i} \leq B_d \quad \forall d \in D. \tag{5.23}$$

Furthermore, to guarantee that for each chunk m a unique ordering option is chosen, we use binary variables $y_{m,o}$, which describe whether ordering o is used for chunk m, i.e.,

$$\sum\nolimits_{o \in O} y_{m,o} = 1 \quad \forall m \in M. \tag{5.24}$$

We have to modify the binary variable $z_{m,n,e,i,d}$ to ensure a unique index, encoding, and tiering combination for chunk m's segment n,

$$\sum\nolimits_{e \in E, i \in I, d \in D} z_{m,n,e,i,d} = 1 \quad \forall m \in M, n \in N. \tag{5.25}$$

By modifying the variable z, we enforce additionally that each segment is assigned to a storage device. The chunk variables y and segment variables z shall together specify the configuration $x_{m,n,e,o,i,d} = y_{m,o} \cdot z_{m,n,e,i,d}$. Similar to Section 5.3.4, we use the following auxiliary coupling constraints to express the variables x linearly for all $m \in M$, $n \in N$, $e \in E$, $o \in O$, $i \in I$, $d \in D$,

$$x_{m,n,e,o,i,d} \geq y_{m,o} + z_{m,n,e,i,d} - 1 \tag{5.26}$$

$$x_{m,n,e,o,i,d} \leq y_{m,o} \tag{5.27}$$

$$x_{m,n,e,o,i,d} \leq z_{m,n,e,i,d} \tag{5.28}$$

Various database systems support the tiering of entire database partitions. In contrast, the research database *Hyrise* also enables the tiering of single segments [84]. In case a chunk-tiering concept is applied, we have to ensure that all segments of a chunk are stored on the *same* storage device d, $d \in D$. For this purpose, we add the (optional) constraint:

$$\sum_{e \in E, i \in I, d \in D} d \cdot z_{m,1,e,i,d} = \sum_{e \in E, i \in I, d \in d} d \cdot z_{m,n,e,i,d} \quad \forall m \in M, n \in N. \tag{5.29}$$

5.4.5 Segment-Based Model with Independent Segment Effects

Similar to the SMS model, we have to enhance the ISE model to include data tiering decisions. Compared to the SMS model, the relaxed ISE model has fewer variables and constraints. Specifically, we use a smaller family of binary decision variables $x_{m,n,e,o,i,d}$. The variables y and z, are not required.

$$min_{\vec{x}} \sum_{m \in M, n \in N, e \in E, o \in \{0,1\}, i \in I, d \in D} x_{m,n,e,o,i,d} \cdot c_{m,n,e,o \cdot n,i,d} \tag{5.30}$$

For the budget constraints we define $\forall d \in D$,

$$\sum_{m \in M, n \in N, e \in E, o \in \{0,1\}, i \in I} x_{m,n,e,o,i,d} \cdot \phi_{m,n,e,o \cdot n,i} \ \leq B_d. \tag{5.31}$$

Moreover, we have to adapt the constraint (5.15) to ensure that also in the ISE model with tiering decisions, for each chunk m, at most one column is sorted,

$$\sum\nolimits_{n \in N, e \in E, i \in I, d \in D} x_{m,n,e,1,i,d} \leq 1 \quad \forall m \in M \tag{5.32}$$

and that for each segment, a unique configuration of compression e, sorting o, indexing i, and tiering d decision is chosen, i.e.,

$$\sum\nolimits_{e\in E,o\in\{0,1\},i\in I,d\in D} x_{m,n,e,o,i,d} = 1 \quad \forall m \in M, n \in N. \tag{5.33}$$

Furthermore, to obtain the same tiering in a chunk $\forall m \in M, n \in N$, we use the following (optional) constraint, cp. (5.29),

$$\sum_{e\in E,o\in\{0,1\},i\in I,d\in D} d \cdot x_{m,1,e,o,i,d} = \sum_{e\in E,o\in\{0,1\},i\in I,d\in D} d \cdot x_{m,n,e,o,i,d}. \tag{5.34}$$

5.4.6 Database-Specific Configuration Constraints

Correspondingly to the model-specific constraints of the SMS and ISE models, we have to adjust the database-specific constraint:

$$x_{m,n,e,s,i,d} \leq v_{e,i} \quad \forall m \in M, n \in N, e \in E, s \in S, i \in I, d \in d. \tag{5.35}$$

5.5 Enhancements of the Segment-Based Models

This section introduces two different enhancements, which are presented for the ISE model with data tiering described in Section 5.4. The first extension (Section 5.5.1) enables the internalization of reconfiguration costs required for a table configuration update (given an existing one). The second one addresses a robust configuration selection for multiple potential workload scenarios (Section 5.5.2), i.e., we look for allocations that are not optimized for one workload but "near-optimal" for various of them.

5.5.1 Minimal-Invasive State-Dependent Reconfiguration with Consideration of Modification Costs

Real-world workloads typically change over time. As a result, current data placements and configuration decisions might be outdated and have to be adapted to avoid performance deterioration. However, the reorganization of an applied configuration is costly and time-consuming [354]. The challenge is to identify 'minimally invasive' reallocations, which have a significant impact compared to their costs [169]. Therefore, we extend the ISE model to show exemplarily how to endogenize reconfiguration costs. We assume a current *configuration state*, e.g., characterized by parameters $\bar{x}_{m,n,e,o,i,d} \in \{0,1\}$, which characterize the applied configuration decisions.

In case a segment (m,n) is transferred from a current configuration $\bar{\eta}_{m,n} := (\bar{e}_{m,n}, \bar{o}_{m,n}, \bar{i}_{m,n}, \bar{d}_{m,n})$ to a new configuration (e,o,i,d), we generally assume given reconfiguration costs $\Delta_{m,n,e,o,i,d}(\bar{\eta}_{m,n})$. As the costs of different modifications vary (e.g., a change of the sorting configuration requires a reordering of all segments of the chunk) [354], the database administrator can define individual costs for each modification operation (e.g., changing the encoding of a segment). There are various metrics for determining the costs of individual modifications

(e.g., estimated time, size of reorganized data, or defined by the database administrator based on experience) [220, 295, 354]. To model reconfiguration costs, we replace the ISE objective as follows

$$min_{\vec{x}} \sum_{\substack{m\in M, n\in N, e\in E,\\ o\in\{0,1\}, i\in I, d\in D}} c_{m,n,e,o\cdot n,i,d} \cdot x_{m,n,e,o,i,d}$$
$$+\alpha \cdot \sum_{\substack{m\in M, n\in N, e\in E,\\ o\in\{0,1\}, i\in I, d\in D}} \Delta_{m,n,e,o\cdot n,i,d}(\bar{\eta}_{m,n}) \cdot x_{m,n,e,o,i,d}. \tag{5.36}$$

The additional cost term in (5.36) (governed by the penalty factor α) prevents configurations from being widely reorganized while the performance increase is only marginal. Note, while for α=0, we obtain the original performance-maximizing model without reconfiguration costs, for large α any costly configuration changes will be prevented. The α enables the DBA to balance the trade-off between potential performance gains and reconfiguration costs. In this context, the selection of the α depends on different aspects (e.g., the time interval between table optimizations and availability of resources). The other constraints of the basic ISE model remain unchanged (cf. (i) the budget constraint, (ii) at most one sorted column, (iii) unique configurations for each segment). Additional variables or constraints are not required. Hence, the ISE model with reconfiguration costs can still be solved via standard solvers.

5.5.2 Robust Configuration Selection for Different Potential Workload Scenarios

In general, spatio-temporal data characteristics and workloads are continuously influenced by the environment [390]. As future workloads are not entirely predictable, the performance can be negatively affected if the actual workload differs from the predicted one. This is a potential weakness of existing approaches that are only optimized for a specific workload. Hence, it is crucial to consider potential workload scenarios to obtain a robust performance. Potential future workload scenarios can be determined, e.g., based on previously observed (seasonal) workloads or forecasts (characterized by query frequencies).

Given a set of potential scenarios, data allocations and tuning configurations can be optimized to maximize expected performance (risk-neutral) or more robust (risk-averse) objectives (cf., e.g., worst case, expected utility, mean-variance criteria, etc.). In particular, such risk-aware objectives seek to avoid the risk of poor performance. To be able to deal with diverse scenarios, one is willing to sacrifice a certain share of the best possible expected performance.

We consider the set W of potential workload scenarios w, e.g., with probability P_w, $w \in W$, where $\sum_w P_w = 1$. We assume that a workload scenario w is characterized by a set of queries with given frequencies $f_q^{(w)}$ (within a certain period). Hence, the workloads costs defined in (5.5) generalize to multiple workloads $w \in W$ as, $m \in M, n \in N, e \in E, o \in O, i \in I, d \in D$,

$$c^{(w)}_{m,n,e,o,i,d} := \sum_{\substack{q\in Q, s\in S_q:\\ n_{q,s}=n}} f_q^{(w)} \cdot p_{q,s,e,o,i} \cdot a_{m,q,s} \cdot \omega_{q,s} \cdot u_{q,s,e} \cdot \tau_{e,i,d} \tag{5.37}$$

For instance, a worst-case optimization for the ISE model reads as follows. Using the non-negative real-valued variable Z for the worst-case performance costs over all scenarios $w \in W$, we use the objective

$$\min_{\vec{x},Z} \quad Z \tag{5.38}$$

subject to the constraints (5.31)-(5.33) and the new ones, $\forall w \in W$,

$$\sum_{m\in M,n\in N,e\in E,o\in\{0,1\},i\in I,d\in D} c^{(w)}_{m,n,e,o\cdot n,i,d} \cdot x_{m,n,e,o,i,d} \leq Z. \tag{5.39}$$

Note, the model (5.38)-(5.39) remains linear and is independent of the distribution P_w. As we only have one additional variable and $|W|$ new constraints, this extended/robust version of our ISE model has low overhead and the number of potential scenarios to be considered can be chosen such that the model remains tractable or the runtime does not exceed a targeted limit. Further, the proposed approach can also be used within the SMS or the CCD model.

5.6 Summary

In this chapter, we introduced our LP approach to determine optimized table configurations consisting of multiple tuning decisions for spatio-temporal data and workloads. To increase the performance or lower the operating costs by reducing the memory footprint, modern database systems provide various tuning options (e.g., data compression or index structures). However, the selection of cost and performance-balancing configurations is challenging due to the vast number of possible setups consisting of mutually dependent individual decisions. As the workload and data characteristics particularly impact the effectiveness of various tuning options (cf. Section 5.1), the impact of tuning decisions on the system's efficiency is challenging to estimate for DBAs. To improve the data management for spatio-temporal data, we proposed an approach that determines fine-grained table configurations to optimize different partitions of the data individually (cf. Section 5.2). We introduced different LP models to jointly optimize the compression, index, and ordering configuration to achieve the best runtime performance for a given workload and memory budget (cf. Section 5.3). The LP models consider cost dependencies at different levels of accuracy to address distinct requirements (e.g., high performance or low solver runtimes). The LP model with chunk-based configuration dependencies (CCD) represents a general solution to the problem. Additionally, we presented heuristic solutions based on segment-specific costs using specialized LP models with (SMS) and without (ISE) sorting dependencies. Based on the volumes of spatio-temporal data accumulated in different applications, the required DRAM capacities to store the data represent a significant cost factor or even exceed the available resources of modern systems. For this reason, we enhanced our LP models to include data placement decisions for multiple storage devices (cf. Section 5.4). Furthermore, we presented two extensions of the LP models to integrate reconfiguration costs and robustness (cf. Section 5.5).

6

Memory-Efficient Storing of Timestamps in Columnar In-Memory Databases

For columnar in-memory databases, the efficient storing of timestamps is challenging as numerous standard optimizations (e.g., compression approaches such as dictionary encoding) for columnar databases are developed for contradicting data characteristics (e.g., low number of distinct values). In the context of spatio-temporal data, the temporal component is represented in various applications by timestamps to reflect different and varying sample rates (cf. Section 2.2.4). Consequently, we have to store vast amounts of timestamps. In contrast to storing the spatial coordinates of moving objects in columnar databases [158, 244, 366], there is less focus on optimized storage concepts for the temporal component. However, it significantly impacts the memory footprint and performance.

By applying storage concepts designed for row-oriented databases, where the data transfer dominates the temporal information processing, we do not leverage the potential of the columnar data layout (e.g., an improved cache line utilization for filter operations). Consequently, optimized data layouts and compression techniques can improve the runtime performance and reduce the memory footprint [33, 36]. In this chapter, we describe the challenges of storing large amounts of timestamps in columnar databases (Section 6.1). We compare different data layouts to store timestamps and evaluate the data layouts' memory consumption and runtime performance in combination with various compression techniques (Section 6.2). Based on the advantages and disadvantages of different data layouts for specific requirements (e.g., memory limitations, performance constraints), we introduce two approaches to optimize the storage format for a given workload (Section 6.3). First, we introduce a heuristic approach for the workload-driven joint selection of a data layout and compression scheme. Second, we present a linear programming (LP) model to select an optimized compression scheme for attribute decomposition approaches that store a timestamp in multiple columns (Section 6.2.2). Parts of the presented content have been published [277].

6.1 Problem Definition

In various applications, in-memory databases are used to address the performance requirements of modern spatio-temporal data mining applications [79, 302, 390]. Especially for main-memory optimized databases that keep the most

data in relatively limited and expensive DRAM, the operating costs can be reduced by utilizing the available resources more efficiently [33]. In this context, the application-specific data characteristics (e.g., sample rate or accuracy of the temporal component) and workload properties significantly impact the memory consumption and performance of different storage formats. Besides the data layout, the applied compression scheme influences these factors and must be considered in the selection process. Consequently, we propose a joint optimization approach of these features to improve the storing and querying of temporal information for a specific application. By taking the application-specific access patterns into account, we are able to optimize the storage layout. One aspect is that the access frequency of trajectory data varies based on the up-to-dateness of the observation date. Furthermore, the access granularity of temporal queries is different for various partitions of the data. Another aspect is that in various spatio-temporal data mining applications, the queries predominantly request specific time ranges (e.g., all values of a day or an hour), which only address parts of the temporal information. By considering these access patterns, we can optimize the storage formats for timestamps and reduce the memory traffic by only processing the relevant parts of the timestamp.

There are different approaches to storing timestamps more efficiently for spatio-temporal and time-series data. A method is to determine the temporal information based on the position of an observed location in the sequence of chronologically ordered locations (cf. Section 2.2.4). By applying this approach, we can significantly reduce memory consumption. However, this approach does not apply to applications where the locations are tracked without a fixed sample rate or the transmission of each observed location can not be guaranteed. Other approaches use specific data layouts to represent the temporal component and aggregate values for fixed periods to reduce the data footprint [352, 353]. A drawback of such approaches is that they increase the uncertainty, and the adaption to varying sample rates is rather complex.

Additionally, various approaches optimize the compression scheme [1, 33, 178] for a database table. These approaches focus on optimizing compression scheme based on data and workload characteristics but do not consider that different data layouts can be applied to store specific types of data. Based on the applied data layout, columns' data characteristics can significantly change and create further optimization potentials.

6.2 Data Layouts for Timestamps in Columnar Databases

This section describes and evaluates different data layouts for timestamps in columnar in-memory databases. In various spatio-temporal applications, we observed that the provided data types of columnar databases are not used. Although these data types offer many features to query and process temporal information, the developers used customized data layouts based on standard data types due to performance or memory constraints. As displayed in Figure 6.1, we analyze standard data layouts such as string format and Unix timestamps. Additionally, we evaluate different attribute decomposition approaches that store the date and time components separately or use multiple columns concerning their applicability in columnar databases.

6.2.1 Standard Data Layouts for Timestamps

The first data layout based on standard data types that we observed in spatio-temporal data mining applications is the string format. It stores a timestamp in a single column of the data type string. Based on the ISO 8601 guidelines, the different time units (e.g., year, month, or day) are stored in descending order and divided by specific delimiters. The format applied by most applications is *YYYY-MM-DD HH:MM:SS*. Due to the specified delimiter symbols, it is possible to query parts of the timestamp (e.g., all observed locations in a specific hour over multiple days) via SQL LIKE statements. Based on the defined order of the time units, we can preserve the chronological order of the timestamps. SQL statements that require such an order (e.g., BETWEEN) can be realized via string comparisons. A disadvantage of this approach is that a relatively high amount of memory is necessary to store a single timestamp. Consequently, we have high memory traffic to process the data even if only parts of the timestamp are queried.

String (string)

TIMESTAMP
2018-11-01 00:00:11
2018-11-01 00:00:26
2018-11-01 00:01:42
2018-11-01 17:00:16
2018-11-03 06:14:40

Unix Timestamp (integer)

TIMESTAMP
1541030411
1541030426
1541030502
1541091616
1541225680

Date/Time (string)

DATE	TIME
2018-11-01	00:00:11
2018-11-01	00:00:26
2018-11-01	00:01:42
2018-11-01	17:00:16
2018-11-03	06:14:40

Multiple Columns (integer)

YEAR	MONTH	DAY	HOUR	MINUTE	SECOND
2018	11	1	0	0	11
2018	11	1	0	0	26
2018	11	1	0	1	42
2018	11	1	17	0	16
2018	11	3	6	14	40

Fig. 6.1: Visualization of different data layouts to store timestamps in relational database systems.

Another approach is the usage of Unix timestamps, which are widely used by operating systems and file formats. Each timestamp is stored as an integer value in a single column. The integer value represents the number of seconds that have elapsed since the Unix epoch, which is defined as the 00:00:00 UTC on the first of January 1970. Compared to the string format, we need less memory to store a single timestamp. Additionally, modern CPUs are optimized to process integers values efficiently [363]. This approach is well-suited for queries requesting a continuous data period. Due to leap seconds and years, calculating specific recurring periods in a more extensive timeframe is not trivial. For that reason, several database systems convert the Unix timestamps into the string format to process these types of queries (e.g., all observed locations in an hour on various days). This entire process is time-consuming, especially for large spatio-temporal data volumes.

Several standard compression techniques for columnar databases benefit from a relatively low number of distinct values (e.g., dictionary encoding) or a high number of equal consecutive values (e.g., run-length encoding) [263]. In general, both presented data layouts produce a high number of distinct values. To address this issue, it can be beneficial to split the timestamp into a date and time part for several applications. By storing the data and time components in separate columns, we can apply at least on the date column more efficient

compression approaches based on resulting data characteristics (e.g., reduced the number of distinct values). Especially for fine-grained tracking applications, which store the position of a moving object several times per minute, we have large sequences of equal values in the date column. Additionally, we can reduce the memory traffic for queries that only address the date or time unit. Correspondingly, a drawback is that we have to perform two scan operations for queries that access both components.

6.2.2 An Attribute Decomposition Approach to Store Timestamps

Based on the concept of the separation of date and time, we propose an attribute decomposition approach to store timestamps. As displayed in Figure 6.1, we store each time unit in a single integer column in this data layout. Additionally, further distributions are possible based on the workload and data characteristics of the specific application scenario. In this context, data access metrics (e.g., histograms of simultaneous queried values) or rule-based pattern analysis (cf. Section 3.2.5) can be applied to identify a decomposing strategy. Based on the tracking technology's accuracy and sample rate, we can reduce the number of columns to store the temporal component. In contrast to row-oriented databases, we can add further columns without high reconfiguration costs in columnar databases if the sample rate changes or higher accuracy is required [262]. Besides, columns containing only a single default value (e.g., the column to store seconds if the sample rate is on a minute base) and columns with a low number of distinct values can be efficiently compressed in column stores.

Furthermore, there are specific access patterns that query fixed standard time ranges (e.g., all observed locations of a day, month, or quarter) in various spatio-temporal applications. The multiple columns data layout is beneficial for such queries, as we can reduce the memory traffic by leveraging cache lines more effectively and processing only parts of the temporal information. Additionally, the multi columns data layout can improve the data characteristics for different standard compression approaches of columnar databases (see Figure 6.1). For instance, all columns have a limited and low number of distinct values, which is beneficial for dictionary encoding, as we have stable dictionaries (except the year column) and a significantly reduced amount of bits to store the dictionary position of a value. Depending on the sample rate, we have sequences of equal values (e.g., month or year column), which can be efficiently compressed by applying run-length encoding. In contrast, a disadvantage of the approach is the selection of specific timestamps or time ranges, as multiple columns have to be scanned. To mitigate these performance drawbacks, multi-column index structures or query optimizations that consider column dependencies can be used [101, 168].

6.2.3 Impact of Different Compression Techniques on the Memory Consumption and Query Performance

Modern main-memory-optimized databases apply a variety of compression techniques [33]. In this section, we analyze the impact on the memory consumption and runtime performance of the presented data layouts (cf. Section 6.2) in combination with different compression approaches. To evaluate the data layouts, we

use a real-world dataset of a transportation company introduced in Section 4.3.1 and the in-memory research database *Hyrise* (cf. Section 2.3.2), which is optimized for columnar data layouts [86]. All measurements have been executed server equipped with Intel Xeon E7-4880v2 CPUs (2.50GHz, 30 logical cores). The dataset includes the timestamps of ten million dispatch process-related observed drivers' locations for three consecutive days of a transportation network company (cf. Section 4.3.1). A certain temporal ordering of the sample points exists based on the insertion order, but we cannot guarantee that the timestamp column is sorted due to transmission problems and delayed transmissions.

Memory Consumption

To analyze the impact on the data footprint of the data layouts for different compression techniques, we applied the three compression techniques (i) dictionary encoding, (ii) LZ4 encoding, and (iii) run-length encoding on the ten million timestamps of the dataset. As displayed in Figure 6.2, the applied compression approach has a significant impact on the memory size that is necessary to store the timestamps. For instance, the Unix timestamp column consumes with applied LZ4 compression 1.3 MB, which is only 3.2 percent of the main memory to store the Unix timestamp column with applied dictionary encoding. Additionally, we observe that data layouts storing the data as integer values have a better compression rate for LZ4 and run-length encoding. Although the timestamps are decomposed and stored in six different columns, the multiple columns approach consumes comparable memory amounts. For run-length (1.7 MB) and LZ4 (2.4 MB) encoding, the multiple columns approach requires significantly less memory to store the data compared to the string format. In addition, the measurements show that different compression techniques are better suited for specific data layouts. For example, the separated date and time approach consumes around two-thirds of the string approach for applied run-length encoding. In contrast, for LZ4 encoding, the string data layout needs 37.5 MB, which is significantly less memory than the separated date and time approach (66.9 MB). A reason for this is that the date column contains long sequences of equal values through the separation, which can be efficiently compressed with run-length encoding.

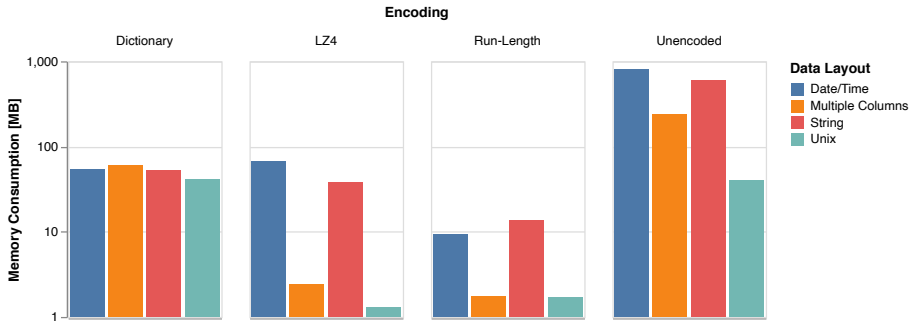

Fig. 6.2: Comparison of the memory consumption of the different data layouts in combination with four different compression approaches.

In Figure 6.2, a notable observation is that the multiple columns approach has the highest memory consumption of the four data layouts for dictionary encoding, even though the data characteristics should be beneficial for this compression technique. The reason for this is that the used columnar database does not support a bit-packing mechanism [363]. Consequently, one byte is used to store the dictionary position for each value instead of the minimal number of bits that are necessary to specify the dictionary entry. By applying such bit-packing mechanisms, we can reduce the data footprint significantly. Figure 6.3 shows the memory consumption for the six columns of the multiple columns data layout with and without bit-packing. Based on the minimal number of bits to encode the corresponding columns dictionaries, we determined the bit-packed memory consumption.

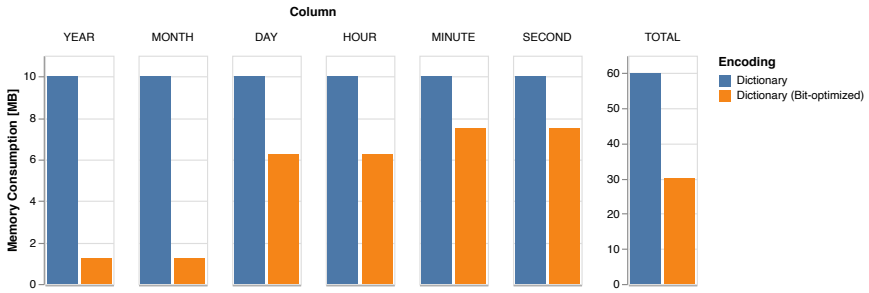

Fig. 6.3: Comparison of the attribute decomposition approach's memory consumption for the different columns with and without applied bit-packing mechanism.

The bit-optimized storing of the values is particularly efficient for our dataset's YEAR and MONTH columns, as both columns only contain one distinct value. Consequently, we can reduce the memory footprint of these columns to about 1.25 MB. By applying a bit-packing mechanism, the overall memory consumption of the multiple columns data layout could be reduced by nearly 50 percent to about 30.5 MB. With a memory consumption of about 30.5 MB, the multiple columns data layout with applied bit-packing has the lowest data footprint of the analyzed data layouts for dictionary encoding. By using such an optimization, the space efficiency of the storage layout increases significantly [90]. Furthermore, we can observe that the specific implementation of the approaches is relevant for selecting an efficient storage layout.

Query Performance

Data compression is usually a trade-off between the size of the data structures and its access performance [235, 276]. To analyze the data layouts' runtime performance for different compression approaches, we defined a set of five benchmark queries. The benchmark contains different commonly used query types, including single-value access and range scans with varying selectivity values and query ranges to consider a broad spectrum of access patterns. The first query (q_0) is an equal scan that selects a specific timestamp. The queries (q_1, q_2) represent standard temporal range queries between two timestamps. In contrast to

q_1 that selects all data points between the 1st of November 2018, 21:01:43, and the 3rd of November 2018, 03:15:06 (including five million entries), q_2 queries all observed locations in a relatively small timeframe of 180 seconds. The queries q_3 and q_4 access only parts of the timestamps. Query q_3 returns all data entries of a specific day, and q_4 returns for all three days all data entries in a specific hour (between 6 am and 7 am).

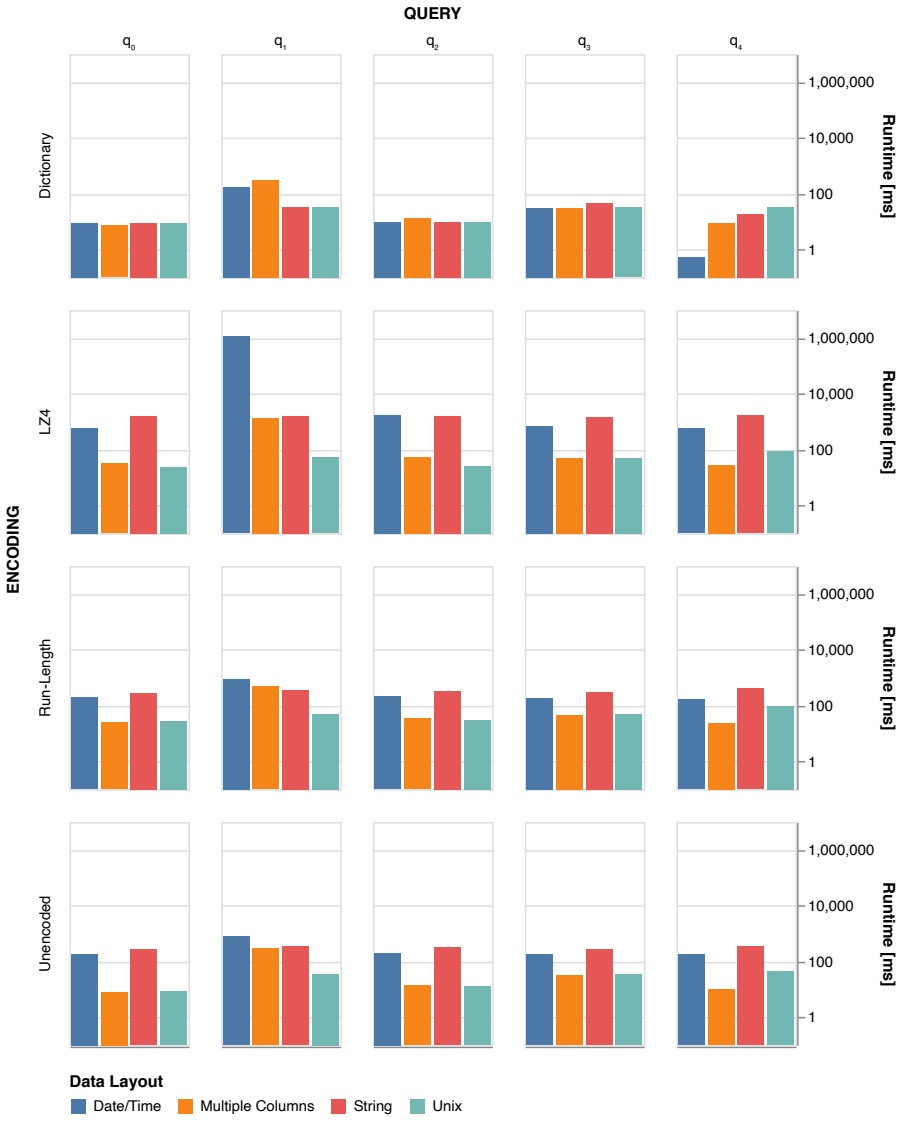

Fig. 6.4: Comparison of the runtime performance of the different data layouts in combination with different compression techniques.

As displayed in Figure 6.4, depending on the applied compression technique, there are significant differences in the query runtime. Overall, dictionary encod-

ing has the lowest query runtimes for our set of benchmark queries. Also, the string and separated date and time approach have a comparable performance for dictionary encoding compared to the multiple columns and Unix timestamp approaches. For query q_4, the separated data and time approach with dictionary encoding has the lowest overall runtime. The Unix timestamp and multiple columns approach generally have a better query runtime for all other compression techniques. Additionally, we can observe that the separated date and time approach has a better query runtime than the string data layout except for query q_2. Moreover, the multiple columns approach has a comparable or even better performance than the Unix timestamp except for query q_2. The performance drawbacks of the data layouts that use more than one column for query q_2 are caused by the fact that multiple scan operations (depending on the previous scan operations) have to be performed. In contrast, these approaches are beneficial for queries that consider only parts of the timestamps (e.g., q_4) as the data traffic can be reduced significantly. Finally, we can observe that the selection of an efficient data layout and compression technique strongly depends on the given workload characteristics and system constraints (e.g., available memory).

6.3 Workload-Aware Optimizations to Store Timestamps in Columnar In-Memory Databases

As described in the previous section, the different configurations to store timestamps have significant differences in memory consumption and runtime performance for various workload characteristics. The selection of cost and performance-balancing configurations is challenging due to the number of possible configurations and the fact that usually spatio-temporal applications with different query characteristics operate on the same database table. To optimize the storage layout of timestamps, we present two different workload-driven approaches. In Section 6.3.1, we describe a heuristic approach for the workload-aware selection of a timestamp storage configuration consisting of a data layout and compression technique. In Section 6.3.2, we introduce an LP model to select an optimized compression scheme for the multiple columns data layout.

6.3.1 Workload-Driven Combined Data Layout and Compression Scheme Optimization

Based on the application-specific constraints, we can apply different heuristic approaches to select a configuration. As the selection decisions of a data layout and compression technique are mutually dependent (cf. Section 6.2.3), we propose a joint optimization approach. By considering the workload and data characteristics, we can adapt the storage format to application-specific constraints. For each data layout $d \in D$, where D describes the set of available data layouts, and compression technique $e \in E$, where E defines the set of relevant data encodings, we specify $\phi_{d,e}$ as the memory consumption to store the timestamps of the given dataset with the data layout d and the encoding e. Besides various compression approaches, the set E can also contain different implementations (e.g., bit compression for dictionary encoding) as well as configurations (e.g., different LZ4 block sizes) of the same compression technique. Table 6.1 presents a notation table for the optimization approach.

Sets	D	set of data layouts d
	E	set of encodings e
	Q	set of queries q
	M	set of partitions m
Parameters	$\phi_{d,e}$	memory consumption for a data layout d and encoding e
	$p_{d,e,q}$	execution time of a query q
	f_q	frequency of query q
	$a_{m,q}$	proportional costs of partition m for query q
	α	factor for balancing memory consumption and runtime
Benefits	r_d	benefit of data layout d
	$r_{d,e}$	benefit of data layout d and encoding e
	$r_{d,e,m}$	benefit of data layout d and encoding e for partition m

Table 6.1: Notation table for the workload-driven combined data layout and compression scheme optimization approach.

For applications with strict memory requirements, we can select the configuration with minimal memory consumption, $min_{d,e}\{\phi_{d,e}\}$ for $d \in D, e \in E$. Equally, we can select the configuration based on the maximum performance for a given workload without any data footprint considerations. We specify a workload Q as a set of queries $q \in Q$. The performance of a given workload Q, we define as the sum of the costs of each query $q \in Q$, which are calculated based on the costs of the query execution $p_{d,e,q}$ multiplied with the frequency of the query f_q,

$$\sum\nolimits_{q \in Q} p_{d,e,q} \cdot f_q. \tag{6.1}$$

To determine the costs $p_{d,e,q}$ for each query, we can measure the runtime performance or use cost models. We calculate the benefit $r_{d,e}$ based on a weighted ratio between memory consumption and runtime performance to optimize the timestamps' storage configuration. A similar approach is used by Valentin et al. [342] in the context of the index selection problem. For each combination of a data layout $d \in D$ and encoding $e \in E$, we define the benefit r as ($\alpha \geq 0$):

$$r_{d,e} = 1/\Big(\phi_{d,e} \cdot (\sum\nolimits_{q \in Q} p_{d,e,q} \cdot f_q)^{\alpha}\Big). \tag{6.2}$$

The α value defines the proportional balancing of memory consumption and runtime performance for the optimization objective. This factor enables the database administrator (DBA) to adapt to different application requirements. The applied data layout and compression approach should be transparent for the application. Based on the Equation (6.2), we select the data layout d and encoding e with the maximum benefit $max_{d,e}\{r_{d,e}\}$ for $d \in D, e \in E$. The database should optimize the used configuration internally and provide a unified interface (e.g., the timestamp data type).

Various modern database systems divide a table into partitions based on partition criteria (e.g., time) to benefit from pruning during query execution, more efficient workload distribution, and simplified data tiering [86, 178, 248, 252]. An optimized adaptation of the storage layout to spatio-temporal access patterns can be achieved by applying optimized configurations individually for

each partition [282]. In this context, we have to calculate the benefit for each partition $r_{d,e,m}$ for $d \in D$, $e \in E$, and $m \in M$, where M describes a set of partitions. Consequently, we have to adapt the equation:

$$r_{d,e,m} = 1/\Big(\phi_{d,e,m} \cdot (\sum\nolimits_{q \in Q} p_{d,e,q} \cdot f_q \cdot a_{m,q})^{\alpha}\Big). \tag{6.3}$$

In (6.3), we determine the memory consumption $\phi_{d,e,m}$ separately for each partition. Additionally, we have to introduce a parameter $a_{m,q}$, which determines the proportional share of the partition m of the runtime performance $p_{d,e,q}$ for a given query $q \in Q$. Similar to the approach without data partitions, we select for each partition $m \in M$ the configuration with the highest benefit $r_{d,e,m}$. This approach requires that the database system is capable of applying different data layouts for various partitions of the data and adjusting the query processing correspondingly. If the data layout should be consistent for all partitions, we can determine the aggregated benefit of all partitions for each data layout $d \in D$ by

$$r_d = \sum\nolimits_{m \in M} \max_e \{r_{d,e,m}\}. \tag{6.4}$$

Based on the data layout with the highest overall benefit $max_d\{r(d)\}$, we can determine the encoding scheme for each partition m correspondingly.

6.3.2 Optimized Compression Scheme Selection for Multiple Column Data Layouts

By applying an attribute decomposition approach (cf. Section 6.2.2), we have further optimization potential to improve the runtime performance or reduce the memory footprint. As the different temporal components of timestamps are stored in separate columns, we can select an optimized encoding approach for each column. Figure 6.5 shows that the columns in the multiple columns data layout have different data characteristics, which are beneficial for specific compression techniques. For instance, in our dataset of a transportation network company, the YEAR column has only one distinct value. Consequently, this column is well-suited for run-length encoding or could also be stored as a single default value. In contrast, the SECOND column has significantly shorter sequences of equal values, which leads to a lower compression rate for run-length encoding. We propose an LP approach to determine an optimized compression scheme for given workloads and memory budgets to address this issue. Furthermore, by introducing memory budgets, we enable DBAs to specify the amount of memory used to store the timestamp and optimize the compression scheme correspondingly.

To select for each column an encoding to achieve the lowest overall runtime for a given workload Q, we have to determine the proportional costs of each database operation (e.g., scan operation) for each column. We define the function $A(q)$, which decomposes a query q with $q \in Q$ and determines all single operations for the individual columns of the data layout. For instance, $A(q)$ divides a temporal range scan q into multiple scan operations based on the temporal units queried by q. As spatio-temporal workloads are often dominated by range queries and trajectory-based queries [275], we are focusing on scan operations in this context. The approach can be extended for further spatio-temporal

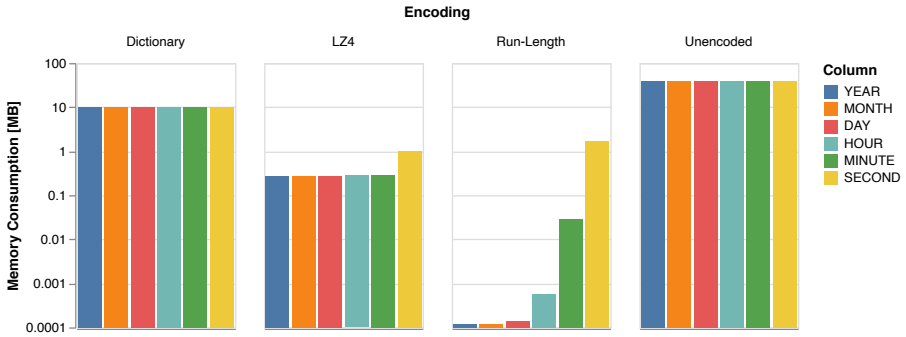

Fig. 6.5: Comparison of the memory consumption of the different columns in the multiple columns data layout for dictionary, LZ4, and run-length encoding. Especially for run-length encoding, the varying data characteristics of the columns have a significant impact on the data footprint.

operations. Correspondingly, S describes the set of all operations of a given workload:

$$S = \bigcup_{q \in Q} A(q). \tag{6.5}$$

The costs of a scan operation s, $s \in S$, on a column is denoted as $c_{s,e}$ and determined by the column's encoding $e \in E$. For $s \in S, e \in E$, we define:

$$c_{s,e} := p_{s,e} \cdot \omega_s \cdot f_s \cdot u_{s,e}. \tag{6.6}$$

The parameter $p_{s,e}$ defines the measured scan performance of an isolated executed scan operation on the column n_s with applied encoding e. Further, the parameter ω_s denotes the accumulated selectivity of the previous operations of the same query and f_s the frequency of the scan operation. We use the successive scan penalty $u_{s,e}$ as we observed that consecutive scans are slower than single scan operations, depending on the applied compression technique e. The order of the scan operations is determined by the selectivity of the different scan operations based on $A(q)$ of a query q. The objective of the model is to minimize the costs (in this case, the runtime) for a given set of scan operations S,

$$\min_{e} \sum\nolimits_{s \in S, e \in E} x_{n_s,e} \cdot c_{s,e} \tag{6.7}$$

where the binary variables $x_{n_s,e}$ describe whether a certain encoding e is applied ('1') or not ('0') on column n_s. Here, $n_s \in N$ is the column in the set of all columns N that is scanned by the given scan operation s. For the model, we define two constraints. The first constraint guarantees that the accumulated memory consumption of all columns $n \in N$ with their selected encoding does not exceed the given memory budget B, i.e.,

$$\sum\nolimits_{n \in N, e \in E} x_{n,e} \cdot b_{n,e} \leq B. \tag{6.8}$$

To guarantee that for each column n exact one compression approach $e \in E$ is selected, we specify the second constraint:

$$\sum\nolimits_{e \in E} x_{n,e} = 1 \quad \forall n \in N. \tag{6.9}$$

To solve the optimization problem, a standard solver with support for mixed-integer programming (e.g., Gurobi [121]) is used.

6.4 Summary

In this chapter, we presented different approaches to store timestamps in columnar in-memory databases more efficiently. The efficient storing of timestamps is challenging as numerous standard optimizations (e.g., compression approaches such as dictionary encoding) for columnar databases are developed for contradicting data characteristics (e.g., low number of distinct values). However, for various applications (e.g., spatio-temporal applications), the access performance and memory consumption of temporal information are crucial aspects. By comparing different combinations of data layouts and compression techniques, we observed significant differences in memory requirements and runtime for different access patterns of the applied configurations (cf. Section 6.2). Based on the advantages and disadvantages of different configurations for specific requirements (e.g., memory limitations or performance constraints) and workload characteristics, we introduced two optimization approaches to improve the storage configuration for a given workload (cf. Section 6.3). We presented a heuristic approach for the workload-driven joint selection of performance and cost-balancing configurations consisting of a data layout and compression scheme. Moreover, we described an attribute decomposition approach that stores a timestamp in multiple columns (cf. Section 6.2.2), which is beneficial for specific access methods and workloads characteristics (cf. Section 6.2.3). Additionally, we proposed an LP model for the multiple column approach to determine an optimized compression scheme for a given workload (cf. Section 6.3.2).

7

Evaluation

This chapter presents the evaluation of the different workload-driven optimization approaches presented in the previous chapters. For the evaluation, we used the dataset and query patterns of a transportation network company (TNC) introduced in Chapter 4. Further information about the experimental setup is provided in Section 7.1. In Section 7.2, we evaluate the different linear programming models (LP) to determine a joint table configuration (cf. Section 5.3). Further, we compare our models against rule-based heuristics and briefly discuss the accuracy and scalability of the approaches. In the following section, we present the evaluation results of the LP models with data tiering decisions (cf. Section 5.4). In Section 7.4, we analyze the impact of the model extensions that consider reconfigurations costs and robust configuration selection. Section 7.5 demonstrates the influence of different configuration decisions for timestamps on the runtime performance and memory consumption. Moreover, we discuss the results (Section 7.6) and threats to validity (Section 7.7). In Section 7.8, we conclude the chapter with a summary.

7.1 Experimental Setup

To evaluate the different proposed optimization approaches for spatio-temporal data management, we use the dataset and workload characteristics of a TNC. We use the research database *Hyrise* (cf. Section 2.3.2) to determine the input values for the LP models (cf. Section 5.2.2) and to measure the performance and memory consumption of different configuration optimizations. In this context, we execute the benchmark queries single-threaded to exclude potential multi-threading and scheduling overheads which might mask the effects of specific tuning decisions. We define the input parameters based on the database's supported encoding and indexing properties. The set of available encodings E consists of five options introduced in Section 5.1. As secondary indexes, we use an approach of Faust et al. [100] that leverages a segment's dictionary to increase space efficiency. Consequently, we have to ensure that indexes are only allowed on dictionary-encoded segments (cf. Section 5.3.6). If not otherwise specified, we focus on single-column indexes in the evaluation, even though the CCD model could reflect multi-column optimizations. A limitation that we further discuss in Section 7.6 is that the index decisions are made for each segment independently. All these settings are database-specific and have to be adopted based on

the specific capabilities of the target database management system. Naturally, this set of tuning options can be flexibly defined and varied. Moreover, our models are able to consider further index structures, storage devices, or encoding approaches for specific application scenarios. The LP framework is of general nature and ensures optimal configurations for a given set of tuning options and associated cost parameters.

If not otherwise indicated, all measurements have been executed on a server equipped with Intel Xeon E7-4880v2 CPUs (2.50GHz, 30 logical cores). Only for the evaluation of the end-to-end measurements of the optimization approaches with data tiering decisions (cf. Section 7.3), we use a server equipped with Intel Xeon Platinum 8180 CPUs (2.50GHz, 56 logical cores) and an Intel P4800X SSD. Our different models are implemented in *Pyomo*, a Python-based open-source optimization modeling language [43, 133]. In contrast to other algebraic modeling languages like AMPL [106], AIMMS [30], GAMS [41], or JuMP [89], *Pyomo* embeds its modeling objects in a full-featured high-level programing language. To solve the LP models, we used the *Gurobi Solver* [121] with 16 parallel threads and no relaxations like optimality gaps or time limits.

7.1.1 Dataset

For the evaluation of the approaches, we use the real-world dataset of a TNC introduced in Section 4.3.1 as a running example. The dataset consists of 400 million observed locations of drivers for three consecutive days in the city of Dubai [279] (raw size 15.9 GB). Compared to other passenger transportation datasets (e.g., New York Taxi Rides [332]), the dataset has a significantly finer granularity as the drivers' position is tracked multiple times per minute. The driver's identifier, timestamp, latitude, longitude, and the driver's status are stored as integer values in the database table for each observed location. Based on the insertion order, a certain temporal ordering of the sample points exists. Due to delayed transmission, there is no guarantee that the table is initially sorted, which could be used in query processing. To evaluate the base LP models (cf. Section 5.3), we used a subset of the data that contains ten million observed locations (raw size 0.4 GB) to reduce the benchmark execution time. We used the entire dataset for the LP models with data tiering decisions (cf. Section 5.4). Unless otherwise noted, we partition the data into ten chunks containing one million or 40 million observed locations depending on the dataset.

As mentioned in Section 4.3.1, the dataset only includes the dispatch-related observed locations of a single city. The volumes of trajectory data that have to be stored and processed by globally operating large TNCs can be significantly larger (e.g., over 100 terabytes per day [96]). Consequently, optimizations that significantly reduce resource requirements (e.g., DRAM capacities) can have a major impact on the operating costs of such systems. As the size of the dataset only influences the input parameters like the runtime of the benchmark queries and the memory consumption, the presented approaches can also be used to optimize significantly larger datasets.

7.1.2 Workloads

Based on the TNC dataset, we defined two different workloads A and B, which are described by the corresponding sets of query templates Q_A and Q_B (cf. Sec-

tion 5.3.1). The workloads are designed to represent the characteristic of spatio-temporal workloads and include domain-specific access patterns of TNCs. TNCs use spatio-temporal data for various applications (e.g., order dispatching, demand predictions, A/B testing, and pricing strategies). In this context, we have different access patterns that, for example, find all available drivers in a specific area and timeframe for order dispatching or provide the data of a particular area for machine learning-based analysis (e.g., demand predictions). These kinds of data requests can be represented as spatio-temporal range or trajectory-based queries, which dominate the majority of spatio-temporal workloads. As our models are of general nature and perform workload-driven optimizations, the approaches do not focus on a specific workload and can be applied to different workloads and datasets.

Both workloads A and B contain six query templates. A query template represents a set of similar database queries. In Appendix A.2, we provide a detailed overview of the different query templates, including the selectivity values of the various scan operations, frequencies, and a list of pruned data chunks. The query template set Q_A has a focus on the temporal aspect of spatio-temporal workloads. Moreover, the workload is dominated by two query templates, which represent 80% of all queries. For the dataset with ten million entries, these queries return the driver's positions with the status free in the last 25 hours (q_{A0}, 30% of the queries in the workload) and all positions of free drivers in a eleven hours time window in an approximate two by two km area (q_{A1}, 50%). These kinds of queries are often used to optimize the routes of drivers and perform demand analysis [286]. Further, the rest of the workload contains four query templates: (q_{A2}, 16%) select all trips of a set of drivers (0.01% of all values) in a timeframe of 25 hours (40% of all values), (q_{A3}, 2.5%) all trips in a timeframe of two days, (q_{A4}, 0.5%) all trips in a specific area (9.5 by 9.5 kilometers), and (q_{A5}, 1%) all trips of a group of drivers (50% of all values) in a specific area and timeframe (25 hours) with the status free.

In contrast to workload A, the frequencies of the different query templates are more equally distributed for workload B. It includes query templates that focus on filtering for specific groups of drivers and areas, which are used for various applications (A/B testing or performance evaluations). The first three query templates of query template set Q_B select the trajectory data of a particular set of drivers: (q_{B0}, 15%) all trips of a group of drivers (0.01% of all values) with the status free, (q_{B1}, 15%) all trips of a group of free drivers (5% of all values) in a specific area of 3.5 by 3.5 km, and (q_{B2}, 10%) all trips of a group of drivers (1% of all values) in a larger area. Further, q_{B3} (25%) selects all trips in a relatively large region (20 by 20 km) and a timeframe of 4 hours. The query template q_{B4} (15%) selects all trips of drivers within a specific set of identifiers that worked in the last 4 hours. Finally, q_{B5} (20%) returns all trips in a small region of 500 by 500 meters for a timeframe of 30 hours. If not otherwise indicated, we execute the queries single-threaded and abstract from result materialization.

7.2 Comparison of the Accuracy, Performance, and Scalability of the Linear Programming Models

For the evaluation of the three LP models introduced in Section 5.3, we analyze the predicted and end-to-end measured results (Section 7.2.1), the performance

compared to a greedy heuristic approach (Section 7.2.2), the impact of fine-grained configuration optimizations (Section 7.2.3), and the scalability of the different LP models (Section 7.2.4).

7.2.1 Predicted vs. End-to-End Results of the Linear Programming Models

In the first step, we evaluate the runtime and memory consumption predicted by the models (cf. Section 5.3) for different memory budgets B compared to the end-to-end measured values for the corresponding table configurations in *Hyrise*. For the end-to-end *Hyrise* measurements, we executed each query of the workload 100 times and determined the average execution time to achieve stable and representative numbers. As displayed in Figure 7.1, the three LP models are able to predict both values for both workloads pretty accurately. In contrast to workload A, the models predict a slightly higher runtime performance than the end-to-end measured values for workload B. But, we can observe the differences between the models are quite accurate. Overall, the measurements show high optimization potentials for fine-grained configuration decisions, especially for limited memory budgets. After a specific memory budget value (e.g., 120 MB for workload A and 140 MB for workload B), the performance only increases slightly or stagnates. Compared to the SMS model's (cf. Section 5.3.4) benchmark results, we can observe that the ISE model (cf. Section 5.3.5) can determine competitive table configurations, especially for larger memory budgets.

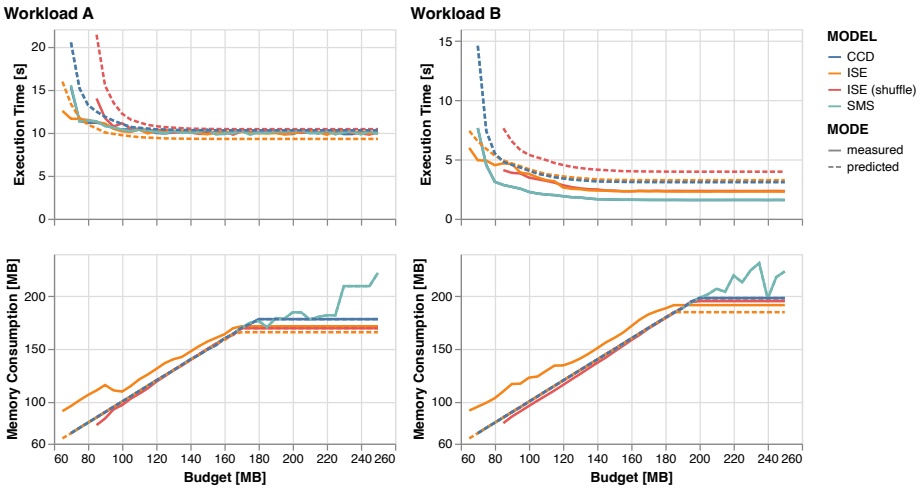

Fig. 7.1: Comparison of predicted (dashed line) and end-to-end measured (solid line) results: performance (top) and memory consumption (bottom) of our three linear programming approaches, cf. CCD, ISE, & SMS (Model), measured in *Hyrise* for the workloads A & B and different memory budgets $B = 65, ..., 250$ MB. Note, the turquoise line covers the blue one as the measured values of the SMS model basically coincide with those of the CCD model. In comparison to ISE, ISE (shuffle) used randomly shuffled data chunks to determine the unsorted benchmark values.

Furthermore, the values of the measured memory consumption in *Hyrise* show that the LP models (except the ISE model) can determine table configurations that use the entire available resources without violating the memory budget constraints. The insert order impacts the data characteristics of specific columns (e.g., driver identifier). As the dataset includes mainly dispatch process-related trajectories, these effects are further enhanced. Therefore, the performance and memory consumption measurements can be inaccurate for a column in the ISE model if a chunk is sorted by another column. To address this issue, the ISE (shuffle) approach determines the input values for the unsorted option based on a shuffled dataset. Before the benchmark queries are executed, the entries in each chunk are randomly shuffled to avoid unintended side effects. As displayed in Figure 7.1 the standard ISE model's table configurations allocate more memory than allowed for various budgets. The ISE model underestimates the memory consumption, especially for lower memory budgets. Here, the ISE model can not consider the effects of a specific sorting column on other columns' data characteristics (e.g., the number of identical values in succession). These characteristics have an increased impact on the compression rate of compression approaches like run-length encoding or frame-of-reference encoding, which are used mainly for low memory budgets. In this context, the faster runtimes for lower memory budgets of the ISE model are caused by the usage of significantly more memory. We can achieve that the ISE model meets the memory budget constraint by applying the ISE (shuffle) approach. As the ISE (shuffle) model has less information about intra-chunk effects, it cannot determine configurations for relatively small memory budgets. We are using the ISE (shuffle) approach for the upcoming evaluations, as it is more consistent in fulfilling the memory budget constraint and able to determine configurations with similar performance. Further, as the predicted results of the SMS and CCD model coincide, it is verified that the CCD and SMS models are equivalent in the case that only single-column indexes are used. For that reason, we limit the evaluation to the SMS and ISE (shuffle) model unless otherwise noted.

Encoding	**Memory Consumption [MB]**	**Execution Time [s]**	
		Workload A	Workload B
Dictionary	181.25	11.493	5.804
Frame of Reference	91.70	30.773	28.694
LZ4	76.10	32.005	30.858
Run-Length	248.92	25.859	26.942
Unencoded	200.01	16.776	11.397

Table 7.1: End-to-end measured memory consumption and execution times of standard table configurations consisting of a single encoding decision for the workloads A & B.

Additionally, we can observe that workload-driven table configuration optimizations can significantly improve the runtime performance and reduce the memory footprint. Compared to standard configurations that use the same encoding approach for each segment of a table (cf. Table 7.1), the LP models enable database administrators to balance performance requirements and budget con-

straints efficiently. For instance, the standard configuration that uses dictionary encoding for each segment – the default setting for various columnar databases – has a memory consumption of 181.25 MB and an execution time of 5.804 s for workload B. In contrast, the table configurations determined by the SMS and CCD model can achieve similar performance results with only 75 MB of main memory. Here, the LP approach can reduce the required memory resources by up to 58 percent. For workload A, we have comparable results achieving similar performance measurements with up to 43 % memory footprint (85 MB). Moreover, the standard configuration that applies LZ4 encoding on each segment has a relatively small memory footprint of 76.1 MB. Compared to the performance of table configurations of the SMS and CCD models for a memory budget of 75 MB, the execution time for workload A is about 2.8 times and for workload B about 6.8 times slower. These numbers demonstrate the high potential of fine-grained workload-driven table configuration optimizations to reduce operating costs and increase the performance of spatio-temporal data management.

7.2.2 Comparison of the Linear Programming Models Against a Rule-Based Heuristic Approach

This section compares the memory consumption and performance of table configurations determined by the LP models against a greedy heuristic approach.

Rule-Based Tuning Heuristics

We seek to evaluate the determined table configurations of our models against standard approaches. As a common approach, we implemented a rule-based greedy heuristic. Similar to the SMS model (cf. Section 5.3.4), the heuristic includes intra-chunk dependencies by taking into account the specific sorting column. As a valid table configuration requires that exactly one sorting option $o \in O$ is selected for each chunk, we have to integrate this constraint into the heuristic. For that reason, we implemented a two-phase approach. In the first phase, we determine a valid base configuration. By selecting the tuning option with the highest benefit $r_{m,n,e,o,i}$ for each chunk, we determine the sorting order of the corresponding chunk. Similar to the approach for the selection of optimized data layouts for timestamps (cf. Section 6.3.1), we calculate the benefit based on a weighted ratio between memory consumption and runtime performance. For each segment (m, n) and each tuning option $e \in E$, $o \in O$, and $i \in I$, we define the benefit r as ($\alpha \geq 0$):

$$r_{m,n,e,o,i} = 1/\left(b_{m,n,e,o,i} \cdot \left(\sum_{\substack{q \in Q, s \in S_q: \\ n_{q,s}=n}} c_{m,n,e,o,i}\right)^{\alpha} \right). \tag{7.1}$$

To calculate the costs $c_{m,n,e,o,i}$ of a scan operation on a segment (m, n) with $m \in M$ and $n \in N$ and the given tuning options, we use the same cost estimations as for the LP models, cp. (5.5). The α value is a factor to define the proportional balancing of the memory consumption and runtime performance for the optimization objective. A higher α value indicates that the performance is higher weighted compared to the memory consumption. Based on the sorting decision of the selected tuning configuration with the highest overall benefit within a chunk, the ordering for the entire chunk is specified. Afterward, the

base configuration is determined by selecting for each segment (m, n) the tuning configuration with the lowest memory consumption for the chunk's given determined sorting option.

In the second phase, we select the tuning option with the largest difference between the benefit of the tuning option and the currently applied configuration of the segment. Afterward, we check if the selected tuning option fits into the remaining memory budget. The table configuration is adapted correspondingly if the configuration change does not violate the remaining memory budget constraint. After a configuration change, the benefits of the tuning options for the corresponding segment have to be recalculated. This step is repeated until no more changes are possible for the given memory budget.

Evaluation Results

As displayed in Figure 7.2, the LP models outperform the greedy heuristics, cf. GH(α), and can leverage the available memory budget more efficiently. Nevertheless, based on the application-specific requirements (e.g., low memory footprint or high performance) that can be defined by the α value, the greedy heuristics' table configurations demonstrate considerable performance improvements compared to standard configurations (cf. Table 7.1). For instance, for a similar memory budget (180 MB), the table configurations determined by the greedy heuristic with an α value of 10 (GH(10)) have a reduced runtime of about 8% for workload A and about 35% for workload B compared to the standard dictionary encoding configuration.

Furthermore, the greedy heuristics' measurements show that the selection of the α value significantly impacts performance and memory consumption. We can observe that the overall performance for different α varies between workloads and memory budget ranges. For example, for $\alpha = 1$, the greedy heuristic does not leverage the available memory budget and performs relatively poorly for all memory budgets for workload A. In comparison, for workload B, there is a wide range of memory budgets (120 MB to 180 MB), where this α value delivers the best performance of the greedy heuristics. By using a two-phase approach, the greedy heuristics select the sorting configuration for the entire table in the first phase. Based on the selected sorting decisions, the calculation of table configurations may not be possible for low memory budgets and specific α values (e.g., for $\alpha = 10$ and $B = 70$ MB).

Moreover, we can observe that the memory consumption and performance stagnate for the heuristic approaches at a particular memory budget depending on the α value. At this point, there are no further tuning options available that provide a benefit. The greedy heuristics with higher performance weighting (e.g., GH(10)) can determine table configurations with only a slight performance decrease for large memory budgets compared to the LP approaches. For workload A and a memory budget of 250 MB, the performance difference between the SMS model and the greedy heuristic (GH (10)) is only 6%. In contrast, the table configurations determined by this heuristic have significantly slower runtimes than the LP models for lower memory budgets (up to 87% performance gain of the SMS model compared to GH (10) for equal memory budgets). Also, the greedy heuristics that focus more on memory consumption (e.g., GH (0.5)) have a significantly lower runtime for more restrictive memory budgets. A reason

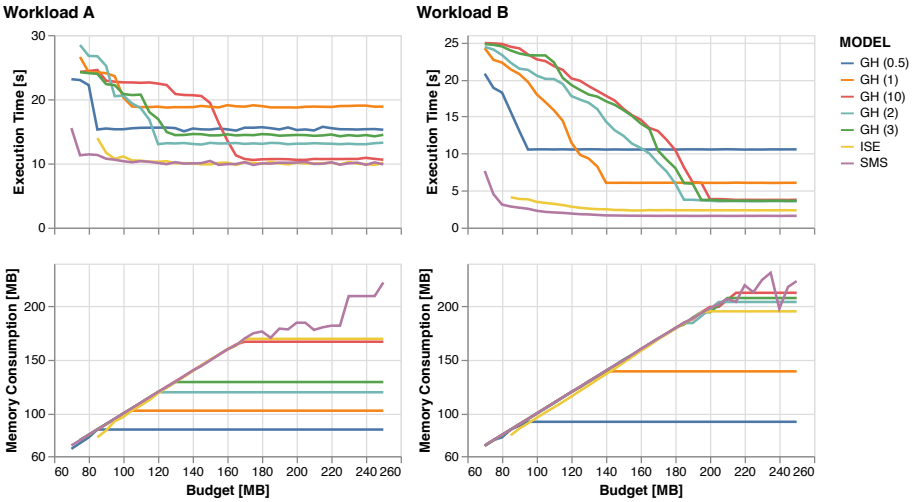

Fig. 7.2: Comparison of end-to-end measured execution time (top) and memory consumption (bottom) for workload A (left) and workload B (right) of the LP models (SMS, ISE) compared to the greedy heuristic approach with different α values (GH (α)).

for this behavior is that greedy heuristics with a higher α value spend a significant amount of memory on optimizing the performance of a limited number of segments.

Overall, we have shown that our LP-based configuration decisions are superior to rule-based heuristics. Our models achieve up to 50% increased performance for a given memory budget, or a comparable runtime is obtained with up to 55% less required main memory for workload A. For workload B, the LP approach achieves up to 82% increased performance by equal memory size or reduces the memory footprint up to 56% by similar performance compared to the best greedy heuristic for a given memory budget.

Detailed Configuration Analysis

Figure 7.3 shows specific table configurations computed by the different approaches for workload B. In each row, the determined table configurations of the approaches (SMS, ISE, and GH(10)) are visualized for four ascending memory budgets. For each table configuration, we depict the individual selected tuning option for each segment. In this context, a segment is determined based on the corresponding chunk (rows) and the column (columns). The different columns of the table are represented by ids, where the mapping is the following: (0) drivers identifier, (1) latitude, (2) longitude, (3) timestamp, and (4) status. The color of each segment determines which encoding is applied. Additionally, a rectangle in the top left corner indicates that the corresponding segment is indexed and a triangle in the top right corner shows that the chunk is sorted by that column. To identify specific partitions of the table with a high memory footprint, the circle in the middle of each segment displays the allocated memory of the corresponding segment.

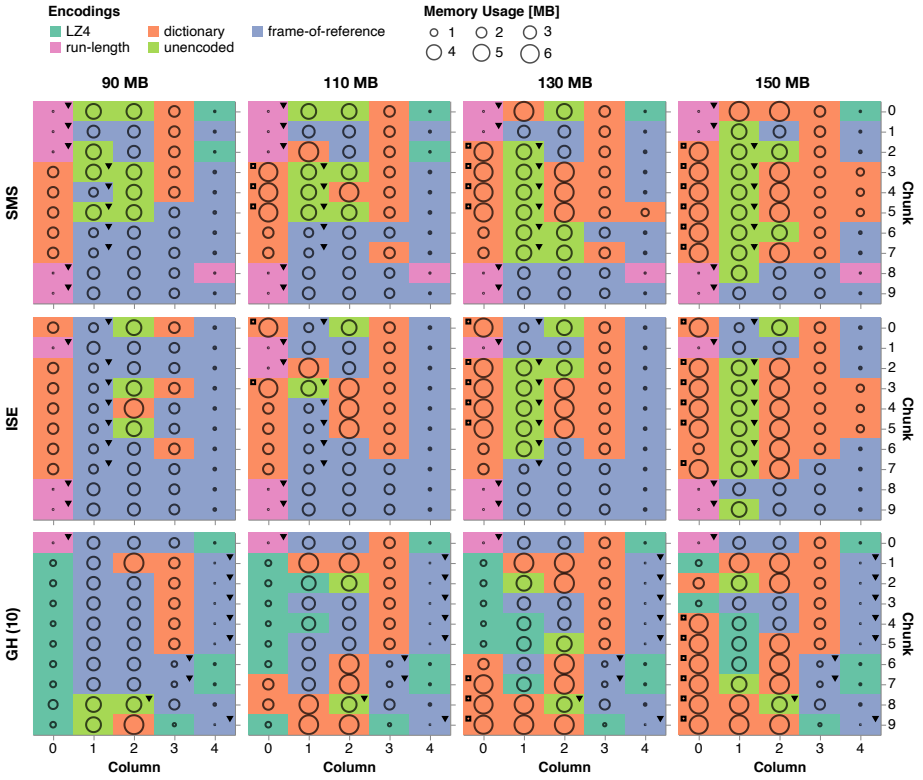

Fig. 7.3: Visualization of table configurations computed for five columns and ten chunks by the SMS model, ISE model, and the greedy heuristic with α=10 for four different memory budgets (90 MB, 110 MB, 130 MB, and 150 MB). A rectangle in the top left corner indicates that the corresponding segment is indexed and a triangle in the top right corner shows that the chunk is sorted by that column. Additionally, the color of a segment represents the applied encoding and the size of the circle indicates the consumed memory by a segment.

We observe that the selected sorting configuration varies strongly between the greedy heuristic and the two LP models (ISE & SMS). As the greedy approach determines the sorting configuration initially for each chunk based on the tuning option with the highest overall benefit, this selection can be suboptimal for lower memory budgets. In contrast, the LP models adopt the sorting configuration with increasing memory budgets. For instance, the third chunk in the SMS as well as in the ISE model is sorted by the first column (the driver identifier column) for the two first memory budgets (90 & 110 MB). For the last ones, the LP models determined that it is more beneficial to change the sorting order of the chunk. The decreased scan performance due to the sorting order's change is compensated by creating an index on the previously sorted segment. For various workload scenarios, we observed that the sorting decision per chunk is based on a segment's access frequency and compression ratio for more restricted memory budgets. With increasing memory budgets, indexes are

applied to segments with low selectivity queries, enabling sorting the chunk by another column.

By analyzing the different configurations, we can further identify that some combinations of tuning options are beneficial for the data characteristics of specific columns concerning memory consumption or performance. For instance, run-length encoding applied on sorted segments with a limited number of distinct values (e.g., the driver identifier or status column) leads to a relatively small memory footprint and fast processing times. In contrast, run-length encoding is significantly less efficient if the chunk is sorted by a column with many distinct values (e.g., longitude, latitude, or timestamp column). Consequently, it is only applied to sorted segments or infrequently accessed segments.

Furthermore, Figure 7.3 demonstrates an advantage of the LP approach compared to the greedy heuristic. The heuristic leverages vast amounts of the available memory (e.g., by applying dictionary encoding) to increase the performance of a limited number of segments. Accordingly, LZ4 encoding is applied to various segments even for increased memory budgets, which negatively affects the overall performance (cf. Figure 7.2). In contrast, the LP models more equally distribute the available memory budget on all accessed segments to improve the performance of various segments. Additionally, we can observe that the joint optimization of different tuning options is an advantage, as index structures are already applied for lower memory budgets. In this case, all approaches use indexes to increase the overall performance even though there are various segments that are still compressed with more heavy-weight compression techniques. We observe that all approaches apply various tuning options for different chunks to take into account pruned segments as well as data and access characteristics. The example shows that optimized joint tuning configurations are complex and cannot be determined manually anymore, particularly for larger problems (e.g., datasets, workloads, or number of tuning options).

7.2.3 Impact of Fine-Grained Configurations

This section investigates the impact of different chunk sizes and the associated number of chunks on the end-to-end measured workload runtime and memory consumption in *Hyrise*. For the evaluation, we use the ISE model (cf. Section 5.3.5) to determine table configurations for different chunk sizes and memory budgets $B = 70, ..., 250$ MB. We partitioned the data into one chunk (chunk size of 10 million), ten chunks (chunk size of 1 million), and 100 chunks (chunk size of 100 000).

As displayed in Figure 7.4, the fine-grained table configurations with smaller chunk sizes achieve significantly better performance than column-based optimizations. One reason for this is that the database can use chunk pruning to skip entire partitions of the data during query execution [86]. Another reason is that we can optimize the applied tuning configuration of each segment specifically for the segment's access and data characteristics. Consequently, we can select different tuning options for partitions with strongly differing access types. In contrast, to choose a tuning option for an entire column, we have to consider all access patterns on the corresponding column (even if the queries access different sections of the data), which can negatively impact the runtime of several queries.

Furthermore, we can observe that the table configurations for 10 and 100 chunks can leverage the available memory resources more efficiently. For instance, for a memory budget of 160 MB, the determined table configuration for workload A and a chunk size of ten million uses only about 140 MB (87.5%) of the available resources. Compared to improving the configuration of individual chunks with smaller chunk sizes, applying configuration optimizations (e.g., lightweight compression such as dictionary encoding) on a single chunk with ten million entries requires significantly larger amounts of memory. Therefore, we can observe a stepwise memory consumption increase for table configurations with a chunk size of ten million. Another drawback of optimizations on column granularity is that we have to use additional memory resources for infrequently accessed data sections if we apply optimizations to increase performance (e.g., indexes). In contrast to the performance differences between a chunk size of ten million and one million (for workload A up to 75% and for workload B up to 78% reduced runtime by equal memory size), we can only observe a performance increase of up to 10% for workload A and up to 38% for workload B by equal memory between the chunk size of one million and 100 000. Based on the overhead introduced by smaller chunk sizes (e.g., separate dictionaries for segments), there is a boundary that limits the performance increase by minimizing the chunk size.

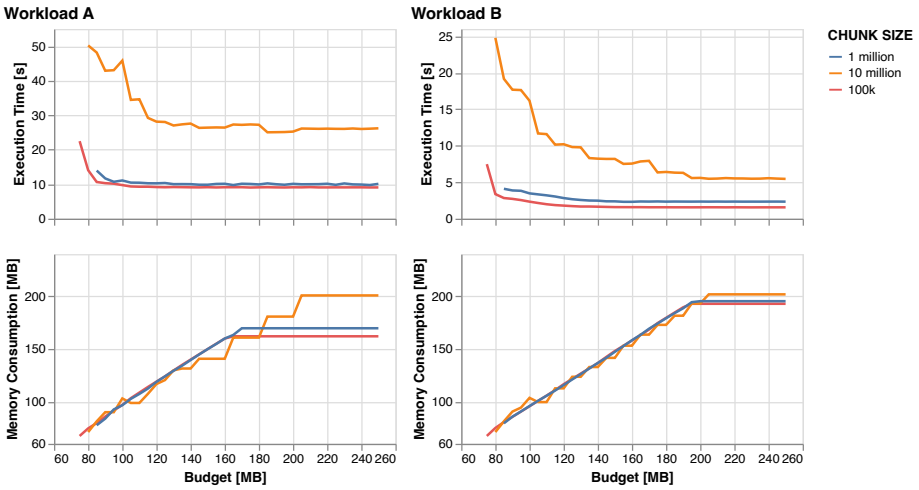

Fig. 7.4: Comparison of the end-to-end measured execution time (top) and memory consumption (bottom) of the table configurations determined by the ISE model for different chunk sizes (1 million, 10 million, and 100k) and memory budgets.

7.2.4 Scalability of the Linear Programming Approach

To analyze the scalability of the different models (ISE, SMS, and CCD), we evaluated the runtime of the solver to compute the table configurations by scaling in the following dimensions: (i) memory budget, (ii) #scan operations that

define the workload, (iii) #chunks, and (iv) #compression and index options. As a general benchmark setup, if not chosen differently, we use the settings described in Section 7.1 and use workload B with $|\cup_q S_q| = 17$ scan operations, where $|M| = 10$, $|E| = 5$, $|N| = 5$, $|O| = 6$, and $|I| = 2$.

Impact of Memory Budgets and Workload Size

At first, we evaluate the solver time of the different models for increasing memory budgets B. As displayed in Figure 7.5 (top left), the ISE model has the lowest solve time with a mean of $0.02s$, followed by the CCD model with $0.1s$. Due to the increased complexity introduced by the number of concrete sorting options (cf. Section 5.3.4), the computation takes significantly longer for the SMS model ($12.6s$ mean solver time). The CCD model's solve time is lower than the one of the SMS model, but the pre-calculation of the chunk configurations takes significantly longer. Note, the number of valid indexing and compression configurations for a chunk k in the example consists of $\binom{|N|}{7776}$ options (cf. Section 7.1). We selected the memory budgets of 100 MB and 200 MB for further evaluation to analyze the differences between a more limited and generous memory budget.

Second, we investigate the capabilities of the models to scale with the workload. Therefore, we evaluated the computation performance for an increasing number of scan operations. As shown in Figure 7.5 (top right), the workload size has no impact on the solver time. As described in Section 5.3.3, the amount of scan operation is only relevant for calculating the parameter c. The computation of the parameters is done in advance and can be executed highly parallel. Based on these observations, we can argue that the different models are capable of handling large workloads.

Impact of Data Size

Due to spatio-temporal data volumes, scalability with regard to data size is crucial. In this context, we analyze our models' solve time for different numbers of chunks. Naturally, the complexity of all three models increases with the number of chunks, see Figure 7.12 (bottom left). The ISE & CCD model scale is linear with the number of chunks, whereby the ISE model's solver times are orders of magnitude lower. For the budget of $B = 100$ MB, the ISE model needs $57ms$ for 100 chunks and $553ms$ for 1 000 chunks to compute the corresponding table configurations. In comparison, the CCD model needs $65s$ for 100 chunks and $637s$ for 1 000 chunks. In contrast, the SMS model does not scale linearly with the cardinality of M; the model computes the table configuration in about $14s$ for 100 chunks and $815s$ (1 000 chunks). The measurements demonstrate that all three models can also determine table configurations for a large number of chunks. Based on the low solver runtimes, the ISE model represents a well-suited approach for large problem domains.

Impact of the Number of Tuning Options

As modern database systems support various compression techniques and index implementations, we investigate the impact of the number of compression and index options on the solver time. For that reason, we generated additional

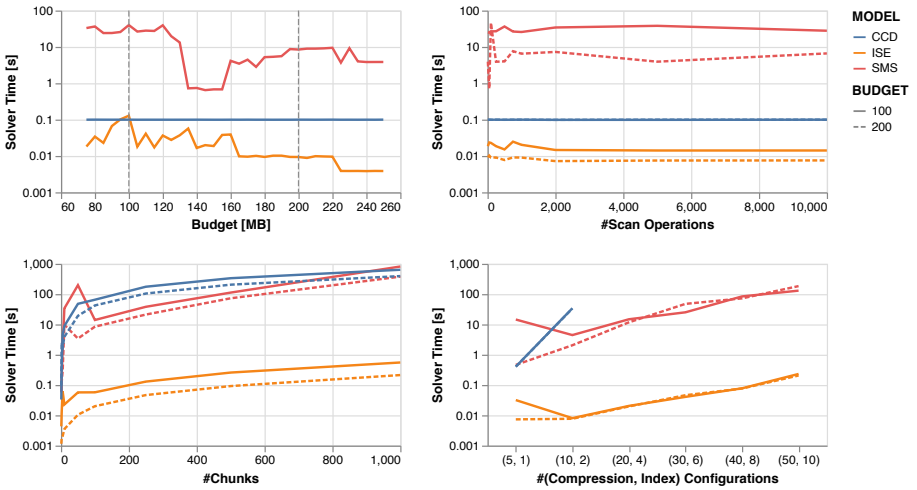

Fig. 7.5: Comparison of the solve time of our three LP models to compute valid table configurations for various memory budgets B =75,..,250 MB (top left) as well as different numbers of tuning options: #scan operation $|S|$ (top right), #chunks $|M|$ (bottom left), and cardinality of the indexing I and compression options E (bottom right) for two selected memory budgets B of 100 MB and 200 MB.

benchmark values (cf. Section 5.2.2) for up to 50 compression techniques and ten different index options. We apply no database-specific limitations for this benchmark so that each index option can be applied in combination with each compression option. As displayed in Figure 7.5 (bottom right), we can observe a slight increase in the solver time for the ISE model up to $231ms$ for the setup with 50 compression techniques, ten indexes, and a memory budget of 100 MB. For the SMS model, the solver time increased from $14s$ for the *Hyrise* example settings to $130s$ for the largest considered setup. Due to memory restriction, it was impossible to determine the results for all setups for the CCD model. As already mentioned in Section 5.3.2, the number of possible chunk configurations $|K|$ can increase quickly. In this context, we further analyzed the applicability of the CCD model for reasonable numbers of $|K|$ based on randomly generated cost inputs. We evaluated that problems with selected six million tuning combinations (per chunk) can be solved in less than 42 seconds, which, in general, shows the applicability of the model. Further, we recall that domain knowledge should be included in specific applications to select only relevant tuning configuration candidates, which can include more complex concepts, such as multi-attribute indexes.

7.3 Linear Programming Approach with Tiering Decisions

In this section, we present the evaluation results for the LP models with data tiering decisions introduced in Section 5.4. To analyze the performance implications of integrated tiering decisions, we increased the dataset of a TNC. As mentioned in Section 7.1.1, the dataset contains 400 million observed locations.

Furthermore, we adopted the queries of workload B to the extended dataset. We adapted the filter predicates of the workload's different scan operations to correspond to the previously defined selectivity values (cf. Appendix A.2). In contrast to the other experiments, we use a server equipped with Intel Xeon Platinum 8180 CPUs (2.50GHz, 56 logical cores) and an Intel P4800X SSD connected via PCI Express 3.0. Note, we only use two storage devices (DRAM and Intel P4800X SSD), but the models can determine configurations for setups with multiple storage devices. The τ value (cf. Section 5.4.3) for each encoding is determined by the mean performance difference of a set of scan operations executed with different selectivity values on both storage devices. As the tiering of segments is based on *UMap* (cf. Section 2.3.2), we have to define a *UMap* page and buffer size. For the user-space page management, *UMap* uses a buffer of pages in DRAM [256]. To ensure that the data has to be read from the slower datastore and avoid caching effects, we selected a small buffer size of 1000 pages and a page size of 128 KiB.

As secondary indexes, we use the approach of Faust et al. [100]. Furthermore, we include multi-column indexes, which are implemented in Hyrise (cf. *compound group-keys* [101]), in the evaluation to demonstrate the capabilities of the CCD model. Both approaches leverage a segment's dictionary to increase space efficiency. Consequently, we have to ensure that indexes are only allowed on dictionary-encoded segments (cf. Section 5.3.6). Besides five single column indexes, we consider 46 multi-column indexes (MCIs) with attribute lengths of 2-4. The MCI set includes all relevant index candidates for the given workload (cf. [166]). For combinations of the four tuning dimensions, we considered about $K = 120\,000$ options per chunk.

7.3.1 End-to-End Results of the Linear Programming Models

As displayed in Figure 7.6, the three LP models are able to improve joint table configurations for the given workload. Overall, we can observe that there are high optimization potentials for fine-grained configuration decisions, especially for lower memory budgets. After a specific DRAM budget value, the performance only increases slightly or stagnates after most of the segments are stored in the main memory. Compared to the SMS benchmark results, we can observe that the ISE model is able to determine competitive table configurations, especially for larger memory budgets.

However, without detailed information about intra-chunk effects, the chunk-based and segment-based ISE models have a 12% decreased performance compared to the other approaches. Also, for the initial case, where all data is stored on SSD, the ISE models have 30% performance decrease. The ISE model cannot consider the effects of a specific sorting column on other columns' data characteristics (e.g., the number of identical values in succession). These characteristics have an increased impact on the compression rate of compression approaches like run-length encoding or frame-of-reference encoding, which are used in particular for low memory budgets. Further, as the SMS model and CCD model without MCI results coincide, it is verified that the SMS model is equivalent to the CCD model if multi-column optimizations are not considered.

In Figure 7.7, we can observe that the different LP models are able to utilize the available DRAM capacities efficiently. Also, the models satisfy the given memory limitations. In contrast to the chunk-based optimization approaches,

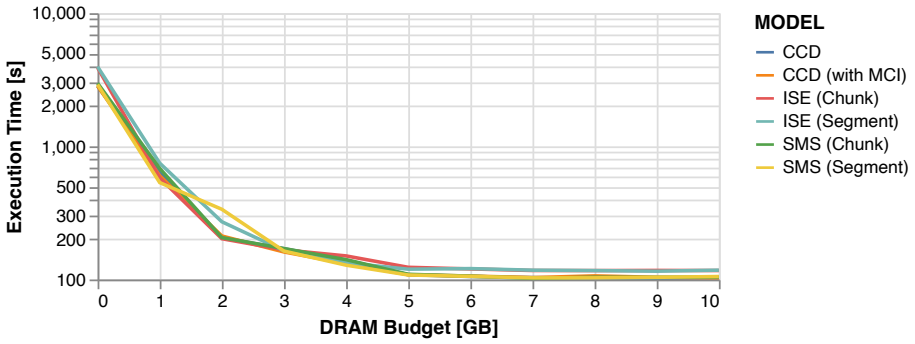

Fig. 7.6: End-to-end measured runtime performance of the table configurations determined by the different LP models for the given workload, a fixed budget of 3 GB on SSD, and increasing DRAM budgets (incl. multi-column indexes (MCI)).

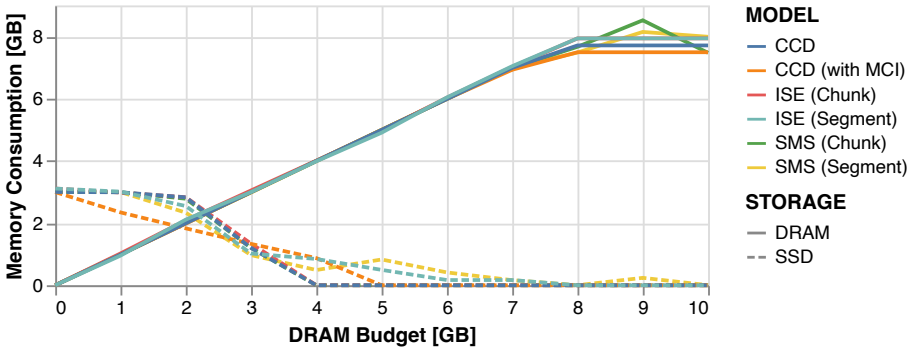

Fig. 7.7: DRAM (solid line) and SSD (dashed line) memory consumption of the table configurations determined by the different LP models for the given workload, a fixed budget of 3 GB on SSD, and increasing DRAM budgets (0-10 GB).

we observe that segment-based approaches store infrequently or never accessed segments even for higher DRAM budgets on SSD to generate more space in DRAM for further optimizations. Finally, we can observe that the SMS and CCD model reached a configuration at 8 GB where no further improvement can be achieved (with more DRAM budget).

In this experiment, segment-based approaches have no significant advantage compared to chunk-based approaches. The reason for that is that our workload accesses 92% of all segments. For workloads that query only a limited set of segments, we can expect better performances for the segment-based approach as less DRAM budget has to be used for never accessed data. By storing infrequently accessed segments on the slower storage, lightweight compression techniques or additional index structures can be applied for frequently accessed segments in the main memory.

The table configurations with MCIs have a slightly better performance for the memory budgets of 3 GB and 4 GB. For the other memory budgets, we

cannot observe a performance gain for the CCD model with MCI. Based on the implementation of MCIs in *Hyrise*, which is optimized for primary key access, the runtime improvements provided by MCIs for spatio-temporal workloads compared to other data structures (e.g., single-column indexes) are not that significant, especially concerning the increased memory footprint of MCIs. Naturally, the performance difference could be increased if the database system supports specifically optimized index structures for spatio-temporal data [215]. In the evaluation, we observed that MCIs are often used on the SSD to mitigate access time differences. The usage of MCIs requires that columns positioned at the beginning of the index are frequently accessed together. While this is typically the case for OLTP workloads, where compound primary keys are accessed with high selectivity, it is less often the case for spatial workloads. MCIs are, in this case, too large for DRAM as they cannot always be used but are still helpful when placed on slower storage devices, where their added storage costs are less of a problem. In DRAM, the LP models apply mainly single-column indexes, which have a significantly reduced memory footprint. Interestingly, the selected MCIs do not only focus on latitude/longitude but rather cover distinct columns such as the driver ID.

7.3.2 Comparisons Against Rule-Based Tuning Heuristics

Similar to Section 7.2.2, we seek to evaluate the determined table configurations of our LP models with tiering decisions against a rule-based heuristic approach. Therefore, we implemented two rule-based greedy heuristics. Equivalent to the SMS model (cf. Section 5.4.4) and ISE model (cf. Section 5.4.5), we developed a heuristic that considers segment-based tiering decisions and one that integrates chunk-based tiering decisions. Both heuristics include intra-chunk dependencies by taking into account the specific sorting column. As a valid table configuration requires that one sorting option $o \in O$ is selected for each chunk, we must integrate this constraint into the selection process of the base configuration. Hence, we apply a two-phase approach. In the first phase, we determine a base configuration with minimal memory consumption by selecting the tuning configuration with the lowest memory consumption for each segment. Additionally, we define for the base configuration that all segments are stored on the storage device with the highest latency, in this case, the SSD. In this context, each chunk's ordering is determined by the tuning option with the highest overall benefit $r_{m,n,e,o,i,b}$ for the corresponding chunk. In the second phase, we calculate the benefit for each tuning option and adopt the table configuration iteratively based on the calculated benefits and the available memory budget. For each segment (m, n) and tuning option $e \in E$, $o \in O$, $i \in I$, $b \in B$, we define r as ($\alpha \geq 0$):

$$r_{m,n,e,o,i,b} = 1 / \left(\phi_{m,n,e,o,i} \cdot \left(\sum_{\substack{q \in Q, s \in S_q: \\ n_{q,s} = n}} c_{m,n,e,o,i,b} \right)^{\alpha} \right). \tag{7.2}$$

To calculate the costs $c_{m,n,e,o,i,b}$ of the scan operations, we use the same cost function as for the LP models, cp. Equation (5.21). The α value is a factor to define the proportional balancing of the memory consumption and runtime performance within the objective. The heuristics optimize the memory budgets B_d for all $d \in D$ separately, starting with the device with the lowest latency. We select the tuning option with the highest benefit compared to the currently

applied configuration. Afterward, we check if the selected tuning option fits into the remaining storage budget of the device and change the configuration correspondingly if the change does not violate the memory budget constraint. The steps are repeated until no more changes are possible. Afterward, the memory budget of the next storage device is optimized. For the chunk-based approach, we additionally ensure that only entire data chunks can be transferred to a tier, cf. Constraint (5.29).

As displayed in Figure 7.8, the LP models outperform the greedy heuristics, cf. H(α), and can use the available memory budget more efficiently (cf. Figure 7.9). The measurements show that greedy heuristics struggle to compute configurations for scenarios with limited DRAM capacities. Here, the SMS model can achieve a runtime of 200s for a DRAM budget of 2 GB, which is only 10% of the fastest runtime of a table configuration determined by a heuristic for the same budget. The heuristics need significantly more DRAM budget (7 GB) to achieve comparable performance results. In this case, the LP model's table configurations use only 29% of the memory budget required by the table configuration of the heuristic (H-10) with segment-based tiering decisions. We observe that the heuristics use the majority of the available DRAM resources to optimize single chunks or segments. Consequently, large parts of the data have to be stored on the significantly slower SSD, even for higher DRAM capacities (cf. Figure 7.9). Additionally, the greedy heuristics' measurements show that the selection of the α value has a significant impact on performance and memory consumption. As the heuristics select the sorting configuration for each chunk in the initial phase, the sorting decisions can be sub-optimal for different budgets.

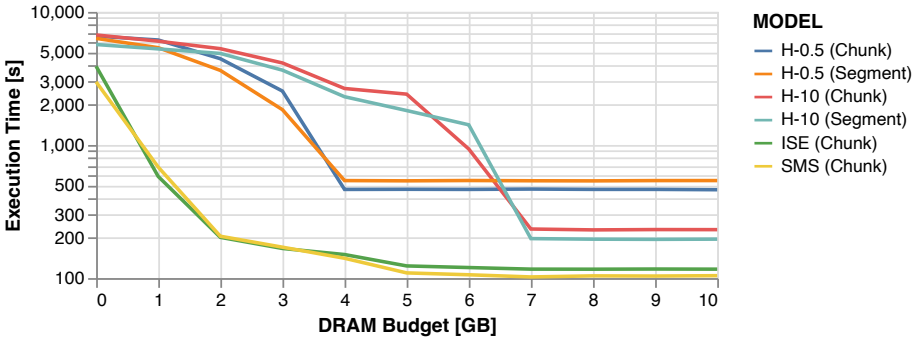

Fig. 7.8: End-to-end measured runtime performance of the table configurations determined by the heuristic approaches compared to the LP models for the given workload, a fixed budget of 3 GB on SSD, and increasing DRAM budgets (0-10 GB).

7.3.3 Comparisons Against Existing Approaches

In this section, we compare our LP approach against existing solutions. A comparison is not straightforward as various joint optimization approaches focus on other aspects (e.g., materialized views, partitioning, or knob configurations).

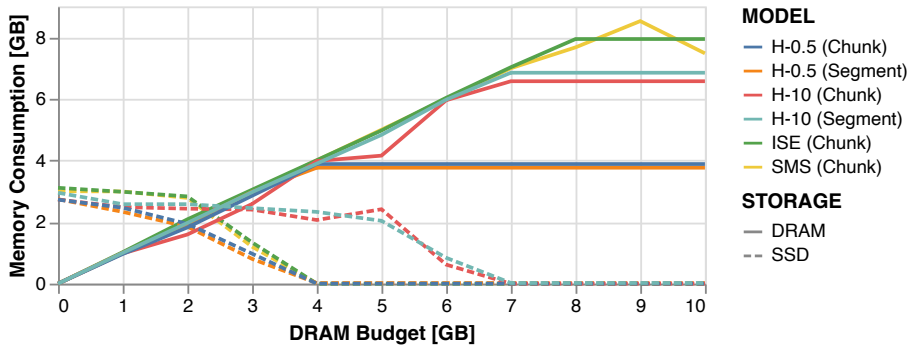

Fig. 7.9: DRAM (solid line) and SSD (dashed line) memory consumption of the table configurations determined by the heuristic approaches for the given workload, a fixed budget of 3 GB on SSD, and increasing DRAM budgets (0-10 GB).

In Figure 7.10, we evaluate the SMS model regarding (i) the tiering dimension [348] and (ii) a consecutive joint tuning approach [169]. Concerning the tiering dimension, we re-implemented the *capacity mode* of *Mosaic's* presented linear optimization strategy (LOPT) [348]. This LP approach is based on Umbra and optimizes the data placement on columnar granularity. As *Mosaic* is optimized for data placement decisions on SSD and HDD devices, the cost model is based on the throughput and accessed data size. Furthermore, parallel scan operations on multiple devices are allowed. To represent the query processing in *Hyrise*, we adopted the cost estimation that scan operations are executed sequentially, where each scan only processes the qualifying positions of the previous scan. An advantage of *Mosaic's* cost model is that less detailed information about the workload (e.g., selectivity of queries) is required, and no benchmarking queries have to be executed. Based on the prerequisite that *Mosaic* requires all segments stored on one device to have the same encoding, we selected the best encoding for each device for the given benchmark setup. For a fair comparison, we partitioned the data to enable pruning during query execution and applied the same configuration for all segments of a column. Based on these restrictions, a relatively large amount of memory has to be used to store a single dictionary-encoded column in DRAM. As displayed in Figure 7.10, for small memory budgets, we can observe a comparably weak performance as significant parts of the available DRAM were unused by *Mosaic* as the remaining DRAM budget was not big enough to store another column. Furthermore, the model is not designed to optimize other aspects (e.g., sorting or indexing), which have a significant impact on the overall performance.

Furthermore, we compare the SMS model against the joint tuning approach of Kossmann and Schlosser [169], which proposes a heuristic to tune different dimensions in a subsequent manner. The authors use predefined memory budgets for each optimization step in this context. We evaluated this approach as follows: Starting with an untuned base configuration, we choose a specific tuning order for our four considered dimensions. Given a budget, we solve our model for the first dimension by freeing the corresponding variables and fixing those for the other dimensions. The same is done for the remaining tuning dimen-

sions. Instead of one LP, for [169], we solve four LPs of smaller size. Further, we iterated over all plausible tuning orders to not miss the best possible one (i.e., in our case, we obtained the order: encoding, sorting, tiering, indexing, which is consistent with the order derived for the three-dimensional tuning example used in [169]). For the index selection, we reserved 10% of the available budget in the previous steps. We discover that our LP approach outperforms [169] by 15% for larger DRAM budgets and achieves up to 6.8 times better performance for more restrictive memory budgets (cf., Figure 7.10). Naturally, the final results also depend on the distribution of the total budget (e.g., budget for indexes) to the four steps. Recall that this reveals another advantage of our model as the nontrivial optimization of the distribution of the budgets is included in our model.

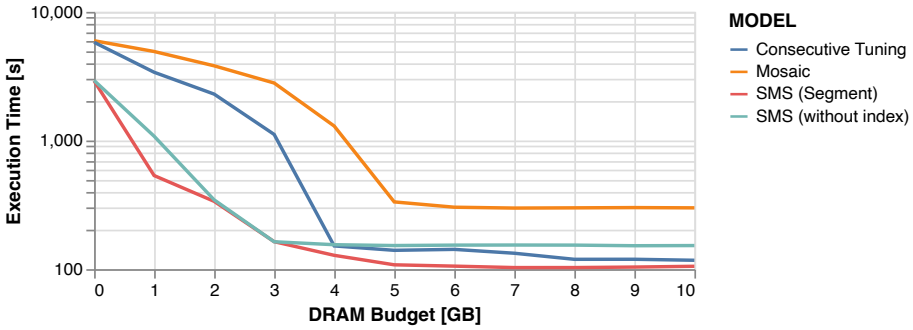

Fig. 7.10: Comparison of the end-to-end measured runtime of the table configurations determined by the *capacity mode* of Mosaic, a consecutive tuning approach, and the SMS model for a fixed budget of 3 GB on SSD, and increasing DRAM budgets (0-10 GB).

7.3.4 Detailed Configuration Analysis

Figure 7.11 shows the specific table configurations computed by the SMS model, the heuristic approach with $\alpha = 10$ (H-10), the CCD model with MCIs, and the consecutive tuning approach. The configurations were determined for a budget of 3 GB on both storage devices. For each configuration, the segments defined by the chunk (y-axis) and the column (x-axis) in the top area are stored in DRAM and the bottom ones on SSD. The color of each segment indicates the applied encoding and the circle in the middle represents the consumed memory. If an index is applied to a segment (indicated by a black rectangle or diamond in the top left corner), the consumed memory of the index is included in the size of the circle. A sorted column is represented by a black triangle in the top right corner of a segment. Note, for the CCD model, we force the tiering decision to be the same within a chunk.

We observe that the selected sorting configuration varies strongly between different approaches. As the greedy heuristic determines the sorting configuration based on the segment with the highest overall benefit for each chunk in the first phase, this selection can be sub-optimal for lower budgets. Additionally, we can see that the greedy heuristic uses significantly more of the available

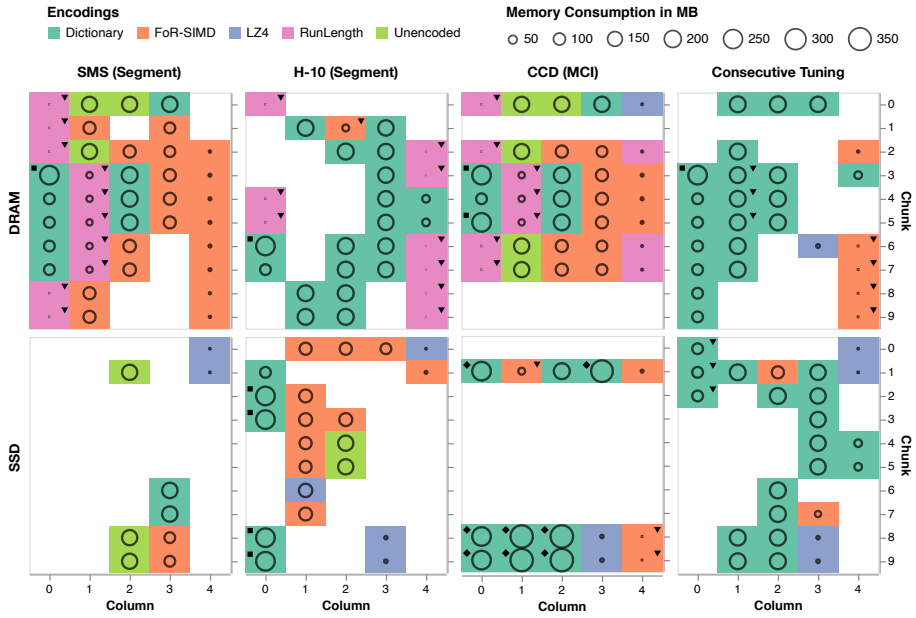

Fig. 7.11: Comparison of the table configurations for five columns and ten chunks: (i) SMS model, (ii) H-10 heuristic with α =10, (iii) CCD model with MCI, and (iv) consecutive tuning approach for a memory budget of 3 GB DRAM (top) and 3 GB SSD (bottom). A triangle in the top right corner of a segment indicates the sorted column of a chunk. A square in the top left corner indicates a single-column index for the segment and a diamond refers to an MCI.

DRAM budget to improve the performance of specific segments, which leads to an improved runtime for scan operations on these segments, but also consumes a significant part of the available main memory. In contrast, the SMS model distributes the available memory on more segments and selects compression techniques with higher compression ratios (e.g., frame-of-reference encoding) to store more segments in DRAM. The CCD model applies different MCIs to mitigate the increased SSD access latency. In contrast, the CCD model uses single-column indexes in DRAM, which have a lower memory footprint. For the consecutive tuning approach, we can observe the problems of the one-by-one execution of tuning steps. In the first step, the LP leverages the entire memory budget to optimize the encoding configuration. It selects dictionary encoding for various segments to increase the runtime performance. Based on the memory footprint of these segments, the LP for the tiering decision can only transfer a limited number of segments to DRAM. Consequently, a relatively large number of segments have to be stored on SSD.

7.3.5 Scaling of the Linear Programming-Based Approach

To analyze the impact of scalability of the different models with integrated data tiering decisions (ISE, SMS, and CCD), we evaluated the runtime of the solver to compute the table configurations by scaling in the following dimensions: (i) the

number of chunks, (ii) the number of scan operations that define the workload, (iii) the number of storage devices, and (iv) the number of compression and index options. As a general benchmark setup, if not chosen differently, we use the settings described in Section 7.1 with $|Q| = 6$, $|\cup_q S_q| = 17$ scan operations, where $|M| = 10$, $|E| = 5$, $|N| = 5$, $|O| = 6$, $|I| = 2$, and $|B| = 2$. The memory budgets are defined as 3 GB for DRAM and 3 GB for SSD.

Impact of Data Size

Naturally, the complexity of all three models increases with the number of chunks, see Figure 7.12 (left top). The ISE & CCD model scale linear with the number of chunks, whereby the ISE model's solver times are orders of magnitude lower. For the given budgets, the segment-based ISE model needs $24ms$ for ten chunks and $613ms$ for 500 chunks to compute the configurations. In comparison, the CCD model needs $22s$ for ten chunks and a similar amount of time for 500 chunks. In contrast, the SMS models do not scale linearly with the cardinality of M; the models compute the table configurations in about $1.5s$ for ten chunks and $205s$ (500 chunks) for the segment-based approach and $2.5s$ for ten chunks and $413s$ (500 chunks) for the chunk-based approach.

Impact of Workload Size

We evaluated the computation performance for an increasing number of scan operations. As shown in Figure 7.12 (bottom left), the workload size has no impact on the solver time. As described in Section 5.4, the amount of scan operation is only relevant for the calculation of the parameter c. The computation of the parameters can be executed highly parallel. Based on these observations, we can argue that the different models are capable of handling large workloads. Another observation is that the segment-based approaches have a faster runtime compared to their chunk-based equivalents.

Impact of the Number of Storage Devices

We evaluate the impact of the number of storage devices on the solver time. Modern systems feature a diversity of storage devices, from NVMe and SSD to network-interconnected memory and HDD [256]. The LP models are designed to support setups with multiple storage devices. We defined a base budget of 8 GB and divided this budget by the number of available storage devices. The storage penalty $\tau_{e,i,b}$ is set for all encodings (e) and index configurations (i) to a fixed value, defined by the storage device's number (b). As displayed in Figure 7.12, all five models can determine table configurations for infrastructures with eight different storage mediums in a suitable time (less than $50s$). Here, we have to consider that the solver times are also depending on the specific setup (e.g., latency between storage devices and storage capacities).

Impact of the Number of Tuning Options

As modern database systems support various compression techniques and index implementations, we investigate the impact of the number of compression and

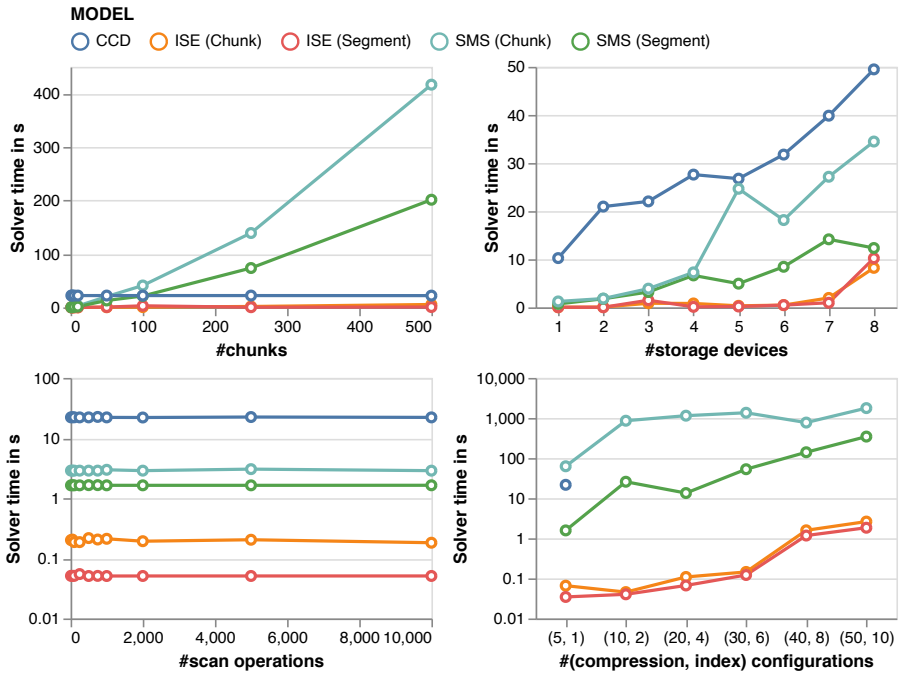

Fig. 7.12: Comparison of the solver times of the five versions of our LP models for the specified scenarios to compute valid table configurations for different numbers of tuning options: #chunks $|M|$ (top left), #scan operations $|\cup_q S_q|$ (bottom left), #storage devices $|D|$ (top right) and cardinality of the indexing $|I|$ and compression options $|E|$ (bottom right).

index options on the solver time. We generated additional values for up to 50 compression techniques and ten index options. Moreover, we apply no database-specific limitations for this benchmark so that each combination of encoding and index options can be used. In Figure 7.12 (bottom right), we observe a slight increase in the solver time for the ISE model up to $2.7s$ (chunk-based) and $1.9s$ (segment-based) for the setup with 50 compression techniques and ten indexes options. For the segment-based SMS model, the solver time increased from $1.5s$ for the *Hyrise* settings to $348s$ for the largest considered setup. Similar to the scalability evaluation of the base LP models (cf. Section 7.2.4), it was not possible to determine results for all setups for the CCD model. As already mentioned in Section 5.4.2, the number of possible chunk configurations $|K|$ can increase quickly. In this context, we further analyzed the applicability of CCD for reasonable numbers of $|K|$ based on randomly generated cost inputs. The evaluation showed that the CCD model solves problems with selected six million tuning combinations (per chunk) in less than $60s$, which, in general, shows the model's applicability. Further, we recall that domain knowledge should be included in specific applications to select only relevant tuning configurations, which can consist of more complex concepts, such as multi-attribute indexes. Also, a hybrid optimization process could be possible, in which the CCD model compares the table configuration determined by the SMS or ISE model with several adopted

versions that include multi-column optimizations (e.g., advanced sorting strategies).

7.4 Model Extensions of the Linear Programming Approach

This section evaluates the proposed model extensions to consider reconfiguration costs (cf. Section 5.5.1) and robustness (cf. Section 5.5.2).

7.4.1 Extension: Reconfiguration Costs

Based on the LP approach, minor workload or infrastructure (e.g., DRAM capacities) changes can lead to significant reconfiguration costs [169]. All individual tuning optimizations produce modification costs (e.g., changing the sorting order of a chunk consumes time and resources). To determine optimized configurations for the given input parameters, the models often apply numerous reconfigurations with only a minor impact on the overall performance. However, huge modification costs are not desirable in practice. By considering modification costs in the models (cf. Section 5.5.1) we are able to identify and perform only minimal-invasive modifications. There are various metrics to determine modification costs (e.g., reorganized data, estimated time, or selected by database administrator) [220, 295, 354].

For the evaluation, we defined for each modification a cost factor. A change of the applied sorting configuration has the highest costs, and a segment reallocation has the lowest. The database administrator can specify these costs per reconfiguration operation individually. For the given base configuration determined by the SMS model for a DRAM budget of 3 GB, we increased the available memory budget by 1 GB and evaluated the configurations for different α values. The α enables the balancing of the trade-off between performance and reconfiguration cost and depends on different aspects (e.g., the time interval between optimizations or availability of resources). The results of the end-to-end measured performance show that compared to the best possible α=0 solution, the α=1 configuration has only a 2.25% performance decrease with about 40% fewer reconfiguration costs. For α=5, there is a performance reduction of 17.6% and a reduction of about 78% of the reconfiguration costs.

7.4.2 Extension: Robust Configuration Selection

As the prediction of precise future workloads is complicated, table configurations optimized for a specific workload can lead to unsatisfactory performance if the actual workload strongly differs from the expected one. Consequently, we introduced a worst-case optimization approach for our LP models (cf. Section 5.5.2) to determine robust table configurations for different workload scenarios specified by the database administrator. By considering multiple potential workload scenarios in the table configuration optimization, we can avoid critical downturns in performance caused by changes in the workload characteristics.

To evaluate the extension of the models, we defined three different workload scenarios based on the workload introduced in Section 7.1.2. Workload 1 represents an increase in the usage of a specific application. Therefore, the frequency

of query q_4 is three times higher than in the standard workload. In the second workload, we removed query q_2 to represent incomplete information about the workload. The third workload contains an additional query that accesses two segments of the status column, which are not accessed by one of the queries in the standard workload. We determined a workload-specific optimized table configuration for each of these workloads for a DRAM budget of 3 GB and an SSD budget of 3 GB using the segment-based SMS model. In Figure 7.13, we denote these configurations correspondingly as $w1$, $w2$, and $w3$. Furthermore, we include a table configuration determined for the standard workload ($w4$) and a robust configuration for the same memory budgets. To determine the robust table configuration, the optimized table configuration of the LP model is based on all four potential workload scenarios.

As displayed in Figure 7.13, we compared the end-to-end measured execution times of the five table configurations for the three different workload scenarios. Here, we can observe that for each workload scenario, the corresponding workload-specific table configuration achieves the best performance. Moreover, we can see that these configurations have a significantly higher runtime for other workload scenarios. For instance, for the second workload scenario, the table configuration $w2$ has the best performance as it is specifically optimized for the workload and can use more of the available memory budget to optimize the segments accessed by the workload scenario. We can observe that this table configuration has an increased runtime for the other workload scenarios as various segments stored on SSD have to be accessed. In contrast, table configuration $w3$ has the best performance for the tired workload scenario, but performance downturns for workload scenario two as large amounts of the available resources are used to optimize the segments of the two additional query templates, which do not occur in workload scenario two. Compared to the workload-specific table configurations, the robust table configuration has a longer runtime for all three evaluated workload scenarios. Based on the worst-case optimization approach, it provides an adequate runtime for all potential workload scenarios and enables database administrators to avoid significant performance drops caused by deviant workload characteristics.

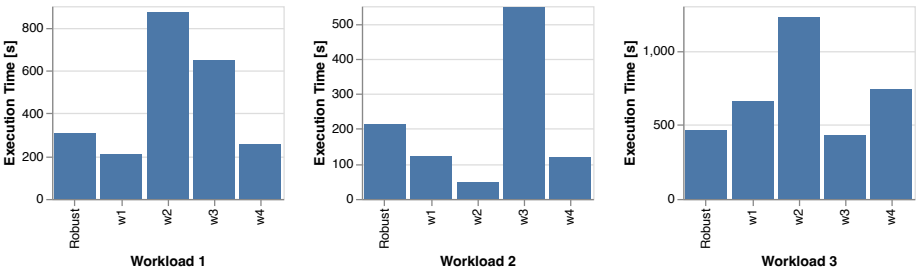

Fig. 7.13: Comparison of execution times of table configurations for three different workload scenarios. We used the segment-based SMS model for each workload to determine an optimized table configuration ($w1$, $w2$, and $w3$). We compare the end-to-end measured performance of these table configurations for each workload scenario against a robust configuration (robust) and the configuration determined for the standard workload ($w4$).

7.4.3 Computation Time Impact of the Model Extensions

The extensions to consider multiple potential workload scenarios and reconfiguration costs increase the complexity of the LP models. Therefore, we investigate the impact of the different extensions on the time the solver needs to compute an optimal configuration. For the evaluation, we determined the table configurations for a memory budget of 3 GB DRAM and 3 GB SSD using the segment-based ISE and SMS models.

As displayed in Table 7.2, the integration of reconfiguration costs only slightly impacts the solver times. Depending on the defined reconfiguration costs Δ and the chosen α value, the solver times of the models with reconfiguration costs can even be faster than the baseline of the LP models without extensions. For instance, the solver can efficiently exclude a large number of configurations from the solution space if the reconfiguration costs are high. Furthermore, we can observe the expected increase in the solver runtimes for the robust selection approaches based on the number of different workload scenarios $|W|$ that must be considered in the optimization. For the evaluation, we determined 100 different variations of the workload described in Section 7.1.2. Each workload scenario contains the same six query templates and randomly chosen distributions of the frequencies. With increasing workload scenarios $w \in W$, minimizing the variable Z is getting costlier. As the database administrators must define these different workload scenarios, we expect only a limited number of potential workloads, leading to an increased but manageable solver runtime.

Model	**Base**	**RC**	**Robust Selection**				**Robust Selection & RC**			
			$\lvert W\rvert = 5$	10	50	100	5	10	50	100
ISE	0.03 s	0.03 s	0.09 s	0.06 s	0.52 s	0.68 s	0.08 s	0.07 s	0.37 s	0.64 s
SSD	1.83 s	2.13 s	2.24 s	6.14 s	8.02 s	23.42 s	4.93 s	4.01 s	10.09 s	24.45 s

Table 7.2: Measured solver runtimes for the ISE and SMS models with tiering decisions for (i) the standard models without extensions (Base), (ii) the models with reconfiguration costs (RC), (ii) the models with robust table configuration selection for different numbers of workload scenarios $|W|$, and (iv) the models that include robustness as well as reconfiguration costs.

7.5 Impact of Optimized Timestamp Storage Layouts for Spatio-Temporal Data

In this section, we evaluate the proposed workload-driven optimization approaches to store timestamps in columnar in-memory databases (cf. Section 6.3). First, we analyze the selected configurations by the heuristic approach (Section 7.5.1). Second, we show the results of the LP approach (cf. Section 6.3.2) to optimize the compression scheme for a given workload (Section 7.5.2).

7.5.1 Heuristic Approach for the Combined Data Layout and Compression Scheme Selection

To evaluate the heuristic selection approach and illustrate the impact of different α values, we analyze the determined configurations for three chosen α values. In Figure 7.14, we visualize the selected data layout and compression techniques for a given α value and workload distribution. The workload is defined based on the queries introduced in Section 6.2.3. Based on the memory consumption (cf. Figure 6.2) and measured query runtimes (cf. Figure 6.4) of the different data layouts and compression approaches, the heuristic selects the combination with the highest benefit determined by Equation (6.2). Further compression techniques and data layouts can be added based on the properties of the used database system. For a value $\alpha \leq 1$, the memory consumption is higher weighted than the runtime performance. Consequently, for all three workload distributions, the Unix timestamp data layout in combination with LZ4 encoding is selected for $\alpha = 0.1$, as this combination has the lowest memory footprint (cf. Figure 6.2). For the first workload distribution, the performance of query q_2 dominates the workload runtime, which is a range query that queries a continuous time interval. The Unix timestamp data layout can achieve a relatively low runtime for such access types as only a single column has to be scanned (cf. Figure 6.4). Consequently, for all three α values, the Unix timestamp data layout is selected. As for $\alpha = 5$ the performance is significantly higher weighted as the memory consumption dictionary encoding is chosen compared to LZ4 encoding for $\alpha = 5$. For the two other workload distributions, the separated date/time format is selected for $\alpha = 5$ based on the superior runtime performance for query q_0, and q_4 with applied dictionary encoding. This data layout has a relatively high memory consumption, so the multiple columns approach is selected for $\alpha = 2$, representing a suitable tradeoff between performance and memory footprint.

Workload Distribution in Percent					Selected Data Layout and Encoding		
Q0	Q1	Q2	Q3	Q4	α = 0.1	α = 2	α = 5
20	20	20	20	20	Unix LZ4	Unix LZ4	Unix Dictionary
15	1	34	40	10	Unix LZ4	Multiple Columns Run-Length	DateTime Dictionary
50	1	9	20	20	Unix LZ4	Multiple Columns Run-Length	DateTime Dictionary

Fig. 7.14: Visualization of the selected data layout and compression technique by the heuristic (cf. Section 6.3.1), for three different workload distributions based on the queries introduced in Section 6.2.3 for three different α values.

7.5.2 Optimized Compression Scheme Selection for Timestamps Stored in the Multiple Columns Data Layout

For the evaluation of our LP approach to optimize the compression scheme for the multiple columns data layout (cf. Section 6.3.2), we define a set of query

templates. The set consists of queries that filter for a specific timestamp or timeframe and queries that return all values in a defined timeframe over multiple days. We have the following query distributions in the benchmark: all timestamps of a day (40 percent of all queries), a specific timestamp (5%), and a specific timeframe of 20 seconds (5%). Furthermore, we have queries that select all entries on each day in a timeframe of 30 minutes (20%), a timeframe of two hours (15%), and before noon (15%).

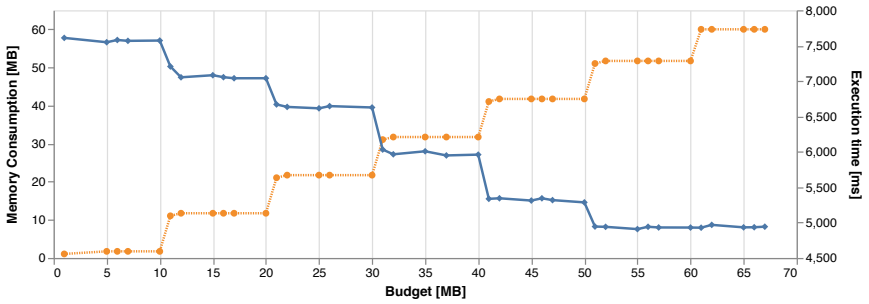

Fig. 7.15: Comparison of the measured runtime performance (solid blue) line and memory consumption (dashed orange) of the applied compression scheme determined based on the given workload for different memory budgets $B = 0, ..., 70$ MB.

We used the model to determine the compression scheme for the different columns of the multiple columns data layout (cf. Section 6.2.2) for various memory budgets B and the described workload. As displayed in Figure 7.15, we applied the configurations on the dataset of ten million timestamps and measured the runtime performance and memory consumption. As expected, the runtime of the benchmark queries decreases for increased memory budgets. The step-wise increases in the runtime performance are based on replacing the applied encoding of a column from run-length or LZ4 encoding to dictionary encoding, which has a faster scan performance. Based on the given memory budget, workload, and the benchmark runtimes of the isolated executed scan operations, the LP approach determines the optimal compression configuration enabling database administrators to adjust the storage configuration to application-specific requirements. Moreover, we can observe that the runtime performance stagnates after the memory budget exceeds 51.1 MB. The optimization approach uses the additional memory resources to apply dictionary encoding also on the SECOND column, but this has no considerable impact on the overall runtime performance.

Figure 7.16 provides more details about the selected compression schemes for different memory budgets. For low memory budgets, the LP model's compression scheme reduces the data footprint to 1.05 MB. This represents a reduction of over one-third compared to the scheme that uses run-length encoding for all columns, with the lowest memory consumption of the four single encoding compression schemes (1.74 MB). By considering the data characteristics in the compression scheme selection, we can reduce the memory footprint and save costs in use cases with limited main memory resources. With increasing memory budgets, we can observe that the LP model replaces run-length encoded

Budget	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND	Memory Consumption	Runtime
1.1 MB	Run-Length	Run-Length	Run-Length	Run-Length	Run-Length	LZ4	1.05 MB	7609.68 ms
6.1 MB	Run-Length	Run-Length	Run-Length	Run-Length	Run-Length	Run-Length	1.73 MB	7580.04 ms
11.1 MB	Run-Length	Run-Length	Dictionary	Run-Length	Run-Length	LZ4	11.05 MB	7205.69 ms
16.1 MB	Run-Length	Run-Length	Dictionary	Run-Length	Run-Length	Run-Length	11.73 MB	7054.54 ms
51.1 MB	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	LZ4	51.02 MB	4943.79 ms
61.1 MB	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	60.00 MB	4927.80 ms
	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	60.00 MB	4927.80 ms
	LZ4	LZ4	LZ4	LZ4	LZ4	LZ4	2.39 MB	14429.28 ms
	Run-Length	Run-Length	Run-Length	Run-Length	Run-Length	Run-Length	1.73 MB	7534.34 ms
	Unencoded	Unencoded	Unencoded	Unencoded	Unencoded	Unencoded	240.00 MB	4967.31 ms

Compression Schema

Measurements

Fig. 7.16: Evaluation of the measured memory consumption and runtime performance for different compression schemes determined by the LP model for different memory budgets (top) compared to standard compression schemes (bottom), where all six columns are encoded with the same compression technique.

columns with dictionary-encoded columns, which have a better runtime performance. We selected the memory budgets in such a way that the model can not only replace run-length encoded columns with dictionary-encoded columns in the compression configurations. For a memory budget of $B = 11.1$ MB, the consumed main memory of the SECOND column has to be reduced by applying LZ4 encoding (compared to the configuration for $B = 6.1$ MB) to enable the application of dictionary encoding on the DAY column without violating the memory budget constraint, cf. Constraint (6.8). This is done because the overall performance benefits more from faster scan operations on the DAY column than on the SECOND column. The DAY column's scan performance has the proportional highest impact for the given workload, as 80 percent of the queries access this column. Consequently, this is the first column the model chooses to apply dictionary encoding.

7.6 Discussion

Overall, the evaluation showed that fine-grained table configurations are well-suited to optimize database systems by reflecting spatio-temporal access patterns. By selecting for each segment optimized tuning options based on the data characteristics and workload, we can reduce the memory footprint or increase the system's performance.

We demonstrated that our LP models are a convenient approach for determining table configurations consisting of multiple tuning decisions for a given memory budget. In this context, we can leverage the characteristics of spatio-temporal workloads, which are dominated by spatial and temporal range queries

and trajectory-based queries for most applications. Based on the scalability and compute times of the approaches, we showed that the LP models are able to optimize the storage layouts for large datasets. Especially for low memory budgets, the LP models can significantly improve performance compared to standard compression or greedy heuristic approaches. As DRAM capacities are a not negligible cost factor for various spatio-temporal data management systems, we can reduce the operating costs by meeting performance requirements with less memory consumption. We demonstrated that SMS and ISE models are suitable approximations of the general CCD approach, which could determine comparable results for the given benchmarks. It turned out that the ISE model has an especially good tradeoff between computation time and the performance of the produced configurations. Based on these characteristics, the ISE model is well-suited for use cases that require fast decisions. For instance, scale-out scenarios with partially replicated databases. In this context, workload-driven partial replication is a mechanism to reduce the memory footprint of a replication cluster and balance the load distribution by executing queries independently on replica nodes [128, 129]. Based on the dynamic allocation of resources, the determination of table configuration optimized for a specific subset of the workload can be beneficial. However, the ISE model is not always capable of complying with the given memory budget. The SMS model has higher accuracy. Consequently, we recommend applying the SMS model in scenarios with strong sorting dependencies, strict memory budget policies, or high-performance requirements. To address the limitations of the ISE and SMS models to represent multi-attribute optimization approaches (e.g., composite indexes or space-filling curves), we introduced the general CCD model with chunk-based cost dependencies. We were able to calculate table configurations for this general model for six million configurations in a reasonable time (42 seconds). The number of configurations that have to be considered by the CCD model can be reduced by incorporating domain knowledge or using a multistage approach, where the SMS or ISE model is used to determine the best configuration without multi-attribute tuning options and the CCD model is used to incorporate further optimizations in a separate step.

Furthermore, we extended the LP models to integrate data tiering decisions. To our knowledge, this is the first approach to jointly optimize a table's sorting, indexing, tiering, and compression configuration. Based on the increased complexity of the optimization problem, we demonstrated that your approach is well-suited concerning computation time and performance. Our LP solutions allow us to reveal the main drivers of effective tunings and to infer recommendations for best practices (which columns to index, how to compress old/new chunks in case of small/large budgets, etc.). For example, we observe the following stylized patterns in optimal solutions. The sorting decision per chunk is based on the access frequency and memory saving of a segment. With increasing memory budgets, indexes are applied on segments with low selectivity queries, enabling sorting by another column. Based on the latency difference between the two storage devices, the determined models apply more heavy-weight compression techniques in main memory and compensate the increased latency with lightweight compression approaches on the slower SSD. Further, we obtain that the possibility to tune chunks and their segments (accounting for their individual specifics) on a fine-grained level is, in general, heavily exploited, cf. Figure 7.11, and does contribute to improving overall performance.

Additionally, we addressed further requirements of spatio-temporal data management systems by integrating extensions for considering reconfiguration costs and robustness. As the LP models determine an optimal solution for the given input parameters, slight variations of these parameters can lead to a large number of cost-intensive modification operations (e.g., changing the sorting order of a chunk). By incorporating the reconfiguration cost of individual tuning options in the objective of the LP models, we are able to identify and perform only minimal-invasive modifications (cf. Section 7.4.1). Consequently, we can reduce the reconfiguration costs significantly. Another aspect is that future workloads are not entirely predictable, and the performance can be negatively affected if the actual workload differs from the predicted one. In this context, we demonstrate that a worst-case optimization for various potential workload scenarios can increase the robustness of the determined table configurations and mitigate performance drops. This optimization approach often achieves not the optimal performance for a specific workload but can avoid unexpected performance decreases, which is often more important for database administrators. By integrating these extensions, the table configurations can be optimized by the LP models with regard to the requirements of real-world scenarios.

Furthermore, we demonstrated the impact of different data layouts and compression approaches for timestamps on data footprint and runtime performance. We pointed out that the presented commonly-established approaches have advantages and disadvantages for different workload characteristics. Consequently, we introduced a heuristic approach that enables the combined selection of a data layout and compression scheme for a given workload. For the joint workload-aware choice of a superior compression scheme and data layout, we evaluated our heuristic approach, which enables the balancing of memory consumption and performance requirements. In this context, we showed that the selected configurations have an equal or better runtime for the given application-specific constraints compared to standard approaches. Furthermore, we presented different heuristic enhancements for partitioned data tables to reflect time-specific access patterns by applying different configurations for various data partitions.

Besides, we describe a data layout that uses attribute decomposition to store a single timestamp in multiple columns. This optimized data layout for columnar databases is beneficial for various workloads and data compression techniques. For instance, for workloads that query specific timeframes (e.g., a day) or a specific timeframe repeatedly in a larger time range, the multi-column data layout can significantly reduce the memory traffic and, consequently, the runtime performance. Moreover, it enables the enhanced selection of a compression scheme that incorporates the specific workload and data characteristics. To determine an optimized compression scheme for a given workload and memory budget, we introduce an LP model. This model enables database administrators to restrict the used memory for timestamps and evaluate the anticipated performance decreases. The efficiency of this approach can be additionally increased by not limiting the optimization to the columns of the timestamp and considering the different temporal columns in the configuration optimization of the entire table (cf. Chapter 5). All these approaches to optimize the storing of timestamps in columnar in-memory databases should be autonomously performed by the database system. It should provide a unified interface (e.g., the timestamp data type provided by various database systems) so that the applied storage optimizations are transparent for applications. Overall, we demonstrate

that workload-aware data layout and compression scheme optimizations can significantly reduce memory consumption and improve performance.

7.7 Threats to Validity

The evaluation showed that fine-grained table configuration optimizations could significantly improve columnar in-memory databases' performance and memory footprint for spatio-temporal workloads. Nevertheless, although the model is of general nature and allows to attack complex and coupled tuning dependencies, there are the following limitations. First, we exploit the fact that in our use case, the number of attributes $|N|$ is small, keeping the LP models tractable with regard to the number of variables and constraints. In contrast to other application scenarios (e.g., enterprise systems), the storing of trajectory data requires only a manageable number of attributes. Real-world datasets of other domains can consist of thousands of attributes spread over various database tables. For instance, a single table in SAP's enterprise resource planning (ERP) system can include hundreds of attributes [35]. Especially for the CCD model, an increased problem space based on the number of columns leads to increased complexity, solver runtimes, and the limitation of different tuning options that can be evaluated (e.g., multi-column indexes). But also, for the SMS model, each additional attribute leads to a significant increase in the input parameters of the model based on the sorting dependencies. Besides the parameters of the models, a larger set of attributes also requires a larger set of benchmark queries and calibrations queries, which are necessary for the applied cost model of the presented LP approach (cf. Section 5.2.2). We discuss a strategy to address these issues in future work.

Second, we achieve to keep the model tractable by also considering a comparably small number of indexes $|I|$ (cf. single-attribute indexes in the SMS model or distinguished multi-attributes in the CCD model). In this context, we leverage that based on the relatively small number of columns, only a limited number of multi-column index candidates exist. Furthermore, different index approaches (e.g., spatial or spatio-temporal indexes) are only applicable to specific attributes or data types. Other index structures are limited to specific encoding approaches or can be excluded based on domain knowledge, which further narrows the problem space. Nevertheless, optimizing joint tuning problems for larger problems will require different, most likely heuristic techniques. In this context, the proposed model may also serve as an upper-bound reference to verify the quality of such techniques by providing optimal solutions for tractable setups of joint tuning problems.

A limitation of the used database (that we reflected in the models) is that an index on a segment has to be used for a scan operation. Modern query optimizers with advanced index usage strategies can produce query plans that do not use inefficient available index structures. We can represent this behavior by adopting (cf. Equation (5.5) and Equation (5.21)) such that for the parameter $p_{s,e,o,i}$, the minimum of the execution time with and without an index structure is used. Correspondingly, the adopted versions of Equation (5.5) would be

$$c_{m,n,e,o,i} := \sum_{\substack{q \in Q, s \in S_q: \\ n_{q,s}=n}} f_q \cdot \min(p_{q,s,e,o,i}, p_{q,s,e,o,0}) \cdot a_{m,q,s} \cdot \omega_{q,s} \cdot u_{q,s,e}. \quad (7.3)$$

By applying the cost estimation (7.3), an applied index structure has less negative performance effects for scan operations with higher selectivity values. Instead of using the index for scan operations with high selectivity values, a table scan is performed. As the modification only impacts the calculation of the cost parameter and does not further increase the number of parameters, it does not affect the solver time of the different models. In this context, the applied cost estimations must represent the capabilities of the target database system and its components (e.g., query optimizer).

Naturally, accurate cost parameter inputs are key in the model. Our cost estimations used in Equation (5.5) and Equation (5.21) provide a reasonable and viable starting point. More accurate estimations seem possible, e.g., using more sophisticated data-driven cost models, and would allow us to minimize further the gap between model-based solutions and their actual end-to-end performance. However, more complex cost models may also add more overhead. Using machine learning-based approaches to determine the segment-based costs of operations can significantly reduce the setup costs consisting of calibration and benchmarking queries (cf. Section 5.2.2). As our cost estimations are based on these parameters, we are able to consider different factors that influence the runtime performance (e.g., implementation and hardware details) but have to execute these queries on the production system or a replicated system. In contrast, other approaches, such as *Mosaic* [348], can determine tuning decisions without the setup overhead. As these approaches reduce the setup costs significantly, the usage of cost estimations should be evaluated.

Another aspect that we have to consider in this section is that we evaluated our approach only with a limited set of encoding and index options supported by the research database *Hyrise*. In this context, we demonstrate that we can restrict the problem space to the database-specific set of supported tuning options by applying our LP models. Furthermore, we can represent specific capabilities of the target database system in our models by excluding not applicable combinations of tuning options (cf. Section 5.3.6). The different models are also capable of considering further compression techniques as well as indexing strategies. The ISE and SMS models can determine configurations even for larger sets of tuning options (cf. Section 7.2.4 and Section 7.3.5). Additionally, further tuning options (e.g., compression techniques) can be added without adjusting the models. In the evaluation, we did not focus on lossy compression techniques for spatio-temporal data (e.g., line generation or trajectory simplification). These approaches minimize the number of entries per chunk by reducing the number of observed locations to represent the trace of a moving object (cf. Section 2.2.3). Moreover, these approaches can be combined with the different encodings that we analyzed in the evaluation. A decrease in the data size by reducing the number of observed locations has a positive impact on the memory consumption and the execution time of database operations. Consequently, applying the lossy compression approach with the highest compression ratio would always be beneficial to reduce the data size and further compress the data by using a lossless compression technique. Zhang et al. [379] presented a detailed evaluation of different trajectory simplification approaches. Nevertheless, it is challenging to identify an efficient table configuration for the simplified trajectory data. In

future work, we describe an advanced concept to integrate lossy compression techniques.

In the evaluation, we showed the benefit of our LP for the use case of a transportation network company. Our approach is not limited to this application scenario, as the models determine the optimal configuration for the given input parameters, which consider the application-specific data and workload characteristics. The different models are of general nature and determine the best configuration consisting of multiple tuning decisions for the defined expected workload. In this context, the performance of the computed table configurations depends on the accuracy of the workload prediction used in the optimization process. The system's performance can be negatively affected if the estimated workload strongly differs from the actual workload. Consequently, it is crucial to have accurate workload estimations. Forecasting workloads is a complex and challenging research area [138, 214]. In the thesis, we do not further focus on this problem but propose a model extension that allows optimizing the configuration for different workload scenarios, which enables database administrators to incorporate uncertainty about future workloads and perform a worst-case optimization. Additionally, we can leverage in this context that spatio-temporal applications often have repetitive similar access patterns.

Another aspect we have to discuss is the focus on table scan operations in the cost estimation (e.g., Equation (5.5)). This restriction is based on the observation that spatio-temporal range queries and trajectory-based queries often dominate spatio-temporal workloads. A valid assumption that is not only utilized in various storage optimizations for spatio-temporal data but also in general optimization approaches (e.g., Mosaic) [245, 348]. Based on the implementation of our system, the models can be extended to support other operations by providing a cost model for the specific operator that enables the estimation of the operator's costs on a segment level. The integration of cost models for further operators only impacts the calculation of the cost parameter and has consequently no impact on the solve time of the different models.

We used the runtime of the solver to solve the different models to present the scalability of our approach and demonstrate the complexity differences between the different models. The solver time strongly depends on the used solver and parameters (e.g., number of threads or available memory resources). Furthermore, we did not use further optimizations as optimality gaps or time limits. By applying these relaxations, we can reduce the solve time by accepting near-optimal solutions or the best result the solver was able to determine within the given time limit.

To highlight the impact of the different tuning options, we executed the benchmarks single-threaded. In general, database operations are executed multi-threaded. Measuring and considering the effects of the parallel execution of database operations is hard and significantly increases complexity. We reduce the complexity based on the assumption that we optimize the performance of the individual scan operations for each segment. In this context, we also abstract from caching effects, system utilization, and the execution order of queries. In general, all these parameters cannot be influenced or predicted during the optimization process.

7.8 Summary

In this chapter, we analyzed the accuracy, performance, and scalability of the models introduced in Chapter 5 based on the real-world application example of a TNC. First, we evaluated the base models (cf. Section 5.3) that jointly optimize the sorting, indexing, and compression configuration for a given workload and memory budget. We demonstrated that fine-grained table configurations are a practical approach to optimizing columnar in-memory databases for spatio-temporal data. In contrast to standard configurations, the runtime and memory consumption can be significantly reduced (cf. Section 7.2.1). Compared to the standard configuration that applies dictionary encoding to all segments, the LP models achieve similar performance with up to 58% less required memory resources. Moreover, we showed that our LP models could determine efficient configurations that outperform greedy heuristics and achieves up to 82% increased performance by equal memory size or reduces the memory footprint by up to 56% with comparable performance. Second, we evaluated our enhanced models that extend the LP models by including data tiering decisions (cf. Section 5.4). In this context, we demonstrated that our models are superior to existing rule-based heuristics (cf. Section 7.3.2). Our models achieve up to 90% increased runtime performance for a given memory budget; a comparable runtime is obtained with up to 70% less required memory. Moreover, we showed that our LP models are superior to consecutive optimization approaches (cf. Section 7.3.3). In this context, the LP approaches could increase the performance by 15% for less restrictive memory budgets and achieve significantly faster runtimes (up to 6.8 times) for more restricted memory budgets.

Further, we discussed the scalability of our base models (cf. Section 7.2.4) and enhanced models with tiering decisions (cf. Section 7.3.5) to demonstrate the scalability of the LP approach concerning the number of (i) chunks, (ii) tuning options, (iii) storage devices, and (iv) query templates. We introduced three LP models (cf. CCD, SMS, and ISE) addressing cost dependencies at different levels of accuracy while allowing for trading their solve time. The SMS model reliably finds optimized tuning configurations if sorting dependencies are present. If those are not strong or shorter runtimes are in focus, we identified that the relaxed ISE model is a suitable scalable alternative with competitive results. Third, we demonstrated that the proposed model extensions, which include reconfiguration costs and robustness, are suitable for reducing modification costs and avoiding significant performance drops (cf. Section 7.4). Furthermore, we showed that workload-driven combined data layout and compression schema optimizations could significantly impact the memory consumption and performance of timestamps. Finally, we briefly discussed the different measurements (cf. Section 7.6) and threats to validity (cf. Section 7.7).

8

Conclusion

In this chapter, we conclude this thesis with suggestions for future research directions (Section 8.1) and a summary of how the presented contributions address our research questions (Section 8.2).

8.1 Future Work

This section introduces approaches for future work to improve the integration of lossy compression techniques (Section 8.1.1) and to adjust the presented workload-driven configuration optimization process for other application domains (Section 8.1.2). For both approaches, we summarize the problem domain, describe how the current implementation addresses the problem, and present enhanced strategies for future work.

8.1.1 Improving the Integration of Lossy Compression Techniques

To reduce resource consumption (e.g., memory usage) for storing and processing large amounts of spatio-temporal data, lossy compression techniques are applied in various use cases [239]. In this context, trajectory simplification is the most common lossy compression method. As described in Section 2.2.3, the basic idea of this method is to remove sample points considered as less important based on a distance measure such that the original trajectory can be approximated by a series of successive line segments constructed from the remaining sample points [379]. In general, the applied distance measures are error-bound or size-bound. Error-bound metrics remove sample points based on a distance function (e.g., angle distance, Euclidean distance, or perpendicular distance) and a defined maximum error [188]. The advantage of this approach is that the maximum error can specify the approximation of the resulting trajectory. A disadvantage is that the expected compression ratio is hard to predict, as it depends on the dataset and the defined maximum error. The size-bound approaches remove the sample points with the lowest deviation compared to the original trajectory until a given compression ratio is reached. The same distance functions as for the error-bound approaches can be used to determine the deviation. In contrast to the error-bound approach, the resulting data footprint is predictable, but the approximated trajectory can be less accurate. Hence, it remains challenging to determine a proper algorithm in a concrete application scenario [379].

In our joint workload-driven optimization approach (cf. Chapter 5), we are able to consider trajectory simplification techniques. By defining a maximum error value or targeted compression ratio for the entire table or individual chunks, the data footprint can be reduced for different data partitions. As these compression methods reduce the number of sample points in different chunks, they also decrease the data footprint and runtime of database operations. Additionally, further compression techniques can be applied (e.g., dictionary encoding) on top of the simplified trajectories. The linear programming (LP) models to optimize a table configuration would select a lossy compression approach whenever possible based on the reduced data footprint and runtime advantages.

To avoid unevenly distributed data partitions, which can negatively impact the system's performance, we must consider different reorganization strategies to integrate lossy compression techniques more efficiently. For size-bound approaches, we can choose a suitable compression ratio for a set of chunks so that entire chunks can be excluded from the data. For error-bound approaches, this is not possible, as we cannot configure the resulting number of sample points. Consequently, we have to reorganize the chunks (e.g., merge chunks) if there are too many chunks containing significantly fewer entries.

Another approach to integrating the basic idea of trajectory simplification into the chunk concept is to cluster data based on different granularity values. In this context, the granularity of a trajectory is defined by a maximum error value (error-bound approaches) or a compression ratio (size-bound approaches). Based on the granularity, we partition all entries with the same granularity in a specific set of chunks and all other entries with a higher granularity in a separate set of chunks. Correspondingly, we have to store the granularity for each chunk. By distributing the data on different partitions, we can efficiently prune and minimize the number of data partitions that have to be accessed by queries based on the requested granularity [202]. We can prune all chunks with a higher granularity value as defined in the request during query processing. This approach would be beneficial if we have various applications with different requirements concerning data resolution. Moreover, different tuning approaches can be applied based on the specific access characteristics for different granularity values.

8.1.2 Adjustments of the Fine-Grained Optimizations Concept for Further Application Scenarios

Besides spatio-temporal data, workload-driven optimizations of table configurations are also suitable for improving performance and reducing operating costs in further applications (e.g., business data). For in-memory HTAP databases that are capable of processing both transactional (OLTP) and analytical (OLAP) workloads, there is an economic desire to leverage the available memory resources efficiently [186]. The ongoing shift towards vendor-hosted cloud deployments increases the significance as smaller memory footprints enable the placement of more tenants on the same server. Consequently, the operating costs can be significantly reduced by applying various table configuration optimizations that reduce the memory footprint. Therefore, the challenge is guaranteeing fast response times (often defined by service level agreements) and optimizing memory consumption. To improve the configurations for various application sce-

narios, we have to scale the presented LP approaches in two dimensions (i) the number of tables and columns and (ii) the number of supported query types.

Especially for enterprise resource planning systems with large parts of the data never or infrequently accessed, workload optimizations can improve resource utilization [210, 264]. The evaluation of our LP approach demonstrated that the different models could determine configurations for larger datasets. Concerning the number of columns, the ISE model showed that it is able to determine table configurations for a larger set N. In contrast, for the SMS model and CCD model, we observed a significant increase in the solver time for larger numbers of columns. Consequently, we have to use the ISE model or reduce the number of attributes for tables of business applications with several hundred columns. In this context, we can leverage the fact that a large part of the columns in such a system is used for configuration purposes and, consequently, never accessed in specific customer workloads [264]. In this case, we do not need to consider all columns in the model and can apply a pre-processing step in which we choose the configuration with the lowest memory consumption on all columns and data partitions that are not accessed by query of the workload. Based on this strategy, we are able to reduce the number of columns significantly. Additionally, we can reduce the number of data partitions by summarizing data chunks with equal access and data characteristics (e.g., fiscal quarter) and apply for each section the determined configuration. Furthermore, the LP models can be used as a baseline for developing heuristic approaches.

The second factor that we have to address is the number of query types. An advantage of spatio-temporal applications is that there is only a limited set of query types. In modern business applications, various query types define the workload. Based on the number of different query types, benchmark queries are no longer suitable for determining the impact of different configurations. As mentioned in Section 5.2.2, there are different approaches to predict the runtime of different operators [5, 32, 219]. In this context, the accuracy of the different approaches and the impact on the determined configurations have to be further evaluated. The presented approach can improve performance and resource consumption in other domains besides spatio-temporal data by further investigating these research areas.

8.2 Summary

The wide adaption of location-acquisition technologies led to large amounts of spatio-temporal data. Analyzing the trajectories of moving objects based on detailed positional information creates opportunities for applications that can improve business decisions and processes in a broad spectrum of industries. In this context, we analyzed the processes of transportation network companies (TNC), which track the positions of their drivers for route planning or demand predictions [96]. Using the collected spatio-temporal data of a TNC, we developed algorithms to optimize the order dispatching of incoming passenger requests. Such process optimizations and advanced decisions support systems based on spatio-temporal data mining represent a crucial business advantage over competitors. However, at the same time, the storing and processing of spatio-temporal data is a significant cost factor for companies due to the large

amounts of continuously accumulated data and interactive performance requirements.

In this thesis, we explored the potential of columnar in-memory databases to store and process spatio-temporal data. We demonstrated the impact of different database optimizations to improve query response times or reduce the operating costs of such systems. Moreover, we presented the concept of fine-grained database optimizations to reflect spatio-temporal access patterns in the storage layer and optimize partitions of the data, particularly for the application-specific workload and data characteristics of spatio-temporal systems. For database administrators, the implications of different tuning options are hard to estimate. Consequently, various tuning approaches are only applied conservatively to avoid performance decreases. To address the problem, we presented two novel methods for the workload-driven joint optimization of different configuration options for spatio-temporal data mining applications. Our developed approaches allow us to answer our two research questions as follows:

- *How can we improve business processes in specific application scenarios by spatio-temporal data mining?*

  To demonstrate the impact of spatio-temporal data mining applications on business decisions, we investigated the use case of a TNC. For peer-to-peer ride-hailing providers, it is a crucial competitive advantage to dispatch passenger requests cost-efficiently and to communicate accurate waiting times. In this context, we analyzed the limitations of established dispatching strategies. In particular, we identified inaccurate positional information as one aspect of drivers' late arrivals at pickup locations. Due to technical restrictions and outdated data (e.g., noise, low sample rates, or continuous movement of drivers), state-of-the-art dispatch algorithms based on the last observed locations of drivers regularly suffer from inefficient driver assignments with critical delays. We introduced an approach that addresses the problem of outdated locations by predicting a probability distribution for a driver's actual location. The proposed algorithm uses patterns observed in past trajectory data to determine sets of potential locations and their corresponding probabilities. Furthermore, we suggested different dispatch strategies that enable quantifying and considering the risk of critical delays. Based on a real-world dataset of a TNC, we demonstrated the applicability and accuracy of our prediction approach using numerical experiments. Furthermore, we highlighted the capabilities of our location prediction approach to (i) minimize the customers' cancellation rates by avoiding critical delays, (ii) estimate waiting times with higher confidence, and (iii) enable risk-averse dispatching strategies.

- *How can we determine cost and performance-balancing table configuration optimizations consisting of fine-grained mutually dependent tuning decisions for the specific data properties and workloads of spatio-temporal applications?*

  To efficiently utilize the available memory capacities, modern database systems apply various optimizations (e.g., data compression or secondary indexes) to reduce the memory footprint or increase performance. However, the selection of cost and performance-balancing configurations is challenging

due to the vast number of possible setups consisting of mutually dependent individual decisions. We introduced a linear programming (LP) approach to determine fine-grained configuration decisions for spatio-temporal workloads. The proposed approach jointly optimizes the compression, sorting, indexing, and data tiering configuration to maximize the performance by minimizing the runtime for a given workload, memory budget, and data characteristics. We presented three LP-based models (CCD, SMS, and ISE) addressing cost dependencies at different levels of accuracy and evaluated the performance of the determined table configurations as well as the scalability. To yield maintainable and robust configurations, we extended the LP-based models to incorporate reconfiguration costs and multiple potential workload scenarios. We demonstrated on the TNC dataset that our models allow us to significantly reduce the memory footprint (up to 71% by equal performance) or increase the performance (up to 90% by equal memory size) compared to established rule-based heuristics.
Furthermore, we analyzed different storage concepts for timestamps in column-oriented databases. As various standard compression approaches for a columnar storage layout are optimized for contradicting data characteristics (e.g., low number of distinct values, sequences of equal values), the memory-efficient storing of timestamps is challenging. We presented and compared different approaches to storing timestamps in columnar in-memory databases. Additionally, we proposed a data layout that uses an attribute decomposition approach to store the temporal information in multiple columns. We showed that this multi column data layout is beneficial for range queries with standard access ranges (e.g., month or year). Moreover, we introduced an LP-based model to determine an optimized compression scheme for this storage concept. Based on the TNC dataset, we evaluated the memory consumption and performance of different data layouts and compression techniques. The results showed that there are significant differences between the various configurations for specific access patterns. For this reason, we developed a heuristic approach to select an optimized data layout and compression scheme for a given workload.

Based on the contributions presented in this thesis, we demonstrated how business decisions can be improved by trajectory data mining. Furthermore, we introduced two approaches to optimizing data management specifically for such applications. By applying joint workload-driven database optimizations, we enable columnar in-memory databases to store and process spatio-temporal data more memory-efficient.

List of Figures

List of Tables

Acronyms

ASA Adaptive Storage Advisor.

CCD LP model with chunk-based configuration dependencies.

DBMS Database Management System.
DRAM Dynamic Random Access Memory.
DTA Database Engine Tuning Advisor.

GFS Google File System.
GH Greedy Heuristic.
GIS Geographical Information System.
GPS Global Positioning System.
GSM Global System for Mobile Communication.

HDD Hard Disk Drive.
HDFS Hadoop Distributed File System.
HMM Hidden Markov Model.
HTAP Hybrid Transactional and Analytics Processing.

ISE LP model with independent segment effects.

KPI Key Performance Indicator.

LP Linear Programming.

NSE Native Storage Extension of SAP HANA.
NVM Non-Volatile Memory.

OLAP Online Analytical Processing.
OLTP Online Transcation Processing.

RDD Resilient Distributed Datasets.
RFID Radio-Frequency Identification.

SMS Segment-based LP model with sorting dependencies.
SSD Solid-State Drive.

TCO Total Cost of Ownership.
TNC Transportation Network Company.

A

Appendix

A.1 Permission of Reuse of Publications

Reuse of Material Published by ACM
Authors can reuse any portion of their own work in a new work of their own (and no fee is expected) as long as a citation and DOI pointer to the Version of Record in the ACM Digital Library are included. Contributing complete papers to any edited collection of reprints for which the author is not the editor, requires permission and usually a republication fee. Authors can include partial or complete papers of their own (and no fee is expected) in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included. Authors can use any portion of their own work in presentations and in the classroom (and no fee is expected).

Reuse of Material Published by IEEE
In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Hasso Plattner Institute's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/right_link.html` to learn how to obtain a License from RightsLink.

Reuse of Material Published by Springer
Authors have the right to reuse their articles Version of Record, in whole or in part, in their own thesis. Additionally, they may reproduce and make available their thesis, including Springer Nature content, as required by their awarding academic institution. Authors must properly cite the published article in their thesis according to current citation standards.

A.2 Benchmark Workloads

Overview of the query templates for the benchmark workloads A and B. For each query template, the selectivity of the corresponding filter values is displayed in curly brackets. For instance, a selectivity value of 0.4 indicates that the corresponding scan operation returns 40% of the entries if executed on the data of the entire table. The specific values depend on the data characteristics of the used dataset. Additionally, the frequency of the query template and the skipped chunks are displayed. Based on statistics maintained for each chunk (cf. Section 2.3.2), several chunks can be pruned during query execution for a specific query. These dataset-specific values are exemplary and vary for different chunk sizes.

ID	Query Template	Frequency	Skipped Chunks
q_{A0}	`SELECT * FROM Table WHERE ("timestamp" <= {value selectivity: 0.4}) AND ("status" <= 0.7)`	30 %	4, 5, 6, 7, 8, 9
q_{A1}	`SELECT * FROM Table WHERE ("timestamp" <= {0.2}) AND ("latitude" BETWEEN {0.05}) AND ("longitude" BETWEEN {0.05})`	50 %	2, 3, 4, 5, 6, 7, 8, 9
q_{A2}	`SELECT * FROM Table WHERE ("driver_id" <= {0.0001}) AND ("timestamp" <= {0.4})`	16 %	0, 3, 4, 5, 6, 7, 8, 9
q_{A3}	`SELECT * FROM Table WHERE ("timestamp" BETWEEN {0.8})`	2.5 %	9
q_{A4}	`SELECT * FROM Table WHERE ("latitude" BETWEEN {0.3}) AND ("longitude" BETWEEN {0.3})`	0.5 %	–
q_{A5}	`SELECT * FROM Table WHERE ("driver_id" <= {0.5}) AND ("latitude" BETWEEN {0.2}) AND ("longitude" BETWEEN {0.2} AND ("timestamp" BETWEEN {0.4}) AND ("status" <= 0.7)`	1 %	0, 1, 7, 8, 9

Table A.1: Overview of the query templates included in the query template set Q_A representing the benchmark workload A. We determined the skipped chunks for the dataset with ten million observed locations (cf. Section 7.1.1).

ID	Query Template	Frequency	Skipped Chunks	
			10 M	400 M
q_{B0}	`SELECT * FROM Table WHERE ("driver_id" <= {0.0001}) AND ("status" <= 0.7)`	15 %	0, 3, 6, 7, 8	0, 1
q_{B1}	`SELECT * FROM Table WHERE ("timestamp" BETWEEN {0.2}) AND ("latitude" BETWEEN {0.1}) AND ("longitude" BETWEEN {0.1}) AND ("status" <= {0.7})`	15 %	0, 1, 2, 6, 7, 8, 9	0, 1, 2, 6, 7, 8, 9
q_{B2}	`SELECT * FROM Table WHERE ("driver_id" <= {0.01}) AND ("latitude" BETWEEN {0.3}) AND ("longitude" BETWEEN {0.3})`	10 %	–	–
q_{B3}	`SELECT * FROM Table WHERE ("timestamp" <= {0.05}) AND ("latitude" BETWEEN {0.7}) AND ("longitude" BETWEEN {0.7})`	25 %	1, 2, 3, 4, 5, 6, 7, 8, 9	1, 2, 3, 4, 5, 6, 7, 8, 9
q_{B4}	`SELECT * FROM Table WHERE ("driver_id" <= {0.01}) AND ("timestamp" <= {0.4})`	15 %	4, 5, 6, 7, 8, 9	4, 5, 6, 7, 8, 9
q_{B5}	`SELECT * FROM Table WHERE ("latitude" BETWEEN {0.01}) AND ("longitude" BETWEEN {0.01}) AND ("timestamp" BETWEEN {0.5})`	20 %	0, 1, 8, 9	0, 1, 8, 9

Table A.2: Overview of the query templates included in the query template set Q_B representing the benchmark workload B. We determined the skipped chunks for the dataset with ten million (10 M) and 400 million (400 M) observed locations (cf. Section 7.1.1). The data of both datasets are partitioned into ten chunks.

A.3 Publications

Our work has been presented at renowned international conferences and workshops as well as published in journals. A list of all publications is provided in this section.

- Richly, K.; Schlosser, R.; Boissier, M.: *Budget-Conscious Fine-Grained Configuration Optimization for Spatio-Temporal Applications*. In *Proceedings of the VLDB Endowment* 15(13), 2022: pp. 4079 – 4092
- Richly, K.; Schlosser, R.; Brauer, J.: *Enabling Risk-averse Dispatch Processes for Transportation Network Companies by Probabilistic Location Prediction*. In *Operations Research and Enterprise Systems*. Springer, 2022, Volume 1623 by *Communications in Computer and Information Science*, pp. 21–42
- Richly, K.; Schlosser, R.; Boissier, M.: *Joint Index, Sorting, and Compression Optimization for Memory-Efficient Spatio-Temporal Data Management*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2021, pp. 1901–1906
- Richly, K.: *Memory-Efficient Storing of Timestamps for Spatio-Temporal Data Management in Columnar In-Memory Databases*. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*. 2021, pp. 542–557
- Richly, K.; Schlosser, R.; Brauer, J.; Plattner, H.: *A Probabilistic Location Prediction Approach to Optimize Dispatch Processes in the Ride-Hailing Industry*. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS)*. 2021, pp. 1830–1840
- Richly, K.; Brauer, J.; Schlosser, R.: *Predicting Location Probabilities of Drivers to Improve Dispatch Decisions of Transportation Network Companies based on Trajectory Data*. In *Proceedings of the International Conference on Operations Research and Enterprise Systems (ICORES)*. 2020, pp. 47–58
- Richly, K.: *Optimized Spatio-Temporal Data Structures for Hybrid Transactional and Analytical Workloads on Columnar In-Memory Databases*. In *Proceedings of the VLDB PhD Workshop*. 2019, pp. 1–4
- Richly, K.: *A Survey on Trajectory Data Management for Hybrid Transactional and Analytical Workloads*. In *Proceedings of the IEEE International Conference on Big Data (BigData)*. 2018, pp. 562–569
- Richly, K.: *Leveraging Spatio-Temporal Soccer Data to Define a Graphical Query Language for Game Recordings*. In *Proceedings of the IEEE International Conference on Big Data (BigData)*. 2018, pp. 3456–3463
- Richly, K.; Moritz, F.; Schwarz, C.: *Utilizing Artificial Neural Networks to Detect Compound Events in Spatio-Temporal Soccer Data*. In *Proceedings of the ACM SIGKDD Workshop on Mining and Learning from Time Series (MiLeTs)*. 2017, pp. 13–17
- Richly, K.; Teusner, R.: *Where is the Money Made? An Interactive Visualization of Profitable Areas in New York City*. In *Proceedings of the International Conference on IoT in Urban Space (Urb-IoT)*. ACM, 2016, pp. 43–46
- Richly, K.; Bothe, M.; Rohloff, T.; Schwarz, C.: *Recognizing Compound Events in Spatio-Temporal Football Data*. In *Proceedings of the In-*

ternational Conference on Internet of Things and Big Data (IoTBD). 2016, pp. 27–35

- Richly, K.; Teusner, R.; Immer, A.; Windheuser, F.; Wolf, L.: *Optimizing Routes of Public Transportation Systems by Analyzing the Data of Taxi Rides*. In *Proceedings of the International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*. 2015, pp. 70–76
- Richly, K.; Lorenz, M.; Oergel, S.: *S4J – Integrating SQL into Java at Compiler-Level*. In *Proceedings of the International Conference on Information and Software Technologies (ICIST)*. Springer, 2016, Volume 639 by *Communications in Computer and Information Science*, pp. 300–315
- Schlosser, R.; Richly, K.: *Dynamic Pricing under Competition with Data-Driven Price Anticipations and Endogenous Reference Price Effects*. In *Journal of Revenue and Pricing Management* 18(6), 2019: pp. 451–464
- Schlosser, R.; Richly, K.: *Dynamic Pricing Competition with Unobservable Inventory Levels: A Hidden Markov Model Approach*. In *Operations Research and Enterprise Systems*. Springer, 2018, Volume 966 by *Communications in Computer and Information Science*, pp. 15–36
- Schlosser, R.; Richly, K.: *Dynamic Pricing Strategies in a Finite Horizon Duopoly with Partial Information*. In *Proceedings of the International Conference on Operations Research and Enterprise Systems (ICORES)*. 2018, pp. 21–30
- Matthies, C.; Kowark, T.; Richly, K.; Uflacker, M.; Plattner, H.: *ScrumLint: Identifying Violations of Agile Practices Using Development Artifacts*. In *Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 2016, pp. 40–43
- Kowark, T.; Richly, K.; Uflacker, M.; Plattner, H.: *Incremental, Per-Query Ontology Matching with RepMine*. In *Proceedings of the International Conference Companion on World Wide Web (WWW)*. 2016, pp. 215–218
- Matthies, C.; Kowark, T.; Richly, K.; Uflacker, M.; Plattner, H.: *How Surveys, Tutors, and Software Help to Assess Scrum Adoption in a Classroom Software Engineering Project*. In *Proceedings of the ACM International Conference on Software Engineering (ICSE)*. 2016, pp. 313–322
- Teusner, R.; Richly, K.; Staubitz, T.; Renz, J.: *Enhancing Content between Iterations of a MOOC–Effects on Key Metrics*. In *European MOOCs Stakeholder Summit* 2015: pp. 147–156
- Kowark, T.; Teusner, R.; Richly, K.; Plattner, H.: *RepMine: A System for Transferrable Analyses of Collaboration Activities in Software Engineering*. In *IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*. 2015, pp. 78–81
- Bross, J.; Richly, K.; Kohnen, M.; Meinel, C.: *Identifying the Top-Dogs of the Blogosphere*. In *Social network analysis and mining* 2(1), 2012: pp. 53–67
- Bross, J.; Richly, K.; Schilf, P.; Meinel, C.: *Social Physics of the Blogosphere - Capturing, Analyzing and Presenting Interdependencies within a Single Framework*. In *From Sociology to Computing in Social Networks - Theory, Foundations and Applications*, Springer, Volume 1 by *Lecture Notes in Social Networks*. 2010, pp. 301–321

References

[1] Abadi, D. J.; Madden, S.; Ferreira, M.: *Integrating compression and execution in column-oriented database systems*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2006, pp. 671–682

[2] Abadi, D. J.; Madden, S. R.; Hachem, N.: *Column-stores vs. row-stores: how different are they really?*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2008, pp. 967–980

[3] Abdelguerfi, M.; Givaudan, J.; Shaw, K.; Ladner, R.: *The 2-3TR-tree, a trajectory-oriented index structure for fully evolving valid-time spatio-temporal datasets*. In *Proceedings of the ACM GIS International Symposium on Advances in Geographic Information Systems*. 2002, pp. 29–34

[4] Abebe, M.; Lazu, H.; Daudjee, K.: *Proteus: Autonomous Adaptive Storage for Mixed Workloads*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2022, pp. 700–714

[5] Abebe, M.; Lazu, H.; Daudjee, K.: *Tiresias: Enabling Predictive Autonomous Storage and Indexing*. In *Proceedings of the VLDB Endowment* 15(11), 2022: pp. 3126–3136

[6] Agarwal, N.; Wenisch, T. F.: *Thermostat: Application-transparent page management for two-tiered main memory*. In *Proceedings of the ACM ASPLOS International Conference on Architectural Support for Programming Languages and Operating Systems*. 2017, pp. 631–644

[7] Agatz, N. A. H.; Erera, A. L.; Savelsbergh, M. W. P.; Wang, X.: *Optimization for dynamic ride-sharing: A review*. In *European Journal of Operational Research* 223(2), 2012: pp. 295–303

[8] Agrawal, S.; Bruno, N.; Chaudhuri, S.; Narasayya, V. R.: *AutoAdmin: Self-Tuning Database Systems Technology*. In *IEEE Data Eng. Bull.* 29(3), 2006: pp. 7–15

[9] Agrawal, S.; Chaudhuri, S.; Kollar, L.; Marathe, A.; Narasayya, V.; Syamala, M.: *Database Tuning Advisor for Microsoft SQL Server 2005*. In *Proceedings of the VLDB Endowment*. 2004, pp. 1110–1121

[10] Agrawal, S.; Narasayya, V.; Yang, B.: *Integrating vertical and horizontal partitioning into automated physical database design*. In *Proceedings*

of the ACM SIGMOD International Conference on Management of Data. 2004, pp. 359–370

[11] Aken, D. V.; Pavlo, A.; Gordon, G. J.; Zhang, B.: *Automatic Database Management System Tuning Through Large-scale Machine Learning*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2017, pp. 1009–1024

[12] Al-Ammar, M. A.; Alhadhrami, S.; Al-Salman, A.; Alarifi, A.; Al-Khalifa, H. S.; Alnafessah, A.; Alsaleh, M.: *Comparative survey of indoor positioning technologies, techniques, and algorithms*. In *Proceedings of the IEEE International Conference on Cyberworlds*. 2014, pp. 245–252

[13] Almeida, A. C.; Baião, F.; Lifschitz, S.; Schwabe, D.; Campos, M. L. M.: *Tun-O_{CM}: A model-driven approach to support database tuning decision making*. In *Decision Support Systems* 145, 2021: p. 113538

[14] Alsahfi, T.; Almotairi, M.; Elmasri, R.: *A survey on trajectory data warehouse*. In *Spatial Information Research* 28(1), 2020: pp. 53–66

[15] Andrei, M.; Lemke, C.; Radestock, G.; Schulze, R.; Thiel, C.; Blanco, R.; Meghlan, A.; Sharique, M.; Seifert, S.; Vishnoi, S.; et al.: *SAP HANA adoption of non-volatile memory*. In *Proceedings of the VLDB Endowment* 10(12), 2017: pp. 1754–1765

[16] Andrienko, N.; Andrienko, G.; Fuchs, G.; Jankowski, P.: *Scalable and privacy-respectful interactive discovery of place semantics from human mobility traces*. In *Information Visualization* 15(2), 2016: pp. 117–153

[17] Appuswamy, R.; Graefe, G.; Borovica-Gajic, R.; Ailamaki, A.: *The five-minute rule 30 years later and its impact on the storage hierarchy*. In *Communications of the ACM* 62(11), 2019: pp. 114–120

[18] Aref, W. G.; Samet, H.: *Extending a DBMS with Spatial Operations*. In *Proceedings of the International Symposium on Advances in Spatial Databases (SSD)*

[19] Arulraj, J.; Pavlo, A.; Dulloor, S. R.: *Let's talk about storage & recovery methods for non-volatile memory database systems*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2015, pp. 707–722

[20] Arulraj, J.; Pavlo, A.; Menon, P.: *Bridging the archipelago between row-stores and column-stores for hybrid workloads*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2016, pp. 583–598

[21] Athanassoulis, M.; Bøgh, K. S.; Idreos, S.: *Optimal Column Layout for Hybrid Workloads*. In *Proceedings of the VLDB Endowment* 12(13), 2019: pp. 2393–2407

[22] Athanassoulis, M.; Chen, S.; Ailamaki, A.; Gibbons, P. B.; Stoica, R.: *MaSM: efficient online updates in data warehouses*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2011, pp. 865–876

[23] Atluri, G.; Karpatne, A.; Kumar, V.: *Spatio-Temporal Data Mining: A Survey of Problems and Methods*. In *ACM Computing Surveys* 51(4), 2018: pp. 1–41

[24] Ayhan, S.; Costas, P.; Samet, H.: *Predicting estimated time of arrival for commercial flights*. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 33–42

[25] Baig, F.; Teng, D.; Kong, J.; Wang, F.: *SPEAR: Dynamic Spatio-Temporal Query Processing over High Velocity Data Streams*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. 2021, pp. 2279–2284

[26] Bajaj, R.; Ranaweera, S. L.; Agrawal, D. P.: *GPS: location-tracking technology*. In *Computer* 35(4), 2002: pp. 92–94

[27] Basik, F.; Gedik, B.; Etemoglu, Ç.; Ferhatosmanoglu, H.: *Spatio-Temporal Linkage over Location-Enhanced Services*. In *IEEE Transactions on Mobile Computing* 17(2), 2018: pp. 447–460

[28] Beckmann, N.; Kriegel, H.; Schneider, R.; Seeger, B.: *The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1990, pp. 322–331

[29] Ben Ticha, H.; Absi, N.; Feillet, D.; Quilliot, A.: *Vehicle routing problems with road-network information: State of the art*. In *Networks* 72(3), 2018: pp. 393–406

[30] Bisschop, J.: *AIMMS-Optimization Modeling*, 2006

[31] Boehm, A.: *In-memory for the masses: enabling cost-efficient deployments of in-memory data management platforms for business applications*. In *Proceedings of the VLDB Endowment* 12(12), 2019: pp. 2273–2275

[32] Boissier, M.: *Robust and budget-constrained encoding configurations for in-memory database systems*. In *Proceedings of the VLDB Endowment* 15(4), 2021: pp. 780–793

[33] Boissier, M.; Jendruk, M.: *Workload-Driven and Robust Selection of Compression Schemes for Column Stores*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2019, pp. 674–677

[34] Boissier, M.; Kurzynski, D.: *Workload-Driven Horizontal Partitioning and Partition Pruning for Large HTAP Systems*. In *Proceedings of the IEEE International Conference on Data Engineering Workshops (ICDEW)*. 2018, pp. 116–121

[35] Boissier, M.; Schlosser, R.; Uflacker, M.: *Hybrid Data Layouts for Tiered HTAP Databases with Pareto-Optimal Data Placements*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. 2018, pp. 209–220

[36] Boncz, P. A.; Manegold, S.; Kersten, M. L.; et al.: *Database architecture optimized for the new bottleneck: Memory access*. In *Proceedings of the VLDB International Conference on Very Large Data Bases*. 1999, pp. 54–65

[37] Brendle, M.; Weber, N.; Valiyev, M.; May, N.; Schulze, R.; Böhm, A.; Moerkotte, G.; Grossniklaus, M.: *SAHARA: Memory Footprint Reduction of Cloud Databases with Automated Table Partitioning*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2022, pp. 1–13

[38] Brendle, M.; Weber, N.; Valiyev, M.; May, N.; Schulze, R.; Böhm, A.; Moerkotte, G.; Grossniklaus, M.: *Precise, Compact, and Fast Data Access Counters for Automated Physical Database Design*. In *Datenbanksysteme für Business, Technologie und Web (BTW)* 2021: pp. 79–100

[39] Bross, J.; Richly, K.; Kohnen, M.; Meinel, C.: *Identifying the Top-Dogs of the Blogosphere*. In *Social network analysis and mining* 2(1), 2012: pp. 53–67

[40] Bross, J.; Richly, K.; Schilf, P.; Meinel, C.: *Social Physics of the Blogosphere - Capturing, Analyzing and Presenting Interdependencies within a Single Framework*. In *From Sociology to Computing in Social Networks - Theory, Foundations and Applications*, Springer, Volume 1 by *Lecture Notes in Social Networks*. 2010, pp. 301–321

[41] Bussieck, M. R.; Meeraus, A.: *General algebraic modeling system (GAMS)*. In *Modeling languages in mathematical optimization*, Springer. 2004, pp. 137–157

[42] Butler, H.; Daly, M.; Doyle, A.; Gillies, S.; Schaub, T.; Schmidt, C.: *The GeoJSON format specification*. In *Rapport technique* 67, 2008

[43] Bynum, M. L.; Hackebeil, G. A.; Hart, W. E.; Laird, C. D.; Nicholson, B. L.; Siirola, J. D.; Watson, J.-P.; Woodruff, D. L.: *Pyomo–Optimization Modeling in Python*, Volume 67. Springer Science & Business Media, 3. Edition, 2021

[44] Caprara, A.; Fischetti, M.; Maio, D.: *Exact and approximate algorithms for the index selection problem in physical database design*. In *IEEE Transactions on Knowledge and Data Engineering* 7(6), 1995: pp. 955–967

[45] Cen, L.; Kipf, A.; Marcus, R.; Kraska, T.: *LEA: A Learned Encoding Advisor for Column Stores*. In *Proceedings of the ACM aiDM Workshop in Exploiting AI Techniques for Data Management*. 2021, p. 3235

[46] Chae, J.; Thom, D.; Bosch, H.; Jang, Y.; Maciejewski, R.; Ebert, D. S.; Ertl, T.: *Spatiotemporal social media analytics for abnormal event detection and examination using seasonal-trend decomposition*. In *Proceedings of the IEEE VAST Conference on Visual Analytics Science and Technology*. 2012, pp. 143–152

[47] Chakka, V. P.; Everspaugh, A.; Patel, J. M.: *Indexing Large Trajectory Data Sets With SETI*. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. 2003

[48] Chasseur, C.; Patel, J. M.: *Design and evaluation of storage organizations for read-optimized main memory databases*. In *Proceedings of the VLDB Endowment* 6(13), 2013: pp. 1474–1485

[49] Chaudhuri, S.; Dageville, B.; Lohman, G.: *Self-managing technology in database management systems*. In *Proceedings of the VLDB International Conference on Very large Data Bases*. 2004

[50] Chaudhuri, S.; Narasayya, V.: *Self-tuning database systems: a decade of progress*. In *Proceedings of the VLDB International Conference on Very Large Data*. 2007, pp. 3–14

[51] Chaudhuri, S.; Narasayya, V. R.: *AutoAdmin 'What-if' Index Analysis Utility*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1998, pp. 367–378

[52] Chaudhuri, S.; Weikum, G.: *Self-Management Technology in Databases*. In *Encyclopedia of Database Systems*, Springer. 2009, pp. 2550–2555

[53] Chen, C.; Ding, Y.; Xie, X.; Zhang, S.; Wang, Z.; Feng, L.: *TrajCompressor: An online map-matching-based trajectory compression frame-*

work leveraging vehicle heading direction and change. In *IEEE Transactions on Intelligent Transportation Systems* 21(5), 2019: pp. 2012–2028

[54] Chen, C. X.; Zaniolo, C.: *SQL$^{\mathrm{ST}}$: A Spatio-Temporal Data Model and Query Language*. In *Proceedings of the International Conference on Conceptual Modeling*. Springer, 2000, Volume 1920 by *Lecture Notes in Computer Science*, pp. 96–111

[55] Chen, K.; Zdorova, M.; Nathan-Roberts, D.: *Implications of Wearables, Fitness Tracking Services, and Quantified Self on Healthcare*. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 61(1), 2017: pp. 1066–1070

[56] Chen, L.; Özsu, M. T.; Oria, V.: *Robust and fast similarity search for moving object trajectories*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2005, pp. 491–502

[57] Chen, M.; Xu, M.; Franti, P.: *Compression of GPS trajectories*. In *Proceedings of the IEEE DCC Data Compression Conference*. 2012, pp. 62–71

[58] Chen, S.; Ooi, B. C.; Tan, K.-L.; Nascimento, M. A.: *ST^{2}B-tree: a self-tunable spatio-temporal b^+-tree index for moving objects*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2008, pp. 29–42

[59] Chen, X.; Zhang, C.; Ge, B.; Xiao, W.: *Spatio-temporal queries in HBase*. In *Proceedings of the IEEE International Conference on Big Data (Big Data)*. 2015, pp. 1929–1937

[60] Chen, Z.; Shen, H. T.; Zhou, X.: *Discovering popular routes from trajectories*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. IEEE, 2011, pp. 900–911

[61] Cheng, C.; Yang, H.; Lyu, M. R.; King, I.: *Where you like to go next: Successive point-of-interest recommendation*. In *Proceedings of the IJCAI International Joint Conference on Artificial Intelligence*. 2013, pp. 2605–2611

[62] Cheng, Y.; Iqbal, M. S.; Gupta, A.; Butt, A. R.: *Cast: Tiering storage for data analytics in the cloud*. In *Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing*. 2015, pp. 45–56

[63] Chou, H.-T.; DeWitt, D. J.: *An evaluation of buffer management strategies for relational database systems*. In *Algorithmica* 1(1), 1986: pp. 311–336

[64] Cici, B.; Markopoulou, A.; Laoutaris, N.: *SORS: a scalable online ridesharing system*. In *Proceedings of the ACM SIGSPATIAL International Workshop on Computational Transportation Science*. 2016, pp. 13–18

[65] Collet, Y.: *Lz4: Extremely fast compression algorithm* 2013

[66] Cudre-Mauroux, P.; Wu, E.; Madden, S.: *Trajstore: An adaptive storage system for very large trajectory data sets*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. 2010, pp. 109–120

[67] Curran, K.; Furey, E.; Lunney, T.; Santos, J.; Woods, D.; McCaughey, A.: *An evaluation of indoor location determination technologies*. In *Journal of Location Based Services* 5(2), 2011: pp. 61–78

[68] Damme, P.; Habich, D.; Hildebrandt, J.; Lehner, W.: *Lightweight Data Compression Algorithms: An Experimental Survey (Experiments and Analyses)*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2017, pp. 72–83

[69] Damme, P.; Ungethüm, A.; Hildebrandt, J.; Habich, D.; Lehner, W.: *From a Comprehensive Experimental Survey to a Cost-based Selection Strategy for Lightweight Integer Compression Algorithms*. In *ACM Trans. Database Syst.* 44(3), 2019: pp. 9:1–9:46

[70] Dantzig, G. B.: *Linear Programming and Extensions* 1963

[71] Das, M.; Ghosh, S. K.: *Data-driven approaches for spatio-temporal analysis: A survey of the state-of-the-arts*. In *Journal of Computer Science and Technology* 35(3), 2020: pp. 665–696

[72] Das, S.; Grbic, M.; Ilic, I.; Jovandic, I.; Jovanovic, A.; Narasayya, V. R.; Radulovic, M.; Stikic, M.; Xu, G.; Chaudhuri, S.: *Automatically indexing millions of databases in microsoft azure sql database*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2019, pp. 666–679

[73] Dash, D.; Polyzotis, N.; Ailamaki, A.: *CoPhy: A Scalable, Portable, and Interactive Index Advisor for Large Workloads*. In *Proceedings of the VLDB Endowment*. 2011, Volume 4, pp. 362–372

[74] Dean, J.; Ghemawat, S.: *MapReduce: simplified data processing on large clusters*. In *Communications of the ACM* 51(1), 2008: pp. 107–113

[75] DeBrabant, J.; Arulraj, J.; Pavlo, A.; Stonebraker, M.; Zdonik, S.; Dulloor, S.: *A Prolegomenon on OLTP Database Systems for Non-Volatile Memory*. In *Proceedings of the International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS)* 2014: pp. 57–63

[76] DeBrabant, J.; Pavlo, A.; Tu, S.; Stonebraker, M.; Zdonik, S.: *Anti-caching: A new approach to database management system architecture*. In *Proceedings of the VLDB Endowment* 6(14), 2013: pp. 1942–1953

[77] Deori, B.; Thounaojam, D. M.: *A survey on moving object tracking in video*. In *International Journal on Information Theory (IJIT)* 3(3), 2014: pp. 31–46

[78] Diaconu, C.; Freedman, C.; Ismert, E.; Larson, P.-A.; Mittal, P.; Stonecipher, R.; Verma, N.; Zwilling, M.: *Hekaton: SQL server's memory-optimized OLTP engine*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2013, pp. 1243–1254

[79] Ding, X.; Chen, L.; Gao, Y.; Jensen, C. S.; Bao, H.: *UlTraMan: a unified platform for big trajectory data management and analytics*. In *Proceedings of the VLDB Endowment* 11(7), 2018: pp. 787–799

[80] Dogramadzi, M.; Khan, A.: *Accelerated Map Matching for GPS Trajectories*. In *IEEE Transactions on Intelligent Transportation Systems* 23(5), 2022: pp. 4593–4602

[81] Dong, X. L.; Srivastava, D.: *Big data integration*. In *Synthesis Lectures on Data Management* 7(1), 2015: pp. 1–198

[82] Donnay, J. D.: *Spherical Trigonometry*. Read Books Ltd, 2011

[83] Douglas, D. H.; Peucker, T. K.: *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature*. In *Cartographica: The International Journal for Geographic Information and Geovisualization* 10(2), 1973: pp. 112–122

[84] Dreseler, M.: *Automatic Tiering for In-Memory Database Systems*. Dissertation, University of Potsdam, Germany, 2022

[85] Dreseler, M.; Boissier, M.; Rabl, T.; Uflacker, M.: *Quantifying TPC-H choke points and their optimizations*. In *Proceedings of the VLDB Endowment* 13(8), 2020: pp. 1206–1220

[86] Dreseler, M.; Kossmann, J.; Boissier, M.; Klauck, S.; Uflacker, M.; Plattner, H.: *Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 3 2019, pp. 313–324

[87] Duan, S.; Thummala, V.; Babu, S.: *Tuning database configuration parameters with ituned*. In *Proceedings of the VLDB Endowment* 2(1), 2009: pp. 1246–1257

[88] Dulloor, S. R.; Roy, A.; Zhao, Z.; Sundaram, N.; Satish, N.; Sankaran, R.; Jackson, J.; Schwan, K.: *Data tiering in heterogeneous memory systems*. In *Proceedings of the European Conference on Computer Systems (EuroSys)*. ACM, 2016, pp. 15:1–15:16

[89] Dunning, I.; Huchette, J.; Lubin, M.: *JuMP: A modeling language for mathematical optimization*. In *SIAM review* 59(2), 2017: pp. 295–320

[90] Dyreson, C. E.; Snodgrass, R. T.: *Timestamp semantics and representation*. In *Information Systems* 1993: pp. 143–166

[91] Egenhofer, M. J.: *Spatial SQL: A query and presentation language*. In *IEEE Transactions on knowledge and data engineering* 6(1), 1994: pp. 86–95

[92] Eiter, T.; Mannila, H.: *Computing Discrete Fréchet Distance*. White paper, Citeseer, 1994

[93] Eldawy, A.; Levandoski, J.; Larson, P.-Å.: *Trekking through siberia: Managing cold data in a memory-optimized database*. In *Proceedings of the VLDB Endowment* 7(11), 2014: pp. 931–942

[94] Eldawy, A.; Mokbel, M. F.: *Spatialhadoop: A mapreduce framework for spatial data*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. 2015, pp. 1352–1363

[95] Erwig, M.; Schneider, M.: *STQLA spatio-temporal query language*. In *Mining Spatio-Temporal Information Systems*, Springer. 2002, pp. 105–126

[96] Fang, Z.; Chen, L.; Gao, Y.; Pan, L.; Jensen, C. S.: *Dragoon: a hybrid and efficient big trajectory management system for offline and online analytics*. In *VLDB Journal* 30(2), 2021: pp. 287–310

[97] Färber, F.; Kemper, A.; Larson, P.-Å.; Levandoski, J.; Neumann, T.; Pavlo, A.; et al.: *Main memory database systems*. In *Foundations and Trends in Databases* 8(1-2), 2017: pp. 1–130

[98] Färber, F.; May, N.; Lehner, W.; Grosse, P.; Müller, I.; Rauhe, H.; Dees, J.: *The SAP HANA Database–An Architecture Overview*. In *IEEE Data Eng. Bull.* 35(1), 2012: pp. 28–33

[99] Faust, M.; Boissier, M.; Keller, M.; Schwalb, D.; Bischoff, H.; Eisenreich, K.; Färber, F.; Plattner, H.: *Footprint reduction and uniqueness enforcement with hash indices in SAP HANA*. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*. Springer, 2016, pp. 137–151

[100] Faust, M.; Schwalb, D.; Krüger, J.; Plattner, H.: *Fast Lookups for In-Memory Column Stores: Group-Key Indices, Lookup and Maintenance*. In *Proceedings of the International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS)*. 2012, pp. 13–22

[101] Faust, M.; Schwalb, D.; Plattner, H.: *Composite Group-Keys: Space-efficient Indexing of Multiple Columns for Compressed In-Memory Column Stores*. In *Proceedings of the International Workshop on In Memory Data Management and Analytics (IMDM)*. 2014, pp. 42–54

[102] Fazzinga, B.; Flesca, S.; Furfaro, F.; Parisi, F.: *Cleaning Trajectory Data of RFID-Monitored Objects Through Conditioning under Integrity Constraints*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2014, pp. 379–390

[103] Fazzinga, B.; Flesca, S.; Furfaro, F.; Parisi, F.: *Offline cleaning of RFID trajectory data*. In *Proceedings of the ACM SSDBM International Conference on Scientific and Statistical Database Management*. 2014, pp. 5:1–5:12

[104] Feng, Z.; Zhu, Y.: *A survey on trajectory data mining: Techniques and applications*. In *IEEE Access* 4, 2016: pp. 2056–2067

[105] Ferguson, R. O.; Sargent, L. F.: *Linear programming*, Volume 19. McGraw-Hill, 1958

[106] Fourer, R.; Gay, D. M.; Kernighan, B. W.: *A modeling language for mathematical programming*. In *Management Science* 36(5), 1990: pp. 519–554

[107] Froehlich, J.; Krumm, J.: *Route Prediction from Trip Observations*. In *Society of Automotive Engineers (SAE)*. 2008

[108] Fuentes, A. D.; Almeida, A. C.; de Carvalho Costa, R. L.; Braganholo, V.; Lifschitz, S.: *Database Tuning with Partial Indexes*. In *Simpósio Brasileiro de Banco de Dados (SBBD)*. 2018, pp. 181–192

[109] Galić, Z.; Mešković, E.; Osmanović, D.: *Distributed processing of big mobility data as spatio-temporal data streams*. In *Geoinformatica* 21(2), 2017: pp. 263–291

[110] Garcia-Molina, H.; Ullman, J. D.; Widom, J.: *Database systems - the complete book*. Pearson Education, 2009

[111] Garus, A.; Alonso, B.; Alonso Raposo, M.; Ciuffo, B.; dellOlio, L.: *Impact of New Mobility Solutions on Travel Behaviour and Its Incorporation into Travel Demand Models*. In *Journal of Advanced Transportation* 2022: pp. 1–24

[112] Ghemawat, S.; Gobioff, H.; Leung, S.-T.: *The Google file system*. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. 2003, pp. 29–43

[113] Ghosh, S.; Ghosh, S. K.; Buyya, R.: *MARIO: A spatio-temporal data mining framework on Google Cloud to explore mobility dynamics from taxi trajectories*. In *Journal of Network and Computer Applications (164)* 2020: p. 102692

[114] Gonzalez, M. C.; Hidalgo, C. A.; Barabasi, A.-L.: *Understanding individual human mobility patterns*. In *nature* 453(7196), 2008: pp. 779–782

[115] Graefe, G.; Kuno, H.: *Self-selecting, self-tuning, incrementally optimized indexes*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2010, pp. 371–381

[116] Grund, M.; Krüger, J.; Plattner, H.; Zeier, A.; Cudre-Mauroux, P.; Madden, S.: *Hyrise: a main memory hybrid storage engine*. In *Proceedings of the VLDB Endowment* 4(2), 2010: pp. 105–116

[117] Guan, X.; Bo, C.; Li, Z.; Yu, Y.: *ST-hash: An efficient spatiotemporal index for massive trajectory data in a NoSQL database*. In *IEEE International Conference on Geoinformatics (Geoinformatics)*. 2017, pp. 1–7

[118] Gudmundsson, J.; Horton, M.: *Spatio-temporal analysis of team sports*. In *ACM Computing Surveys (CSUR)* 50(2), 2017: pp. 1–34

[119] Guerra, J.; Pucha, H.; Glider, J.; Belluomini, W.; Rangaswami, R.: *Cost effective storage using extent based dynamic tiering*. In *USENIX Conference on File and Storage Technologies (FAST)*. 2011, pp. 273–286

[120] Guo, N.; Ma, M.; Xiong, W.; Chen, L.; Jing, N.: *An Efficient Query Algorithm for Trajectory Similarity Based on Fréchet Distance Threshold*. In *ISPRS International Journal of Geo-Information* 6(11), 2017: p. 326

[121] Gurobi Optimization, LLC: *Gurobi Optimizer Reference Manual*, 2021

[122] Güting, R. H.: *An introduction to spatial database systems*. In *VLDB Journal* 3(4), 1994: pp. 357–399

[123] Güting, R. H.; Behr, T.; Xu, J.: *Efficient k-nearest neighbor search on moving object trajectories*. In *VLDB Journal* 19(5), 2010: pp. 687–714

[124] Guttman, A.: *R-Trees: A Dynamic Index Structure for Spatial Searching*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1984, pp. 47–57

[125] Hadjieleftheriou, M.; Kollios, G.; Bakalov, P.; Tsotras, V. J.: *Complex spatio-temporal pattern queries*. In *Proceedings of the VLDB Endowment* 5, 2005: pp. 877–888

[126] Hagedorn, S.: *Efficient Processing of Large-scale Spatio-temporal Data*. Dissertation, Technische Universität Ilmenau, Germany, 2020

[127] Hagedorn, S.; Gotze, P.; Sattler, K.-U.: *The STARK framework for spatio-temporal data analytics on spark*. In *Datenbanksysteme für Business, Technologie und Web (BTW)* 2017: pp. 123–142

[128] Halfpap, S.; Schlosser, R.: *Workload-driven fragment allocation for partially replicated databases using linear programming*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. 2019, pp. 1746–1749

[129] Halfpap, S.; Schlosser, R.: *Exploration of Dynamic Query-Based Load Balancing for Partially Replicated Database Systems with Node Failures*. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CKIM)*. 2020, pp. 3409–3412

[130] Hammer, M.; Niamir, B.: *A heuristic approach to attribute partitioning*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1979, pp. 93–101

[131] Härder, T.; Rahm, E.: *Datenbanksysteme: Konzepte und Techniken der Implementierung*. Springer-Verlag, 2013

[132] Harizopoulos, S.; Abadi, D. J.; Madden, S.; Stonebraker, M.: *OLTP through the looking glass, and what we found there*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2008, p. 981992

[133] Hart, W. E.; Watson, J.-P.; Woodruff, D. L.: *Pyomo: Modeling and Solving Mathematical Programs in Python*. In *Mathematical Programming Computation* 3(3), 2011: pp. 219–260

[134] Hatwar, R. B.; Kamble, S. D.; Thakur, N. V.; Kakde, S.: *A review on moving object detection and tracking methods in video*. In *International Journal of Pure and Applied Mathematics* 118(16), 2018: pp. 511–526

[135] He, S.; Bastani, F.; Abbar, S.; Alizadeh, M.; Balakrishnan, H.; Chawla, S.; Madden, S.: *RoadRunner: improving the precision of road network inference from GPS trajectories*. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2018, pp. 3–12

[136] Heinzl, L.; Hurdelhey, B.; Boissier, M.; Perscheid, M.; Plattner, H.: *Evaluating Lightweight Integer Compression Algorithms in Column-Oriented In-Memory DBMS*. In *Proceedings of the International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures (ADMS)*. 2021, pp. 26–36

[137] Herodotou, H.; Kakoulli, E.: *Automating Distributed Tiered Storage Management in Cluster Computing*. In *Proceedings of the VLDB Endowment* 13(1), 2019: pp. 43–56

[138] Holze, M.; Ritter, N.: *Autonomic databases: Detection of workload shifts with n-gram-models*. In *Proceedings of the East European Conference on Advances in Databases and Information Systems (ADBIS)*. Springer, 2008, pp. 127–142

[139] Hoseinzadeh, M.: *A Survey on Tiering and Caching in High-Performance Storage Systems*. In *CoRR* abs/1904.11560, 2019

[140] Hu, J.; Yang, B.; Guo, C.; Jensen, C. S.: *Risk-Aware Path Selection with Time-Varying, Uncertain Travel CostsA Time Series Approach*. In *VLDB Journal* 27, 2018: pp. 179–200

[141] Huang, L.; Wen, Y.; Guo, W.; Zhu, X.; Zhou, C.; Zhang, F.; Zhu, M.: *Mobility pattern analysis of ship trajectories based on semantic transformation and topic model*. In *Ocean Engineering* 201, 2020: p. 107092

[142] Huang, S.; Wei, Q.; Feng, D.; Chen, J.; Chen, C.: *Improving Flash-Based Disk Cache with Lazy Adaptive Replacement*. In *ACM Transactions on Storage* 12(2), 2016: pp. 1–24

[143] Hughes, J. N.; Annex, A.; Eichelberger, C. N.; Fox, A.; Hulbert, A.; Ronquest, M.: *Geomesa: A Distributed Architecture for Spatio-Temporal Fusion*. In *Geospatial Informatics, Fusion, and Motion Video Analytics V*. 2015, Volume 9473, pp. 128–140

[144] Iliadis, I.; Jelitto, J.; Kim, Y.; Sarafijanovic, S.; Venkatesan, V.: *ExaPlan: Efficient Queueing-Based Data Placement, Provisioning, and Load Balancing for Large Tiered Storage Systems*. In *ACM Transactions on Storage* 13(2), 2017: pp. 1–41

[145] Jensen, C. S.; Lin, D.; Ooi, B. C.: *Query and Update Efficient B+-Tree Based Indexing of Moving Objects*. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. 2004, pp. 768–779

[146] Jensen, C. S.; Tielsytye, D.; Tradilauskas, N.: *Robust B+-Tree-Based Indexing of Moving Objects*. In *Proceedings of the IEEE International Conference on Mobile Data Management (MDM)*. 2006, pp. 1–12

[147] Jeung, H.; Lu, H.; Sathe, S.; Yiu, M. L.: *Managing evolving uncertainty in trajectory databases*. In *IEEE Transactions on Knowledge and Data Engineering* 26(7), 2013: pp. 1692–1705

[148] Jeung, H.; Yiu, M. L.; Zhou, X.; Jensen, C. S.: *Path prediction and predictive range querying in road network databases*. In *VLDB Journal* 19(4), 2010: pp. 585–602

[149] Jiang, H.; Liu, C.; Jin, Q.; Paparrizos, J.; Elmore, A. J.: *PIDS: attribute decomposition for improved compression and query performance in columnar storage*. In *Proceedings of the VLDB Endowment* 13(6), 2020: pp. 925–938

[150] Jiang, H.; Liu, C.; Paparrizos, J.; Chien, A. A.; Ma, J.; Elmore, A. J.: *Good to the Last Bit: Data-Driven Encoding with CodecDB*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2021, pp. 843–856

[151] Jiang, Z.; Zhang, Y.; Wang, J.; Xing, C.: *A Cost-aware Buffer Management Policy for Flash-based Storage Devices*. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*. Springer, 2015, pp. 175–190

[152] Jin, P.; Ou, Y.; Härder, T.; Li, Z.: *AD-LRU: An efficient buffer replacement algorithm for flash-based databases*. In *Data & Knowledge Engineering* 72, 2012: pp. 83–102

[153] Jung, J.; Jayakrishnan, R.; Park, J. Y.: *Design and Modeling of Real-Time Shared-Taxi Dispatch Algorithms*. In *TRB Annual Meeting*. 2013, Volume 9, pp. 1–20

[154] Jung, Y. C.; Youn, H. Y.; Kim, U.: *Efficient Indexing of Moving Objects Using Time-Based Partitioning with R-Tree*. In *Proceedings of the International Conference on Computational Science (ICCS)*. Springer, 2005, pp. 568–575

[155] Kakoulli, E.; Herodotou, H.: *OctopusFS: A distributed file system with tiered storage management*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2017, pp. 65–78

[156] Kallman, R.; Kimura, H.; Natkins, J.; Pavlo, A.; Rasin, A.; Zdonik, S.; Jones, E. P.; Madden, S.; Stonebraker, M.; Zhang, Y.; et al.: *H-store: a high-performance, distributed main memory transaction processing system*. In *Proceedings of the VLDB Endowment* 1(2), 2008: pp. 1496–1499

[157] Karimi, H. A.; Liu, X.: *A Predictive Location Model for Location-Based Services*. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems (GIS)*. 2003, pp. 126–133

[158] Kazmaier, G. S.; Mindnich, T.; Weyerhaeuser, C.; Baeumges, D.: *Managing and querying spatial point data in column stores*, 2021. US Patent 10,929,501

[159] Kellaris, G.; Pelekis, N.; Theodoridis, Y.: *Map-matched trajectory compression*. In *Journal of Systems and Software* 86(6), 2013: pp. 1566–1579

[160] Kemper, A.; Neumann, T.: *HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. 2011, pp. 195–206

[161] Keogh, E.; Chu, S.; Hart, D.; Pazzani, M.: *An online algorithm for segmenting time series*. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. IEEE, 2001, pp. 289–296

[162] Kimura, H.; Narasayya, V.; Syamala, M.: *Compression Aware Physical Database Design*. In *Proceedings of the VLDB Endowment* 4(10), 2011: pp. 657–668

[163] Koide, S.; Xiao, C.; Ishikawa, Y.: *Fast Subtrajectory Similarity Search in Road Networks under Weighted Edit Distance Constraints*. In *Proceedings of the VLDB Endowment* 13(11), 2020: pp. 2188–2201

[164] Kong, X.; Li, M.; Ma, K.; Tian, K.; Wang, M.; Ning, Z.; Xia, F.: *Big trajectory data: A survey of applications and services*. In *IEEE Access* 6, 2018: pp. 58295–58306

[165] Kossmann, J.: *Self-Driving: From General Purpose to Specialized DBMSs*. In *Proceedings of the VLDB PhD Workshop*. 2018

[166] Kossmann, J.; Halfpap, S.; Jankrift, M.; Schlosser, R.: *Magic mirror in my hand, which is the best in the land? An Experimental Evaluation of Index Selection Algorithms*. In *Proceedings of the VLDB Endowment* 13(11), 2020: pp. 2382–2395

[167] Kossmann, J.; Kastius, A.; Schlosser, R.: *SWIRL: Selection of Workload-aware Indexes using Reinforcement Learning*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2022, pp. 2:155–2:168

[168] Kossmann, J.; Lindner, D.; Naumann, F.; Papenbrock, T.: *Workload-driven, Lazy Discovery of Data Dependencies for Query Optimization*. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. 2022

[169] Kossmann, J.; Schlosser, R.: *Self-driving database systems: a conceptual approach*. In *Distributed Parallel Databases* 38(4), 2020: pp. 795–817

[170] Koubarakis, M.; Kyzirakos, K.: *Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL*. In *Proceedings of the Extended Semantic Web Conference (ESWC)*. 2010, pp. 425–439

[171] Kowark, T.; Richly, K.; Uflacker, M.; Plattner, H.: *Incremental, Per-Query Ontology Matching with RepMine*. In *Proceedings of the International Conference Companion on World Wide Web (WWW)*. 2016, pp. 215–218

[172] Kowark, T.; Teusner, R.; Richly, K.; Plattner, H.: *RepMine: A System for Transferrable Analyses of Collaboration Activities in Software Engineering*. In *IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*. 2015, pp. 78–81

[173] Kraska, T.; Beutel, A.; Chi, E. H.; Dean, J.; Polyzotis, N.: *The Case for Learned Index Structures*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2018, pp. 489–504

[174] Krish, K.; Anwar, A.; Butt, A. R.: *hats: A heterogeneity-aware tiered storage for hadoop*. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 2014, pp. 502–511

[175] Krumm, J.: *A markov model for driver turn prediction* 2008

[176] Kumar, M.: *World geodetic system 1984: A modern and accurate global reference frame*. In *Marine Geodesy* 12(2), 1988: pp. 117–126

[177] Lahiri, T.; Neimat, M.-A.; Folkman, S.: *Oracle TimesTen: An In-Memory Database for Enterprise Applications*. In *IEEE Data Eng. Bull.* 36(2), 2013: pp. 6–13

[178] Lang, H.; Mühlbauer, T.; Funke, F.; Boncz, P. A.; Neumann, T.; Kemper, A.: *Data blocks: Hybrid OLTP and OLAP on compressed storage using both vectorization and compilation*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2016, pp. 311–326

[179] Lange, R.; Dürr, F.; Rothermel, K.: *Scalable processing of trajectory-based queries in space-partitioned moving objects databases*. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2008, pp. 1–10

[180] Lasch, R.; Legler, T.; May, N.; Scheirle, B.; Sattler, K.-U.: *Cost modelling for optimal data placement in heterogeneous main memory*. In *Proceedings of the VLDB Endowment* 15(11), 2022: pp. 2867–2880

[181] Lassoued, Y.; Monteil, J.; Gu, Y.; Russo, G.; Shorten, R.; Mevissen, M.: *A Hidden Markov model for route and destination prediction*. In *Proceedings of the IEEE International Conference on Intelligent Transportation Systems (ITSC)*. 2017, pp. 1–6

[182] Laurini, R.; Paolino, L.; Sebillo, M.; Tortora, G.; Vitiello, G.: *A spatial SQL extension for continuous field querying*. In *Proceedings of the IEEE International Computer Software and Applications Conference (COMPSAC)*. 2004, pp. 78–81

[183] Laurini, R.; Thompson, D.: *Fundamentals of spatial information systems*, Volume 37 by *A.P.I.C. series*. Academic press, 1992

[184] Lee, D.; Chang, A.; Ahn, M.; Gim, J.; Kim, J.; Jung, J.; Choi, K.-W.; Pham, V.; Rebholz, O.; Malladi, K.; et al.: *Optimizing Data Movement with Near-Memory Acceleration of In-memory DBMS*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2020, pp. 371–374

[185] Lee, E.; Bahn, H.: *Caching strategies for high-performance storage media*. In *ACM Transactions on Storage* 10(3), 2014: pp. 1–22

[186] Lee, J.; Muehle, M.; May, N.; Faerber, F.; Sikka, V.; Plattner, H.; Krueger, J.; Grund, M.: *High-Performance Transaction Processing in SAP HANA*. In *IEEE Data Eng. Bull.* 36(2), 2013: pp. 28–33

[187] Lee, J.-G.; Han, J.; Li, X.; Gonzalez, H.: *TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering*. In *Proceedings of the VLDB Endowment* 1(1), 2008: pp. 1081–1094

[188] Lee, J.-G.; Han, J.; Whang, K.-Y.: *Trajectory clustering: a partition-and-group framework*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2007, pp. 593–604

[189] Leis, V.; Haubenschild, M.; Kemper, A.; Neumann, T.: *LeanStore: In-memory data management beyond main memory*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. 2018, pp. 185–196

[190] Lemire, D.; Boytsov, L.: *Decoding billions of integers per second through vectorization*. In *Software: Practice and Experience* 45(1), 2015: pp. 1–29

[191] Levandoski, J. J.; Larson, P.; Stoica, R.: *Identifying Hot and Cold Data in Main-Memory Databases*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2013, pp. 26–37

[192] Li, R.; Feng, W.; Wu, H.; Huang, Q.: *A replication strategy for a distributed high-speed caching system based on spatiotemporal access patterns of geospatial data*. In *Computers, Environment and Urban Systems* 61, 2017: pp. 163–171

[193] Li, R.; He, H.; Wang, R.; Huang, Y.; Liu, J.; Ruan, S.; He, T.; Bao, J.; Zheng, Y.: *Just: Jd urban spatio-temporal data engine*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. 2020, pp. 1558–1569

[194] Li, R.; He, H.; Wang, R.; Ruan, S.; He, T.; Bao, J.; Zhang, J.; Hong, L.; Zheng, Y.: *TrajMesa: A Distributed NoSQL-Based Trajectory Data Management System*. In *IEEE Transactions on Knowledge and Data Engineering* 35(1), 2021: pp. 1013–1027

[195] Li, R.; He, H.; Wang, R.; Ruan, S.; Sui, Y.; Bao, J.; Zheng, Y.: *TrajMesa: A Distributed NoSQL Storage Engine for Big Trajectory Data*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. 2020, pp. 2002–2005

[196] Li, Y.; Chow, C.-Y.; Deng, K.; Yuan, M.; Zeng, J.; Zhang, J.-D.; Yang, Q.; Zhang, Z.-L.: *Sampling big trajectory data*. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*. 2015, pp. 941–950

[197] Li, Y.; Luo, J.; Chow, C.-Y.; Chan, K.-L.; Ding, Y.; Zhang, F.: *Growing the charging station network for electric vehicles with trajectory data analytics*. In *Proceedings of the IEEE ICDE International Conference on Data Engineering*. 2015, pp. 1376–1387

[198] Li, Z.; Ji, M.; Lee, J.-G.; Tang, L.-A.; Yu, Y.; Han, J.; Kays, R.: *MoveMine: mining moving object databases*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2010, pp. 1203–1206

[199] Liao, Z.: *Real-time taxi dispatching using global positioning systems*. In *Communications of the ACM* 46(5), 2003: pp. 81–83

[200] Liebner, M.; Baumann, M.; Klanner, F.; Stiller, C.: *Driver intent inference at urban intersections using the intelligent driver model*. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. 2012, pp. 1162–1167

[201] Lima, A. A.; Furtado, C.; Valduriez, P.; Mattoso, M.: *Parallel OLAP query processing in database clusters with data replication*. In *Distributed and Parallel Databases* 25(1-2), 2009: pp. 97–123

[202] Lindner, D.; Löser, A.; Kossmann, J.: *Learned What-If Cost Models for Autonomous Clustering*. In *Proceedings of the European Conference on Advances in Databases and Information Systems (ADBIS)*. Springer, 2021, pp. 3–13

[203] Liu, J.; Zhao, K.; Sommer, P.; Shang, S.; Kusy, B.; Jurdak, R.: *Bounded Quadrant System: Error-bounded trajectory compression on the go*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2015, pp. 987–998

[204] Liu, J.; Zhao, K.; Sommer, P.; Shang, S.; Kusy, B.; Lee, J.-G.; Jurdak, R.: *A novel framework for online amnesic trajectory compression in*

resource-constrained environments. In *IEEE Transactions on Knowledge and Data Engineering* 28(11), 2016: pp. 2827–2841

[205] Liu, X.; Karimi, H. A.: *Location awareness through trajectory prediction*. In *Computers, Environment and Urban Systems* 30(6), 2006: pp. 741–756

[206] Liu, X.; Salem, K.: *Hybrid storage management for database systems*. In *Proceedings of the VLDB Endowment* 6(8), 2013: pp. 541–552

[207] Liu, Y.; Seah, H. S.: *Points of interest recommendation from GPS trajectories*. In *International Journal of Geographical Information Science* 29(6), 2015: pp. 953–979

[208] Lomet, D. B.: *Cost/performance in modern data stores: how data caching systems succeed*. In *Proceedings of the International Workshop on Data Management on New Hardware (DaMoN)*. 2018, pp. 9:1–9:10

[209] Long, C.; Wong, R. C.-W.; Jagadish, H.: *Direction-preserving trajectory simplification*. In *Proceedings of the VLDB Endowment* 6(10), 2013: pp. 949–960

[210] Loos, P.; Lechtenbörger, J.; Vossen, G.; Zeier, A.; Krüger, J.; Müller, J.; Lehner, W.; Kossmann, D.; Fabian, B.; Günther, O.; et al.: *In-Memory-Datenmanagement in betrieblichen Anwendungssystemen*. In *Wirtschaftsinformatik* 53(6), 2011: pp. 383–390

[211] Lou, Y.; Zhang, C.; Zheng, Y.; Xie, X.; Wang, W.; Huang, Y.: *Map-matching for low-sampling-rate GPS trajectories*. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2009, pp. 352–361

[212] Luo, T.; Lee, R.; Mesnier, M.; Chen, F.; Zhang, X.: *hStorage-DB: Heterogeneity-aware Data Management to Exploit the Full Capability of Hybrid Storage Systems*. In *Proceedings of the VLDB Endowment* 5(10), 2012: pp. 1076–1087

[213] Luo, W.; Tan, H.; Chen, L.; Ni, L. M.: *Finding time period-based most frequent path in big trajectory data*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2013, pp. 713–724

[214] Ma, L.; Van Aken, D.; Hefny, A.; Mezerhane, G.; Pavlo, A.; Gordon, G. J.: *Query-based workload forecasting for self-driving database management systems*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2018, pp. 631–645

[215] Mahmood, A. R.; Punni, S.; Aref, W. G.: *Spatio-temporal access methods: a survey (2010-2017)*. In *GeoInformatica* 23(1), 2019: pp. 1–36

[216] Makris, A.; da Silva, C. L.; Bogorny, V.; Alvares, L. O.; de Macêdo, J. A. F.; Tserpes, K.: *Evaluating the effect of compressing algorithms for trajectory similarity and classification problems*. In *GeoInformatica* 25(4), 2021: pp. 679–711

[217] Makris, A.; Tserpes, K.; Spiliopoulos, G.; Zissis, D.; Anagnostopoulos, D.: *MongoDB Vs PostgreSQL: A comparative study on performance aspects*. In *GeoInformatica* 25, 2021: pp. 243–268

[218] Mandelman, J. A.; Dennard, R. H.; Bronner, G. B.; DeBrosse, J. K.; Divakaruni, R.; Li, Y.; Radens, C. J.: *Challenges and future directions for the scaling of dynamic random-access memory (DRAM)*. In *IBM Journal of Research and Development* 46(2.3), 2002: pp. 187–212

[219] Marcus, R.; Papaemmanouil, O.: *Plan-Structured Deep Neural Network Models for Query Performance Prediction*. In *Proceedings of the VLDB Endowment* 12(11), 2019: pp. 1733–1746

[220] Marcus, R.; Papaemmanouil, O.; Semenova, S.; Garber, S.: *NashDB: an end-to-end economic method for elastic database fragmentation, replication, and provisioning*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2018, pp. 1253–1267

[221] Martinez, D.; Cristobal, S.; Belkoura, S.: *Smart data fusion: Probabilistic record linkage adapted to merge two trajectories from different sources*. In *Proceedings of the SESAR* 2018

[222] Masoud, N.; Jayakrishnan, R.: *A real-time algorithm to solve the peer-to-peer ride-matching problem in a flexible ridesharing system*. In *Transportation Research Part B: Methodological* 106, 2017: pp. 218–236

[223] Matthies, C.; Kowark, T.; Richly, K.; Uflacker, M.; Plattner, H.: *How Surveys, Tutors, and Software Help to Assess Scrum Adoption in a Classroom Software Engineering Project*. In *Proceedings of the ACM International Conference on Software Engineering (ICSE)*. 2016, pp. 313–322

[224] Matthies, C.; Kowark, T.; Richly, K.; Uflacker, M.; Plattner, H.: *ScrumLint: Identifying Violations of Agile Practices Using Development Artifacts*. In *Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 2016, pp. 40–43

[225] May, N.; Lehner, W.; Shahul Hameed, P.; Maheshwari, N.; Müller, C.; Chowdhuri, S.; Goel, A. K.: *SAP HANA-From Relational OLAP Database to Big Data Infrastructure*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2015, pp. 581–592

[226] Mazimpaka, J. D.; Timpf, S.: *Trajectory data mining: A review of methods and applications*. In *Journal of Spatial Information Science* 13(1), 2016: pp. 61–99

[227] Mehta, P.; Kotlarski, M.; Skoutas, D.; Sacharidis, D.; Patroumpas, K.; Voisard, A.: *μTOP: Spatio-Temporal Detection and Summarization of Locally Trending Topics in Microblog Posts*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2017, pp. 558–561

[228] Mehta, P.; Sacharidis, D.; Skoutas, D.; Voisard, A.: *Finding Socio-Textual Associations Among Locations*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2017, pp. 120–131

[229] Meratnia, N.; Rolf, A.: *Spatiotemporal compression techniques for moving point objects*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2004, pp. 765–782

[230] Mikluscák, T.; Gregor, M.; Janota, A.: *Using Neural Networks for Route and Destination Prediction in Intelligent Transport Systems*. In *Proceedings of International Conference on Transport Systems Telematics (TST)*. 2012, pp. 380–387

[231] Moerkotte, G.: *Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing*. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. pp. 476–487

[232] Mokbel, M. F.; Ghanem, T. M.; Aref, W. G.: *Spatio-Temporal Access Methods*. In *IEEE Data Eng. Bull.* 26(2), 2003: pp. 40–49

[233] Morra, L.; Manigrasso, F.; Lamberti, F.: *SoccER: Computer graphics meets sports analytics for soccer event recognition*. In *SoftwareX* 12, 2020: p. 100612

[234] Muckell, J.; Hwang, J.-H.; Patil, V.; Lawson, C. T.; Ping, F.; Ravi, S.: *SQUISH: an online approach for GPS trajectory compression*. In *Proceedings of the International Conference on Computing for Geospatial Research & Applications*. 2011, pp. 13:1–13:8

[235] Müller, I.; Ratsch, C.; Färber, F.: *Adaptive String Dictionary Compression in In-Memory Column-Store Database Systems*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2014, pp. 283–294

[236] Nathan, V.; Ding, J.; Alizadeh, M.; Kraska, T.: *Learning multi-dimensional indexes*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2020, pp. 985–1000

[237] Neumann, T.; Freitag, M. J.: *Umbra: A Disk-Based System with In-Memory Performance*. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. 2020

[238] Newson, P.; Krumm, J.: *Hidden Markov map matching through noise and sparseness*. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2009, pp. 336–343

[239] Nibali, A.; He, Z.: *Trajic: An effective compression system for trajectory data*. In *IEEE Transactions on Knowledge and Data Engineering* 27(11), 2015: pp. 3138–3151

[240] Noulas, A.; Scellato, S.; Lathia, N.; Mascolo, C.: *Mining user mobility features for next place prediction in location-based services*. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. 2012, pp. 1038–1043

[241] Olma, M.; Karpathiotakis, M.; Alagiannis, I.; Athanassoulis, M.; Ailamaki, A.: *Adaptive Partitioning and Indexing for In Situ Query Processing*. In *VLDB Journal* 29(1), 2020: pp. 569–591

[242] Österreicher, F.; Vajda, I.: *A new class of metric divergences on probability spaces and its applicability in statistics*. In *Annals of the Institute of Statistical Mathematics* 55(3), 2003: pp. 639–653

[243] Panagiotakis, C.; Pelekis, N.; Kopanakis, I.; Ramasso, E.; Theodoridis, Y.: *Segmentation and sampling of moving object trajectories based on representativeness*. In *IEEE Transactions on Knowledge and Data Engineering* 24(7), 2012: pp. 1328–1343

[244] Pandey, V.; Kipf, A.; Vorona, D.; Mühlbauer, T.; Neumann, T.; Kemper, A.: *High-performance geospatial analytics in hyperspace*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2016, pp. 2145–2148

[245] Pandey, V.; van Renen, A.; Kipf, A.; Sabek, I.; Ding, J.; Kemper, A.: *The Case for Learned Spatial Indexes*. In *Proceedings of the International Workshop on Applied AI for Database Systems and Applications(AIDB)* 2020: pp. 1–9

[246] Pant, N.; Fouladgar, M.; Elmasri, R.; Jitkajornwanich, K.: *A survey of spatio-temporal database research*. In *Proceedings of the Asian Conference on Intelligent Information and Database Systems (ACIIDS)*.

Springer, 2018, Volume 10752 by *Lecture Notes in Computer Science*, pp. 115–126

[247] Park, C.; Sohn, S. Y.: *An optimization approach for the placement of bicycle-sharing stations to reduce short car trips: An application to the city of Seoul*. In *Transportation Research Part A: Policy and Practice* 105, 2017: pp. 154–166

[248] Patel, J. M.; Deshmukh, H.; Zhu, J.; Potti, N.; Zhang, Z.; Spehlmann, M.; Memisoglu, H.; Saurabh, S.: *Quickstep: A data platform based on the scaling-up approach*. In *Proceedings of the VLDB Endowment* 11(6), 2018: pp. 663–676

[249] Patil, V.; Singh, P.; Parikh, S.; Atrey, P. K.: *GeoSClean: Secure Cleaning of GPS Trajectory Data Using Anomaly Detection*. In *Proceedings of the IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. 2018, pp. 166–169

[250] Patrou, M.; Alam, M. M.; Memarzia, P.; Ray, S.; Bhavsar, V. C.; Kent, K. B.; Dueck, G. W.: *DISTIL: a distributed in-memory data processing system for location-based services*. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2018, pp. 496–499

[251] Patterson, D. J.; Liao, L.; Fox, D.; Kautz, H. A.: *Inferring High-Level Behavior from Low-Level Sensors*. In *Proceedings of the International Conference on Ubiquitous Computing*. 2003, pp. 73–89

[252] Pavlo, A.; Angulo, G.; Arulraj, J.; Lin, H.; Lin, J.; Ma, L.; Menon, P.; Mowry, T. C.; Perron, M.; Quah, I.; et al.: *Self-Driving Database Management Systems*. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. 2017

[253] Peixoto, D. A.; Zhou, X.; Hung, N. Q. V.; He, D.; Stantic, B.: *A System for Spatial-Temporal Trajectory Data Integration and Representation*. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*. Springer, 2018, pp. 807–812

[254] Pelekis, N.; Kopanakis, I.; Panagiotakis, C.; Theodoridis, Y.: *Unsupervised Trajectory Sampling*. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases* 2010: pp. 17–33

[255] Pelkonen, T.; Franklin, S.; Teller, J.; Cavallaro, P.; Huang, Q.; Meza, J.; Veeraraghavan, K.: *Gorilla: A fast, scalable, in-memory time series database*. In *Proceedings of the VLDB Endowment* 8(12), 2015: pp. 1816–1827

[256] Peng, I.; McFadden, M.; Green, E.; Iwabuchi, K.; Wu, K.; Li, D.; Pearce, R.; Gokhale, M.: *UMap: Enabling application-driven optimizations for page management*. In *Proceedings of the IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. 2019, pp. 71–78

[257] Peng, I. B.; Gokhale, M. B.; Youssef, K.; Iwabuchi, K.; Pearce, R.: *Enabling Scalable and Extensible Memory-Mapped Datastores in Userspace*. In *IEEE Transactions on Parallel and Distributed Systems* 33(4), 2022: pp. 866–877

[258] Pfoser, D.; Jensen, C. S.; Theodoridis, Y.: *Novel Approaches to the Indexing of Moving Object Trajectories*. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. 2000, pp. 395–406

[259] Phillips, D. J.; Wheeler, T. A.; Kochenderfer, M. J.: *Generalizable intention prediction of human drivers at intersections*. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 1665–1670

[260] Pirk, H.; Funke, F.; Grund, M.; Neumann, T.; Leser, U.; Manegold, S.; Kemper, A.; Kersten, M.: *CPU and cache efficient management of memory-resident databases*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2013, pp. 14–25

[261] Plattner, H.: *A common database approach for OLTP and OLAP using an in-memory column database*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2009, pp. 1–2

[262] Plattner, H.: *A Course in In-Memory Data Management*. Springer, 2013

[263] Plattner, H.: *The impact of columnar in-memory databases on enterprise systems: implications of eliminating transaction-maintained aggregates*. In *Proceedings of the VLDB Endowment* 7(13), 2014: pp. 1722–1729

[264] Plattner, H.; Zeier, A.: *In-memory data management: technology and applications*. Springer Science & Business Media, 2012

[265] Psaraftis, H. N.: *Dynamic Vehicle Routing: Status and Prospects*. In *Annals of Operations Research* 61(1), 1995: pp. 143–164

[266] Psaraftis, H. N.; Wen, M.; Kontovas, C. A.: *Dynamic Vehicle Routing Problems: Three Decades and Counting*. In *Networks* 67(1), 2016: pp. 3–31

[267] Psaroudakis, I.; Scheuer, T.; May, N.; Sellami, A.; Ailamaki, A.: *Scaling up concurrent main-memory column-store scans: towards adaptive NUMA-aware data and task placement*. In *Proceedings of the VLDB Endowment* 8(12), 2015: pp. 1442–1453

[268] Psaroudakis, I.; Scheuer, T.; May, N.; Sellami, A.; Ailamaki, A.: *Adaptive NUMA-aware data placement and task scheduling for analytical workloads in main-memory column-stores*. In *Proceedings of the VLDB Endowment* 10(2), 2016: pp. 37–48

[269] Quddus, M. A.; Ochieng, W. Y.; Zhao, L.; Noland, R. B.: *A General Map Matching Algorithm for Transport Telematics Applications*. In *GPS Solutions* 7(3), 2003: pp. 157–167

[270] Raman, V.; Swart, G.: *How to Wring a Table Dry: Entropy Compression of Relations and Querying of Compressed Relations*. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. 2006, pp. 858–869

[271] Ranu, S.; Deepak, P.; Telang, A. D.; Deshpande, P.; Raghavan, S.: *Indexing and matching trajectories under inconsistent sampling rates*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2015, pp. 999–1010

[272] Rasetic, S.; Sander, J.; Elding, J.; Nascimento, M. A.: *A Trajectory Splitting Model for Efficient Spatio-Temporal Indexing*. In *Proceedings of the International Conference on Very Large Data (VLDB)*. 2005, pp. 934–945

[273] Ratnasamy, S.; Francis, P.; Handley, M.; Karp, R.; Shenker, S.: *A scalable content-addressable network*. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies*. 2001, pp. 161–172

[274] Richly, K.: *Leveraging Spatio-Temporal Soccer Data to Define a Graphical Query Language for Game Recordings*. In *Proceedings of the IEEE International Conference on Big Data (BigData)*. 2018, pp. 3456–3463

[275] Richly, K.: *A Survey on Trajectory Data Management for Hybrid Transactional and Analytical Workloads*. In *Proceedings of the IEEE International Conference on Big Data (BigData)*. 2018, pp. 562–569

[276] Richly, K.: *Optimized Spatio-Temporal Data Structures for Hybrid Transactional and Analytical Workloads on Columnar In-Memory Databases*. In *Proceedings of the VLDB PhD Workshop*. 2019, pp. 1–4

[277] Richly, K.: *Memory-Efficient Storing of Timestamps for Spatio-Temporal Data Management in Columnar In-Memory Databases*. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*. 2021, pp. 542–557

[278] Richly, K.; Bothe, M.; Rohloff, T.; Schwarz, C.: *Recognizing Compound Events in Spatio-Temporal Football Data*. In *Proceedings of the International Conference on Internet of Things and Big Data (IoTBD)*. 2016, pp. 27–35

[279] Richly, K.; Brauer, J.; Schlosser, R.: *Predicting Location Probabilities of Drivers to Improve Dispatch Decisions of Transportation Network Companies based on Trajectory Data*. In *Proceedings of the International Conference on Operations Research and Enterprise Systems (ICORES)*. 2020, pp. 47–58

[280] Richly, K.; Lorenz, M.; Oergel, S.: *S4J – Integrating SQL into Java at Compiler-Level*. In *Proceedings of the International Conference on Information and Software Technologies (ICIST)*. Springer, 2016, Volume 639 by *Communications in Computer and Information Science*, pp. 300–315

[281] Richly, K.; Moritz, F.; Schwarz, C.: *Utilizing Artificial Neural Networks to Detect Compound Events in Spatio-Temporal Soccer Data*. In *Proceedings of the ACM SIGKDD Workshop on Mining and Learning from Time Series (MiLeTs)*. 2017, pp. 13–17

[282] Richly, K.; Schlosser, R.; Boissier, M.: *Joint Index, Sorting, and Compression Optimization for Memory-Efficient Spatio-Temporal Data Management*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2021, pp. 1901–1906

[283] Richly, K.; Schlosser, R.; Boissier, M.: *Budget-Conscious Fine-Grained Configuration Optimization for Spatio-Temporal Applications*. In *Proceedings of the VLDB Endowment* 15(13), 2022: pp. 4079 – 4092

[284] Richly, K.; Schlosser, R.; Brauer, J.: *Enabling Risk-averse Dispatch Processes for Transportation Network Companies by Probabilistic Location Prediction*. In *Operations Research and Enterprise Systems*. Springer, 2022, Volume 1623 by *Communications in Computer and Information Science*, pp. 21–42

[285] Richly, K.; Schlosser, R.; Brauer, J.; Plattner, H.: *A Probabilistic Location Prediction Approach to Optimize Dispatch Processes in the Ride-Hailing Industry*. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS)*. 2021, pp. 1830–1840

[286] Richly, K.; Teusner, R.: *Where is the Money Made? An Interactive Visualization of Profitable Areas in New York City*. In *Proceedings of the International Conference on IoT in Urban Space (Urb-IoT)*. ACM, 2016, pp. 43–46

[287] Richly, K.; Teusner, R.; Immer, A.; Windheuser, F.; Wolf, L.: *Optimizing Routes of Public Transportation Systems by Analyzing the Data of Taxi Rides*. In *Proceedings of the International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*. 2015, pp. 70–76

[288] Roddick, J. F.; Spiliopoulou, M.: *A bibliography of temporal, spatial and spatio-temporal data mining research*. In *ACM SIGKDD Explorations* 1(1), 1999: pp. 34–38

[289] Sacks-Davis, R.; McDonell, K.; Ooi, B.: *GEOQL-A query language for geographic information systems*. In *Technical Report 87/2, Monash University*. 1987

[290] Sagan, H.: *Space-filling curves*. Springer Science & Business Media, 2012

[291] Šaltenis, S.; Jensen, C. S.; Leutenegger, S. T.; Lopez, M. A.: *Indexing the positions of continuously moving objects*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2000, pp. 331–342

[292] Sandu Popa, I.; Zeitouni, K.; Oria, V.; Barth, D.; Vial, S.: *Indexing in-network trajectory flows*. In *VLDB Journal* 20(5), 2011: pp. 643–669

[293] Santipantakis, G. M.; Glenis, A.; Patroumpas, K.; Vlachou, A.; Doulkeridis, C.; Vouros, G. A.; Pelekis, N.; Theodoridis, Y.: *SPARTAN: Semantic integration of big spatio-temporal data from streaming and archival sources*. In *Future Generation Computer Systems* 110, 2020: pp. 540–555

[294] Sattler, K.-U.; Schallehn, E.; Geist, I.: *Towards Indexing Schemes for Self-Tuning DBMS*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE) Workshops*. 2005, p. 1216

[295] Schlosser, R.; Kossmann, J.; Boissier, M.: *Efficient Scalable Multi-Attribute Index Selection Using Recursive Strategies*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2019, pp. 1238–1249

[296] Schlosser, R.; Richly, K.: *Dynamic Pricing Competition with Unobservable Inventory Levels: A Hidden Markov Model Approach*. In *Operations Research and Enterprise Systems*. Springer, 2018, Volume 966 by *Communications in Computer and Information Science*, pp. 15–36

[297] Schlosser, R.; Richly, K.: *Dynamic Pricing Strategies in a Finite Horizon Duopoly with Partial Information*. In *Proceedings of the International Conference on Operations Research and Enterprise Systems (ICORES)*. 2018, pp. 21–30

[298] Schlosser, R.; Richly, K.: *Dynamic Pricing under Competition with Data-Driven Price Anticipations and Endogenous Reference Price Effects*. In *Journal of Revenue and Pricing Management* 18(6), 2019: pp. 451–464

[299] Schweizer, H.; Besta, M.; Hoefler, T.: *Evaluating the Cost of Atomic Operations on Modern Architectures*. In *Proceedings of the IEEE International Conference on Parallel Architecture and Compilation (PACT)*. 2015, pp. 445–456

[300] Seshadri, P.; Swami, A.: *Generalized Partial Indexes*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 1995, pp. 420–427

[301] Shang, S.; Lu, H.; Pedersen, T. B.; Xie, X.: *Finding Traffic-Aware Fastest Paths in Spatial Networks*. In *Proceedings of the International*

Symposium on Spatial and Temporal Databases (SSTD). 2013, pp. 128–145

[302] Shang, Z.; Li, G.; Bao, Z.: *Dita: Distributed In-Memory Trajectory Analytics*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2018, pp. 725–740

[303] Sherkat, R.; Florendo, C.; Andrei, M.; Blanco, R.; Dragusanu, A.; Pathak, A.; Khadilkar, P.; Kulkarni, N.; Lemke, C.; Seifert, S.; et al.: *Native store extension for SAP HANA*. In *Proceedings of the VLDB Endowment* 12(12), 2019: pp. 2047–2058

[304] Sherkat, R.; Florendo, C.; Andrei, M.; Goel, A. K.; Nica, A.; Bumbulis, P.; Schreter, I.; Radestock, G.; Bensberg, C.; Booss, D.; et al.: *Page as you go: Piecewise columnar access in SAP HANA*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2016, pp. 1295–1306

[305] Shoval, N.: *Tracking technologies and urban analysis*. In *Cities* 25(1), 2008: pp. 21–28

[306] Shvachko, K.; Kuang, H.; Radia, S.; Chansler, R.: *The hadoop distributed file system*. In *Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies (MSST)*. 2010, pp. 1–10

[307] Sikka, V.; Färber, F.; Lehner, W.; Cha, S. K.; Peh, T.; Bornhövd, C.: *Efficient transaction processing in SAP HANA database: the end of a column store myth*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2012, pp. 731–742

[308] Simmons, R. G.; Browning, B.; Zhang, Y.; Sadekar, V.: *Learning to Predict Driver Route and Destination Intent*. In *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*. 2006, pp. 127–132

[309] Snodgrass, R. T.: *The TSQL2 temporal query language*. Springer Science & Business Media, 2012

[310] Song, R.; Sun, W.; Zheng, B.; Zheng, Y.: *PRESS: A novel framework of trajectory compression in road networks*. In *Proceedings of the VLDB Endowment* 7(9), 2014: pp. 661–672

[311] Song, Z.; Roussopoulos, N.: *SEB-tree: An Approach to Index Continuously Moving Objects*. In *Proceedings of the International Conference on Mobile Data Management (MDM)*. 2003, pp. 340–344

[312] Stoica, R.; Ailamaki, A.: *Enabling efficient OS paging for main-memory OLTP databases*. In *Proceedings of the International Workshop on Data Management on New Hardware (DaMoN)*. 2013, pp. 1–7

[313] Stonebraker, M.: *Operating system support for database management*. In *Communications of the ACM* 24(7), 1981: pp. 412–418

[314] Stonebraker, M.: *The case for partial indexes*. In *ACM Sigmod Record* 18(4), 1989: pp. 4–11

[315] Stonebraker, M.; Abadi, D. J.; Batkin, A.; Chen, X.; Cherniack, M.; Ferreira, M.; Lau, E.; Lin, A.; Madden, S.; O'Neil, E. J.; O'Neil, P. E.; Rasin, A.; Tran, N.; Zdonik, S. B.: *C-Store: A Column-oriented DBMS*. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. 2005, pp. 553–564

[316] Stonebraker, M.; Kemnitz, G.: *The POSTGRES next generation database management system*. In *Communications of the ACM* 34(10), 1991: pp. 78–92

[317] Stonebraker, M.; Weisberg, A.: *The VoltDB Main Memory DBMS*. In *IEEE Data Eng. Bull.* 36(2), 2013: pp. 21–27

[318] Stougiannis, A.; Pavlovic, M.; Tauheed, F.; Heinis, T.; Ailamaki, A.: *Data-driven neuroscience: enabling breakthroughs via innovative data management*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2013, pp. 953–956

[319] Strötgen, J.; Gertz, M.: *TimeTrails: a system for exploring spatio-temporal information in documents*. In *Proceedings of the VLDB Endowment* 3(1-2), 2010: pp. 1569–1572

[320] Su, H.; Liu, S.; Zheng, B.; Zhou, X.; Zheng, K.: *A survey of trajectory distance measures and performance evaluation*. In *VLDB Journal* 29(1), 2020: pp. 3–32

[321] Su, H.; Zheng, K.; Huang, J.; Wang, H.; Zhou, X.: *Calibrating trajectory data for spatio-temporal similarity analysis*. In *VLDB Journal* 24(1), 2015: pp. 93–116

[322] Su, H.; Zheng, K.; Wang, H.; Huang, J.; Zhou, X.: *Calibrating trajectory data for similarity-based analysis*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2013, pp. 833–844

[323] Su, H.; Zheng, K.; Zeng, K.; Huang, J.; Zhou, X.: *STMaker: a system to make sense of trajectory data*. In *Proceedings of the VLDB Endowment* 7(13), 2014: pp. 1701–1704

[324] Subha, S.: *An Algorithm for Buffer Cache Management*. In *Proceedings of the IEEE International Conference on Information Technology: New Generations (ITNG)*. 2009, pp. 889–893

[325] Tailor, P.; Morena, R. D.: *A survey of database buffer cache management approaches*. In *International Journal of Advanced Research in Computer Science* 8(3), 2017: pp. 409–414

[326] Tajalli, M.; Hajbabaie, A.: *Traffic Signal Timing and Trajectory Optimization in a Mixed Autonomy Traffic Stream*. In *IEEE Transactions on Intelligent Transportation Systems* 23(7), 2022: pp. 6525–6538

[327] Tan, H.; Luo, W.; Ni, L. M.: *CloST: A Hadoop-Based Storage System for Big Spatio-Temporal Data Analytics*. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*. 2012, pp. 2139–2143

[328] Tang, M.; Yu, Y.; Malluhi, Q. M.; Ouzzani, M.; Aref, W. G.: *Locationspark: A distributed in-memory data management system for big spatial data*. In *Proceedings of the VLDB Endowment* 9(13), 2016: pp. 1565–1568

[329] Tanuja, V.; Govindarajulu, P.: *A Survey on Trajectory Data Mining*. In *International Journal of Computer Science and Security (IJCSS)* 10(5), 2016: pp. 195–214

[330] Tao, Y.; Papadias, D.; Sun, J.: *The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries*. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. 2003, pp. 790–801

[331] Tariq, Z. B.; Cheema, D. M.; Kamran, M. Z.; Naqvi, I. H.: *Non-GPS positioning systems: A survey*. In *ACM Computing Surveys (CSUR)* 50(4), 2017: pp. 1–34

[332] Taxi, N.; (TLC), L. C.: *Trip Record Data, 2020, (Accessed: 2023-02-01)*.
https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[333] Teusner, R.; Richly, K.; Staubitz, T.; Renz, J.: *Enhancing Content between Iterations of a MOOC–Effects on Key Metrics*. In *European MOOCs Stakeholder Summit* 2015: pp. 147–156

[334] Tian, Y.; Ji, Y.; Scholer, J.: *A prototype spatio-temporal database built on top of relational database*. In *Proceedings of the IEEE International Conference on Information Technology: New Generations (ITNG)*. 2015, pp. 14–19

[335] Toohey, K.; Duckham, M.: *Trajectory similarity measures*. In *ACM SIGSPATIAL Special* 7(1), 2015: pp. 43–50

[336] Trajcevski, G.; Ding, H.; Scheuermann, P.; Cruz, I. F.: *Bora: Routing and aggregation for distributed processing of spatio-temporal range queries*. In *Proceedings of the IEEE International Conference on Mobile Data Management (MDM)*. 2007, pp. 36–43

[337] Trasarti, R.; Guidotti, R.; Monreale, A.; Giannotti, F.: *MyWay: Location prediction via mobility profiling*. In *Information Systems* 64, 2017: pp. 350–367

[338] Treiber, M.; Kesting, A.: *Traffic Flow Dynamics*. In *Traffic Flow Dynamics: Data, Models and Simulation* 2013: pp. 983–1000

[339] Tsubouchi, Y.; Wakisaka, A.; Hamada, K.; Matsuki, M.; Abe, H.; Matsumoto, R.: *HeteroTSDB: An Extensible Time Series Database for Automatically Tiering on Heterogeneous Key-Value Stores*. In *Proceedings of the IEEE Annual Computer Software and Applications Conference (COMPSAC)*. 2019, Volume 1, pp. 264–269

[340] Ungureanu, C.; Debnath, B.; Rago, S.; Aranya, A.: *TBF: A memory-efficient replacement policy for flash-based caches*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2013, pp. 1117–1128

[341] Valdés, F.; Güting, R. H.: *A framework for efficient multi-attribute movement data analysis*. In *VLDB Journal* 28(4), 2019: pp. 427–449

[342] Valentin, G.; Zuliani, M.; Zilio, D. C.; Lohman, G. M.; Skelley, A.: *DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2000, pp. 101–110

[343] Van Aken, D.; Yang, D.; Brillard, S.; Fiorino, A.; Zhang, B.; Bilien, C.; Pavlo, A.: *An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems*. In *Proceedings of the VLDB Endowment* 14(7), 2021: pp. 1241–1253

[344] Vanderbei, R. J.: *Linear Programming – Foundations and Extensions*. Springer, 2014

[345] Vermeij, M.; Quak, W.; Kersten, M.; Nes, N.: *MonetDB, A Novel Spatial Column-Store DBMS*. In *Proceedings of the Free and Open Source for Geospatial (FOSS4G) Conference*. 2008, pp. 193–199

[346] Vieira, M. R.; Bakalov, P.; Tsotras, V. J.: *Querying Trajectories Using Flexible Patterns*. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2010, pp. 406–417

[347] Vlachos, M.; Kollios, G.; Gunopulos, D.: *Discovering Similar Multidimensional Trajectories*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2002, pp. 673–684

[348] Vogel, L.; Leis, V.; van Renen, A.; Neumann, T.; Imamura, S.; Kemper, A.: *Mosaic: a budget-conscious storage engine for relational database systems*. In *Proceedings of the VLDB Endowment* 13(12), 2020: pp. 2662–2675

[349] Wang, D.; Miwa, T.; Morikawa, T.: *Big trajectory data mining: a survey of methods, applications, and services*. In *Sensors* 20(16), 2020: p. 4571

[350] Wang, H.; Su, H.; Zheng, K.; Sadiq, S.; Zhou, X.: *An Effectiveness Study on Trajectory Similarity Measures*. In *Proceedings of the Australasian Database Conference (ADC)*. 2013, pp. 13–22

[351] Wang, H.; Varman, P.: *Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-Aware Allocation*. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. 2014, pp. 229–242

[352] Wang, H.; Zheng, K.; Jeung, H.; Bracher, S.; Islam, A.; Sadiq, W.; Sadiq, S.; Zhou, X.: *Storing and Processing Massive Trajectory Data on SAP HANA*. In *Proceedings of the Australasian Database Conference (ADC)*. 2015, pp. 66–77

[353] Wang, H.; Zheng, K.; Xu, J.; Zheng, B.; Zhou, X.; Sadiq, S.: *SharkDB: An In-Memory Column-Oriented Trajectory Storage*. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*. 2014, pp. 1409–1418

[354] Wang, J.; Trummer, I.; Basu, D.: *UDO: Universal Database Optimization Using Reinforcement Learning*. In *Proceedings of the VLDB Endowment* 14(13), 2021: p. 34023414

[355] Wang, S.; Bao, Z.; Culpepper, J. S.; Cong, G.: *A survey on trajectory data management, analytics, and learning*. In *ACM Computing Surveys (CSUR)* 54(2), 2021: pp. 1–36

[356] Wang, S.; Cao, J.; Yu, P. S.: *Deep Learning for Spatio-Temporal Data Mining: A Survey*. In *IEEE Transactions on Knowledge and Data Engineering* 34(8), 2022: pp. 3681–3700

[357] Wang, X.; Ma, Y.; Di, J.; Murphey, Y. L.; Qiu, S.; Kristinsson, J.; Meyer, J.; Tseng, F.; Feldkamp, T.: *Building Efficient Probability Transition Matrix Using Machine Learning from Big Data for Personalized Route Prediction*. In *Proceedings of the INNS Conference on Big Data*. 2015, pp. 284–291

[358] Wang, Y.; Zheng, Y.; Xue, Y.: *Travel Time Estimation of a Path Using Sparse Trajectories*. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2014, pp. 25–34

[359] Wang, Y.; Zhu, Y.; He, Z.; Yue, Y.; Li, Q.: *Challenges and Opportunities in Exploiting Large-Scale GPS Probe Data*. In *HP Laboratories, Technical Report HPL-2011-109* 2011: pp. 1–10

[360] Wang, Z.; Long, C.; Cong, G.: *Trajectory Simplification with Reinforcement Learning*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2021, pp. 684–695

[361] Weisgut, M.: *Experimental Index Evaluation for Partial Indexes in Horizontally Partitioned In-Memory Databases*. In *Proceedings of the GI-Workshop Grundlagen von Datenbanken (GvDB)*. 2021

[362] Westmann, T.; Kossmann, D.; Helmer, S.; Moerkotte, G.: *The implementation and performance of compressed databases*. In *ACM Sigmod Record* 29(3), 2000: pp. 55–67

[363] Willhalm, T.; Popovici, N.; Boshmaf, Y.; Plattner, H.; Zeier, A.; Schaffner, J.: *SIMD-scan: ultra fast in-memory table scan using on-chip vector processing units*. In *Proceedings of the VLDB Endowment* 2(1), 2009: pp. 385–394

[364] Wu, E.; Madden, S.: *Partitioning Techniques for Fine-Grained Indexing*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2011, pp. 1127–1138

[365] Xiao, Z.; Wang, Y.; Fu, K.; Wu, F.: *Identifying Different Transportation Modes from Trajectory Data Using Tree-Based Ensemble Classifiers*. In *ISPRS International Journal of Geo-Information* 6(2), 2017: p. 57

[366] Xie, D.; Li, F.; Yao, B.; Li, G.; Zhou, L.; Guo, M.: *Simba: Efficient in-memory spatial analytics*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2016, pp. 1071–1085

[367] Xie, X.; Mei, B.; Chen, J.; Du, X.; Jensen, C. S.: *Elite: an elastic infrastructure for big spatiotemporal trajectories*. In *VLDB Journal* 25(4), 2016: pp. 473–493

[368] Xie, Z.; Wang, H.; Wu, L.: *The improved Douglas-Peucker algorithm based on the contour character*. In *Proceedings of the International Conference on Geoinformatics*. 2011, pp. 1–5

[369] Xu, Z.; Li, Z.; Guan, Q.; Zhang, D.; Li, Q.; Nan, J.; Liu, C.; Bian, W.; Ye, J.: *Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach*. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 905–913

[370] Yang, B.; Guo, C.; Ma, Y.; Jensen, C. S.: *Toward personalized, context-aware routing*. In *VLDB Journal* 24(2), 2015: pp. 297–318

[371] Ye, N.; Wang, Z. Q.; Malekian, R.; Lin, Q.; Wang, R. C.: *A Method for Driving Route Predictions Based on Hidden Markov Model*. In *Mathematical Problems in Engineering* 2015: pp. 1–12

[372] You, S.; Zhang, J.; Gruenwald, L.: *Spatial Join Query Processing in Cloud: Analyzing Design Choices and Performance Comparisons*. In *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW)*. 2015, pp. 90–97

[373] Yu, J.; Wu, J.; Sarwat, M.: *Geospark: A cluster computing framework for processing large-scale spatial data*. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2015, pp. 70:1–70:4

[374] Yuan, J.; Zheng, Y.; Xie, X.: *Discovering regions of different functions in a city using human mobility and POIs*. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2012, pp. 186–194

[375] Yuan, N. J.; Zheng, Y.; Xie, X.; Wang, Y.; Zheng, K.; Xiong, H.: *Discovering urban functional zones using latent activity trajectories*. In

IEEE Transactions on Knowledge and Data Engineering 27(3), 2014: pp. 712–725

[376] Yue, Z.; Zhang, J.; Zhang, H.; Yang, Q.: *Time-Based Trajectory Data Partitioning for Efficient Range Query*. In *Proceedings of the DASFAA International Workshop on Big Data Management ad Service (BDMS)*. 2018, pp. 24–35

[377] Zacharatou, E. T.: *Efficient Query Processing for Spatial and Temporal Data Exploration*. Dissertation, Ecole Polytechnique Fédérale de Lausanne, 2019

[378] Zaharia, M.; Xin, R. S.; Wendell, P.; Das, T.; Armbrust, M.; Dave, A.; Meng, X.; Rosen, J.; Venkataraman, S.; Franklin, M. J.; et al.: *Apache spark: a unified engine for big data processing*. In *Communications of the ACM* 59(11), 2016: pp. 56–65

[379] Zhang, D.; Ding, M.; Yang, D.; Liu, Y.; Fan, J.; Shen, H. T.: *Trajectory simplification: an experimental study and quality analysis*. In *Proceedings of the VLDB Endowment* 11, 2018: pp. 934–946

[380] Zhang, H.; Andersen, D. G.; Pavlo, A.; Kaminsky, M.; Ma, L.; Shen, R.: *Reducing the Storage Overhead of Main-Memory OLTP Databases with Hybrid Indexes*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2016, pp. 1567–1581

[381] Zhang, H.; Chen, G.; Ooi, B. C.; Tan, K.-L.; Zhang, M.: *In-memory big data management and processing: A survey*. In *IEEE Transactions on Knowledge and Data Engineering* 27(7), 2015: pp. 1920–1948

[382] Zhang, J.; Liu, Y.; Zhou, K.; Li, G.; Xiao, Z.; Cheng, B.; Xing, J.; Wang, Y.; Cheng, T.; Liu, L.; et al.: *An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning*. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 2019, pp. 415–432

[383] Zhang, L.; Yi, J.: *Management Methods of Spatial Data Based on PostGIS*. In *Proceedings of the Pacific-Asia Conference on Circuits, Communications and System*. 2010, pp. 410–413

[384] Zhang, N.; Zheng, G.; Chen, H.; Chen, J.; Chen, X.: *HBaseSpatial: A Scalable Spatial Data Storage Based on HBase*. In *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2014, pp. 644–651

[385] Zhang, R.; Xie, P.; Wang, C.; Liu, G.; Wan, S.: *Classifying Transportation Mode and Speed from Trajectory Data via Deep Multi-Scale Learning*. In *Computer Networks* 162(C), 2019: p. 106861

[386] Zhang, X.; Chang, Z.; Li, Y.; Wu, H.; Tan, J.; Li, F.; Cui, B.: *Facilitating Database Tuning with Hyper-Parameter Optimization: A Comprehensive Experimental Evaluation*. In *Proceedings of the VLDB Endowment* 15(9), 2022: pp. 1808–1821

[387] Zhang, X.; Li, W.; Zhang, F.; Liu, R.; Du, Z.: *Identifying Urban Functional Zones Using Public Bicycle Rental Records and Point-of-Interest Data*. In *ISPRS International Journal of Geo-Information* 7(12), 2018: pp. 459–474

[388] Zhang, X.; Xie, L.; Wang, Z.; Zhou, J.: *Boosted Trajectory Calibration for Traffic State Estimation*. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. 2019, pp. 866–875

[389] Zhang, Z.; Huang, K.; Tan, T.: *Comparison of Similarity Measures for Trajectory Clustering in Outdoor Surveillance Scenes*. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*. 2006, pp. 1135–1138

[390] Zhang, Z.; Jin, C.; Mao, J.; Yang, X.; Zhou, A.: *TrajSpark: A Scalable and Efficient In-Memory Management System for Big Trajectory Data*. In *Proceedings of the International Joint Conference on Web and Big Data (APWeb-WAIM)*. 2017, pp. 11–26

[391] Zhao, L.; Jin, P.; Zhang, L.; Wang, H.; Lin, S.: *Developing an Oracle-Based Spatio-Temporal Information Management System*. In *Proceedings of the International Conference on Database Systems for Adanced Applications (DASFAA)*. 2011, pp. 168–176

[392] Zhao, Y.; Shang, S.; Wang, Y.; Zheng, B.; Nguyen, Q. V. H.; Zheng, K.: *REST: A Reference-based Framework for Spatio-temporal Trajectory Compression*. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 2797–2806

[393] Zheng, B.; Wang, H.; Zheng, K.; Su, H.; Liu, K.; Shang, S.: *Sharkdb: An In-Memory Column-Oriented Storage for Trajectory Analysis*. In *World Wide Web* 21(2), 2018: pp. 455–485

[394] Zheng, K.; Zheng, Y.; Xie, X.; Zhou, X.: *Reducing Uncertainty of Low-Sampling-Rate Trajectories*. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 2012, pp. 1144–1155

[395] Zheng, Y.: *Trajectory data mining: an overview*. In *ACM Transactions on Intelligent Systems and Technology (TIST)* 6(3), 2015: pp. 1–41

[396] Zheng, Y.; Zhou, X.: *Computing with spatial trajectories*. Springer Science & Business Media, 2011

[397] Zhou, J.; Tung, A. K.; Wu, W.; Ng, W. S.: *A "semi-lazy" approach to probabilistic path prediction*. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2013, pp. 748–756

[398] Zhou, X.; Chai, C.; Li, G.; Sun, J.: *Database Meets Artificial Intelligence: A Survey*. In *IEEE Transactions on Knowledge and Data Engineering* 34(3), 2020: pp. 1096–1116

[399] Zhu, L.; Holden, J. R.; Gonder, J. D.: *Trajectory Segmentation Map-Matching Approach for Large-Scale, High-Resolution GPS Data*. In *Transportation Research Record* 2645(1), 2017: pp. 67–75

[400] Ziebart, B. D.; Maas, A. L.; Bagnell, J. A.; Dey, A. K.: *Maximum Entropy Inverse Reinforcement Learning*. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*. 2008, pp. 1433–1438

[401] Ziebart, B. D.; Maas, A. L.; Dey, A. K.; Bagnell, J. A.: *Navigate Like a Cabbie: Probabilistic Reasoning from Observed Context-Aware Behavior*. In *Proceedings of the ACM International Conference on Ubiquitous Computing (UbiComp)*. 2008, pp. 322–331

[402] Zilio, D. C.; Rao, J.; Lightstone, S.; Lohman, G.; Storm, A.; Garcia-Arellano, C.; Fadden, S.: *DB2 Design Advisor: Integrated Automatic Physical Database Design*. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. 2004, pp. 1087–1097

[403] Zimányi, E.; Sakr, M. A.; Lesuisse, A.: *MobilityDB: A Mobility Database Based on PostgreSQL and PostGIS*. In *ACM Transactions on Database Systems* 45(4), 2020: pp. 19:1–19:42

[404] Zukowski, M.; Boncz, P. A.; Nes, N.; Héman, S.: *MonetDB/X100-A DBMS in the CPU Cache*. In *IEEE Data Eng. Bull.* 28(2), 2005: pp. 17–22

[405] Zyner, A.; Worrall, S.; Ward, J. R.; Nebot, E. M.: *Long Short Term Memory for Driver Intent Prediction*. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 1484–1489

Eigenständigkeitserklärung
Declaration of Authorship

Hiermit versichere ich an Eides statt, dass die vorliegende Arbeit bisher an keiner anderen Hochschule eingereicht worden ist sowie selbständig und ausschließlich mit den angegebenen Mitteln angefertigt worden ist. Die Stellen der Arbeit, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, sind durch Angaben und Quellen kenntlich gemacht.

Potsdam, 31. März 2024

Keven Richly