Proceedings of the Fall 2010 Future SOC Lab Day

Christoph Meinel, Andreas Polze, Alexander Zeier,
Gerhard Oswald, Dieter Herzog, Volker Smid,
Doc D'Errico, Zahid Hussain (Hrsg.)

Technische Berichte Nr. 42

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 42

Christoph Meinel | Andreas Polze | Alexander Zeier | Gerhard Oswald
Dieter Herzog | Volker Smid | Doc D'Errico | Zahid Hussain (Hrsg.)

Proceedings of the Fall 2010 Future SOC Lab Day

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de/ abrufbar.

Universitätsverlag Potsdam 2011
http://info.ub.uni-potsdam.de/verlag.htm

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 4623 / Fax: 3474
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2011/4976/
URN urn:nbn:de:kobv:517-opus-49761
http://nbn-resolving.org/urn:nbn:de:kobv:517-opus-49761

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-114-1

Contents

Simulating the Internet of Services at HPI Future SOC Lab

Stefan König
University of Bayreuth
Chair of Information Systems Management
95440 Bayreuth
stefan.koenig@uni-bayreuth.de

Torsten Eymann
University of Bayreuth
Chair of Information Systems Management
95440 Bayreuth
torsten.eymann@uni-bayreuth.de

Abstract

The Internet-of-Services (IoS) describes a general computational paradigm, which allows companies to procure computational resources externally and thus to save both internal capital expenditures and operational costs. Due to the decentralized and open nature of the IoS common access control mechanisms ("hard security") are not sufficient. Additionally, socio-economic mechanisms like controlled markets in combination with reputation mechanisms should be implemented. This project focuses on large scale simulation experiments to substantiate former findings.

1. Project Idea

In the proposed project we will investigate to what extend socio-economical mechanisms can help to close the gap that traditional access control mechanisms leave open. In particular, we want to conduct simulations to assess the effect of different reputation tracking mechanisms; e.g. the detection of fraudsters in the market population in terms of speed, rate and adaptability; different options to calculate the risk of fraud and to rank transaction partners accordingly; the overall utility for the market as well as the individual utility for the single participant. In a more abstract view on the problem, we will thus try to answer the question how much of the ex-ante concepts of traditional security can be substituted or enhanced by using dynamic and runtime socio-economic control [1].
Due to the visionary nature of the IoS [8], we need a simulation environment that allows us to test different mechanisms in future IoS settings. Managed by the University of Bayreuth, the simulation toolset SimIS (Simulating the Internet of Services: http://sourceforge.net/projects/simis/), which allows to simulate socio-economical mechanisms in these Future Internet settings, has been developed. Preliminary results show that the combination of negotiation protocols with established reputation mechanisms promise a suitable usage control of these open and distributed systems during runtime.

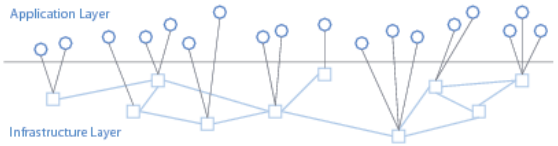


Figure 1: SimIS Architecture [3]

1.1 Simulation Environment

In order to evaluate the socio-economic mechanisms later, we have to introduce a simulation environment, called SimIS [3], that follows the IoS vision. This system is able to model Internet-like networks where the nodes are hosting active services. The messages follow the SOAP messages structure and the service interfaces follow real-world Web Services technology.

1.1.1 Technological Base: Repast Toolkit

The SimIS[1] toolkit was implemented as an extension to the Recursive Porous Agent Simulation Toolkit [7], developed at the Argonne National Lab, Chicago. Repast is a free and open source agent-based modelling toolkit [4]. This foundation was chosen due to its comprehensive API, the very generic and easy to use set of data gathering and analysis functions as well as the support for network modelling (including respective programming libraries). Technically, the current version of SimIS is based on Repast Symphony and is completely implemented in the Java programming language.

1.1.2 SimIS Architecture

In order to map the abstract IoS architecture to our simulation model a two-tiered architecture for SimIS seems suitable. The overall system is thus divided into an Application Layer and an Infrastructure Layer. An overview of the overall architecture is illustrated in Figure 1.
The *Infrastructure Layer* models topological settings of the IoS. The basic idea is that all Application Layer

[1] For more information see http://simis.sourceforge.net

Agents or Services are linked to a single Infrastructure Agent each, which is representing their server platform. This platform is therefore responsible for sending messages to other Application Layer Agents (including routing and communication patterns, such as broad- or multicast), and receiving messages from other Infrastructure Agents and passing them on to either other Infrastructure Agents (in case the agent represents only the next step on the message's route) or to one or more Application Layer Agents associated with it (in case these are the recipients) [3].
Within the *Application Layer* the actual services of the IoS vision are modelled. Basically the underlying Infrastructure Layer provides us with a high-enough flexibility for implementing any service logic in terms of Application Layer Agents communicating via the offered message objects and routing functionality. Each service (Application Layer Agent) is implemented as a plain Java class and can therefore exploit the full potential this programming language offers in addition to the libraries present within the SimIS toolkit.

2 Used Future SOC Lab Resources

Before joining the Future SOC Lab Consortium, we conducted simulation experiments using graphical interfaces. For the project we are going to extend the simulation environment to enable a complete command line control. This phase still lasts for some time. Nevertheless we have started to replicate some experiments (see section 3 for details). The methodological approach of replicating agent-based simulation model is based on the general framework of Edmonds and Hales [2], see figure 2, but considers also more detailed issues of Sansores and Pavón [9].
In order to replicate the simulation experiments, we worked on a virtual machine that has been provided by the HPI Future SOC Lab. The machine has 8 GB RAM, 2-Core CPU and 40 GB HDD available. In order to replicate the simulation experiment the machine was busy for about eight hours.

3 Findings

This section first presents the simulation scenario that is the same for the original simulation experiments and the replicated experiments. Following, the metrics are defined before the simulation results of the original experiments are presented.

3.1 Simulation Scenario

As just mentioned, the topology will be divided into an Infrastructure Layer consisting of nodes and edges between them, and an Application Layer. The network used for simulation experiments consists of 100 nodes that are connected not heavy-weighted and not long-tailed. The mean distance between the nodes is about 3.26 with an maximum distance between two nodes of six hops.
For the following simulation experiments, 200 Service Provider (SP) agents and 200 Service Consumer (SC) agents will be deployed at the beginning of the simulation experiment. In order to introduce dynamics, participants are substituted by newcomers during the simulation experiments. The time range for the substitution process is set to a value that in average the complete population of agents is replaced once during one simulation experiment of 100,000 time ticks. Depending on the payment model 10% SP cheaters or 10% SC cheaters are deployed. This rate might fluctuate due to the dynamic character of the system.
For each simulation setting, the products that are negotiated are fixed by a certain functional attribute definition. This attribute combination is assumed to be defined by an underlying service description. The one and only attribute that is negotiated is denoted by the price. The negotiation outcome is determined by a (M+1)st price Double Auction [10].

3.2 Metrics

As the simulation scenario has been defined, the metrics for the simulation experiments are introduced now. The fulfilment rate and the negotiation rate are plotted in dependence of the simulation time.
The concrete implementation of the fulfilment depends on the payment model of the simulation experiment: if the service has to be paid in advance, the fulfilment rate considers the service fulfilment (see equation 1). For each participant the rate of successful services against failed services are noted. Without any reputation system, one would expect that the service fulfilment rate corresponds to the rate of cheaters in the system. If the service has to be paid after it has been fulfilled, the fulfilment rate considers the payment fulfilment. Then, for each SP the rate of successful payments against failed payments are plotted over time.

$$fr = \sum_{i}^{|SCs|} \frac{services_i^{received}}{services_i^{paid}} \tag{1}$$

3.3 Simulation Results

In the following simulations each single experiment is repeated for 10 times using different Random valuations and makes use of the modified Double Auction Protocol. During the data analysis the mean of the time series is taken for further analysis. The simulation outcome is further compared to the case that uses the initial Double Auction protocol. In both cases, $AVALANCHE_{dec}$ is used as decentralized trust and reputation model for all participants (SPs and SCs). $AVALANCHE_{dec}$ is an extended approach

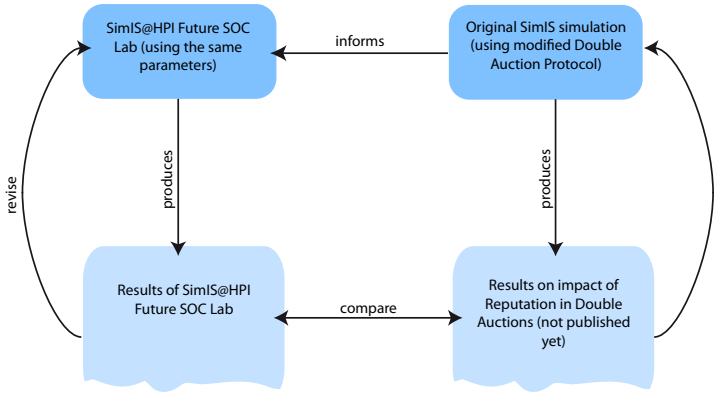


Figure 2: Relationship between published model and re-implementation (following [2])

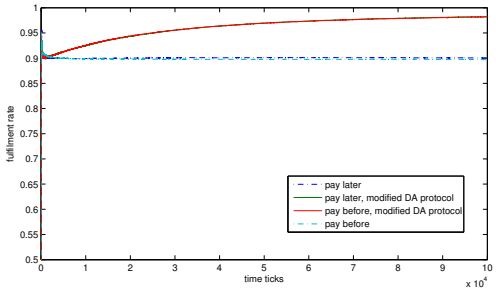


Figure 3: Double Auction Fulfilment Rate

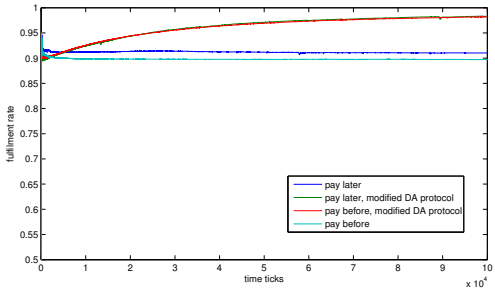


Figure 4: Double Auction Fulfilment Rate

of $AVALANCHE$, a reputation tracking mechanism for Multi-Agent Systems [5, 6]. $AVALANCHE_{dec}$ is identical to the original approach with one exception: the reputation value are not stored within a centralized reputation unit. Instead, all intelligent services (agents) are responsible for managing their own experiences and providing this information to requesting agents. This change becomes necessary due to the IoS requirements of avoiding central instances to keep this decentralized system running.

Figure 3 illustrates the overall fulfilment rate of the scenario's simulation experiments. The figure denotes the payment fulfilment in the pay-later and the service fulfilment in the pay-before model. As we have 10% of cheating agents, the fulfilment rate is expected at about 90%. As we have seen above, the reference case (initial Double Auction protocol) exactly fulfils the expectation with a constant fulfilment rate of about 90% (dotted lines).

The continuous lines illustrate the service or payment fulfilment rates when using $AVALANCHE_{dec}$ combined with the modified Double Auction protocol. The service fulfilment rate increases for about eight percent points compared to the simulation outcome with the initial Double Auction protocol. Both suitable payment models (pay-before or pay-later) lead to analogous results.

A value of approximated 100% is unrealistic due to the following reasons: during the settlement phase of the simulation run the reputation system has to be filled with information. Within this settlement phase some interactions fail, such that the rate can not reach the 100% value. Further, during the simulation runs the implemented dynamics lead to a continuous arrival of unknown agents.

3.4 Replication

The same experiments have been conducted on the HPI Future SOC Lab infrastructure. Even if the duration of each single simulation experiment decreases, the simulation outcome can be assumed as identical. Figure 4 illustrates the corresponding fulfilment rate. This finding is important for the future steps of this project. Only with ensuring the possibility to replicate simulation experiments, it is possible to conclude new findings from the next steps for the whole context.

4 Next Steps

The next step of the project "SimIS@HPI Future SOC Lab" are can be stated as follows:

- The results presented above already show that the usage of reputation mechanisms in a Double Auction market seems to be promising. Nevertheless the amount of services has been quite low in these simulations due to resource limitations of simulation environments. The corresponding research interest is two-fold: First, the scalability of the SimIS environment: is it possible to simulate IoS environments with 1000 and more services? If so, the second question regards the emergent behaviour of the system: does it differ to the already simulated small environment in their trend?

- In a second step the influence of the network structure on the simulation outcomes might be interesting. What about reputation information spreading in heavy-tailed or clustered networks? How will changes in the network structure affect the simulation outcomes?

References

[1] R. AlNemr, S. König, T. Eymann, and C. Meinel. Enabling usage control through reputation objects: A discussion on e-commerce and the internet of services environments. *Journal of Theoretical and Applied E-Commerce Research*, 5(2):59–76, 2010. to appear.

[2] B. Edmonds and D. Hales. Replication, replication and replication: Some hard lessons from model alignment. *Journal of Artificial Societies and Social Simulation*, 6(4), 2003.

[3] S. König, S. Hudert, and T. Eymann. Socio-Economic Mechanisms to Coordinate the Internet of Services - the Simulation Environment SimIS. *Journal of Artificial Societies and Social Simulation (JASSS)*, 13(2), 2010.

[4] M. J. North, N. T. Collier, and J. R. Vos. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation*, 16:1–25, 2006.

[5] B. Padovan, S. Sackmann, and T. Eymann. Secure Electronic Marketplaces Based on the Multi Agent System Avalanche. *SSRN eLibrary*, 2000.

[6] B. Padovan, S. Sackmann, T. Eymann, and I. Pippow. A prototype for an agent-based secure electronic marketplace including reputation tracking mechanisms. *International Journal of Electronic Commerce*, 6(4):93–113, 2002.

[7] Repast Development Group. Repast Home Page. Website. http://repast.sourceforge.net.

[8] R. Ruggaber. Internet of services sap research vision. In *WETICE '07: Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, page 3, Washington, DC, USA, 2007. IEEE Computer Society.

[9] C. Sansores and J. Pavón. Agent-based simulation replication: A model driven architecture approach. In *Proceedings of the 4th Mexican international conference on artificial intelligence (MICAI), Monterrey, Mexico*, Lecture Notes on Computer Science, pages 244–253. Springer, 2005.

[10] P. R. Wurman, W. E. Walsh, and M. P. Wellman. Flexible double auctions for electronic commerce: theory and implementation. *Decision Support Systems*, 24:17–24, 1998.

A protocol-generic Infrastructure for electronic SLA Negotiations in the Internet of Services

Sebastian Hudert
University of Bayreuth
Chair of Information Systems Management
95440 Bayreuth
sebastian.hudert@uni-bayreuth.de

Torsten Eymann
University of Bayreuth
Chair of Information Systems Management
95440 Bayreuth
torsten.eymann@uni-bayreuth.de

Abstract

Visions of the next-generation Internet of Services (IoS) are driven by digital resources traded on a global scope. For the resulting economic setting, automated on-line techniques for handling services and resources themselves, for advertising and discovering as well as for the on-the-fly negotiation of proper terms for their use are needed. Hence, a flexible infrastructure for the respective handling of services and associated service level agreements (SLAs) is mandatory.
In the course of this project, we want to design such an infrastructure, building on software-agent technology and an expressive but still machine manageable protocol description language capable of specifying a multitude of different negotiation protocols. It will support the discovery of services with appropriate SLA negotiation styles as well as the actual SLA negotiation based on chosen protocol description documents.

1. Project Idea and Motivation

Current developments in the area of Information Systems show a tendency towards massively distributed infrastructures, consisting of highly specialized digital resources. Today's Internet of mainly human interactions will evolve towards a socio-technical and global information infrastructure, where humans as well as software agents, acting on their behalf, continuously interact to exchange data and computational resources. This vision of globally interacting electronic services can be observed in both research and industry alike and is commonly referred to as the IoS [15, 14].
Building on currently applied computing paradigms, such as Service-oriented [3], Grid [4] or Cloud Computing [2], the IoS vision defines highly dynamic networks of composable services, offered and consumed on demand and on a global scope. Taking such ideas one step further, it rigorously focuses on the goal of an Internet-based service economy similar to the real-world service sector. Digital services will be offered over electronic service markets, purchased by respective customers and then combined with internal or other external services to business workflows of varying complexity.

Hence, the IoS will primarily focus on new business models and the commercial application of distributed computing, concerning trading processes down to the level of an individual service, and the subsequent charging based on its usage and delivered quality-of-service (QoS).

Two of the main challenges for the IoS, from a commercial perspective, are reliability of the services traded and the technical infrastructure underlying the service economy. Since such scenarios inherently lack the applicability of centralized QoS management, service guarantees must be obtained in the form of bilateral SLAs assuring service quality across individual sites [9]. These SLAs subsequently act as a signed contract governing the actual service invocation [2], enabling the structured monitoring and assessment of the service's compliance.

A very crucial part of the SLA-based service life cycle can be seen in the discovery and above all the negotiation phase. All subsequent steps (binding, execution and monitoring, post-processing etc.) depend on the SLA documents which were agreed-upon in this phase. On that account we focus on the negotiation and prior discovery of SLAs for our work presented in this project.

Aiming at this very phase, economic reserach claims that differences in system configuration, or the services actually traded, demand different negotiation protocols in order to reach the highest-possible efficiency of the overall system (see for example [10]). Based on these findings and the global context of the envisioned scenario it is not likely, or even efficient, that only one central marketplace for electronic services will emerge, offering a single, known protocol. Instead a system of marketplaces offering different protocols will probably emerge, each of which is best be suited for a given context.

Fortifying this, we argue that restricting service consumers (SCs) or providers (SPs) in that they are only able to interact with one distinct service market they were implemented for (and are therefore only technically compatible with the applied negotiation protocol), unnecessarily decreases the potential flexibility and efficiency of the whole system. SCs should be able to buy, and therefore negotiate about, any fitting service, regardless of the market it is offered in, and thus regardless of the protocol with which it is offered. Also, given the dynamic nature of distributed workflow executions and the increased complexity of global service selection manual negotiations of the human users are by far not efficient enough. This process should be automated by electronic software agents that negotiate on the users' behalf.
The research goal of our work is thus to develop a service-oriented infrastructure supporting software agents to discover and negotiate about electronic SLAs and not restricting them to a pre-defined negotiation protocol.

2 Research Method

Due to the nature of the problem considered a constructivist research process was chosen for this project, following the principles of Design Science (DS) in Information Systems (IS). According to Hevner et al., DS represents a research paradigm originating in engineering, basically aiming at the generation of innovative solutions to non-trivial problems [6].
The problem considered in this project,is the lack of mechanisms for a comprehensive and fully automated management of SLAs in distributed service systems. For this problem domain a new and innovative infrastructure definition is to be derived.
Several research groups have come up with frameworks and idealized process models for DS efforts (e. g. [6] or [13]). All are basically geared to the abstract engineering cycle of problem identification, problem solving and evaluation of the solution. Based on these process models the research approach applied in this project was defined.
The first step is concerned with the definition of a given research problem and the justification of the solution's value. This step regularly involves a detailed scenario definition in order to give a clue on the state of the research problem and to motivate a solution to be developed. Based on a description of the anticipated scenario the problem to be addressed and the resulting research goals is detailed and motivated.
In a second step desired objectives of the envisioned solution are derived from the presented scenario model and the research goals, eventually resulting in a set of defined requirements. The requirements for this project are identified in section 3 based on findings in scientific literature. These requirements either both act as input for the subsequent Design and Development phase, in that they set the boundaries within which design decisions can be made, as well as for the Demonstration and Evaluation phase, since they represent the criteria against which the designed solution will be evaluated.
There are two distinct approaches for evaluating our infrastructure: Its implementation in the context of a physically distributed service management middleware and subsequent usage or its integration into an IoS simulation toolkit and subsequent simulative assessment. Within the current project the latter approach will be used in order to investigate the effectiveness and efficiency of the developed mechanisms before porting it to a productive environment in a second step.

3 Requirements

In the following requirements for such an infrastructure definition, as they were found in the literature, will be presented and structured semantically.

General requirements:

- Mechanisms are needed for dynamic discovery and negotiation of electronic services (i.e. [2]).
- SLAs should be employed for service description and QoS assurance (i.e. [2] or [12]).

Requirements, structured according to the service management process:

Discovery Phase:

- After the Discovery Phase all parties must have a common understanding of the protocol to be executed in the negotiation phase [8].
- This common understanding must be generated dynamically at runtime [1].

Negotiation Phase:
Negotiation Object:

- Services (and thus SLAs) of different complexity must be negotiable (i.e. [10]).
- Possible offers should be restrictable, incl. non-negotiable SLA terms (i.e. [8]).

Negotiation Protocol / Setting:

- Different marketplaces and protocols even within one market are needed for dif-ferent services to be traded (i.e. [10] or [2]).
- Service requestors and consumers must be able to start the negotiation (i.e. [16]).

Negotiation Strategy / Participants:

- Software Agents should act as negotiators (i.e. [12] or [8]).
- Intermediaries, such as auctioneers or brokers, should be present (i.e. [2]).

4 Simulation Environment

In order to evaluate the developed infrastructure concepts, we developed a simulation environment, called SimIS [7], that follows the IoS vision. This system is able to model Internet-like networks where the nodes are hosting active services. The messages follow the SOAP message structure and the service interfaces follow real-world Web Services technology.

4.1 Technological Base: Repast Toolkit

The SimIS[1] toolkit was implemented as an extension to the Recursive Porous Agent Simulation Toolkit [5], developed at the Argonne National Lab, Chicago. Repast is a free and open source agent-based modelling toolkit [11]. This foundation was chosen due to its comprehensive API, the very generic and easy to use set of data gathering and analysis functions as well as the support for network modelling (including respective programming libraries). Technically, the current version of SimIS is based on Repast Symphony and is completely implemented in the Java programming language.

4.2 SimIS Architecture

In order to map the abstract IoS architecture to our simulation model a two-tiered architecture for SimIS seems suitable. The overall system is thus divided into an Application Layer and an Infrastructure Layer. An overview of the overall architecture is illustrated in Figure 1.

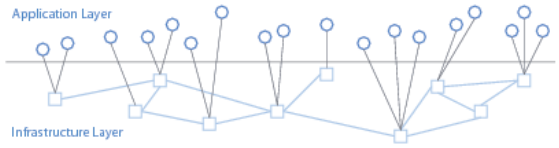


Figure 1: SimIS Architecture [7]

The *Infrastructure Layer* models topological settings of the IoS. The basic idea is that all Application Layer Agents or Services are linked to a single Infrastructure Agent each, which is representing their server platform. This platform is therefore responsible for sending messages to other Application Layer Agents (including routing and communication patterns, such as broad- or multicast), and receiving messages from other Infrastructure Agents and passing them on to either other Infrastructure Agents or to one or more Application Layer Agents associated with it.

[1]For more information see http://simis.sourceforge.net

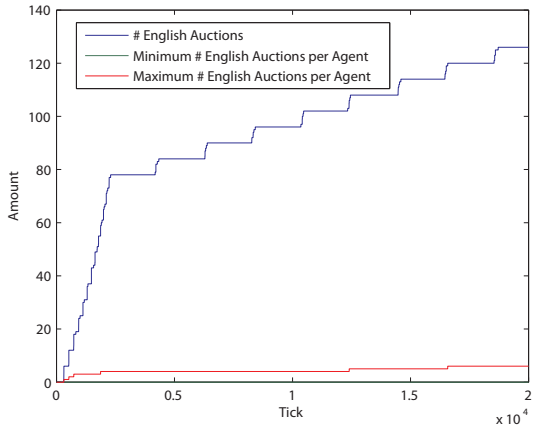


Figure 2: Initial Simulation Results

Within the *Application Layer* the actual services of the IoS vision are modelled. Basically the underlying Infrastructure Layer provides us with a high-enough flexibility for implementing any service logic in terms of Application Layer Agents communicating via the offered message objects and routing functionality. Each service (Application Layer Agent) is implemented as a plain Java class and can therefore exploit the full potential this programming language offers.

5 Used Future SOC Lab Resources

At the current state the developed software pieces represent a proof-of-concept prototype which is currently work in progress. Very initial simulation runs were already undertaken on a simple desktop computer, the results of which can be seen in figure 2.

For these experiments, the simulator was configured with ten infrastructure nodes, upon which 40 SPs and 50 SCs are placed. Each SP offers the same service, however 8 of those with the Alternate Offers, 12 with the English Auction and the remaining 20 with the Double Auction protocol. Additionally one registry node for storing and retrieving the service and protocol description documents as well as one broker for the Double Auction are present.

At this point no resources from the HPI Future SOC Lab were used, as the mechanisms and data structures are currently developed. The prototype system is currently in a very early feasibility test phase. Once these initial tests are successful the whole system will be ported to the HPI Future SOC infrastructure for extensive evaluation runs. These will include simulations of very extreme market configurations and different timeout and communication settings in order to investigate the boundaries of our approach.

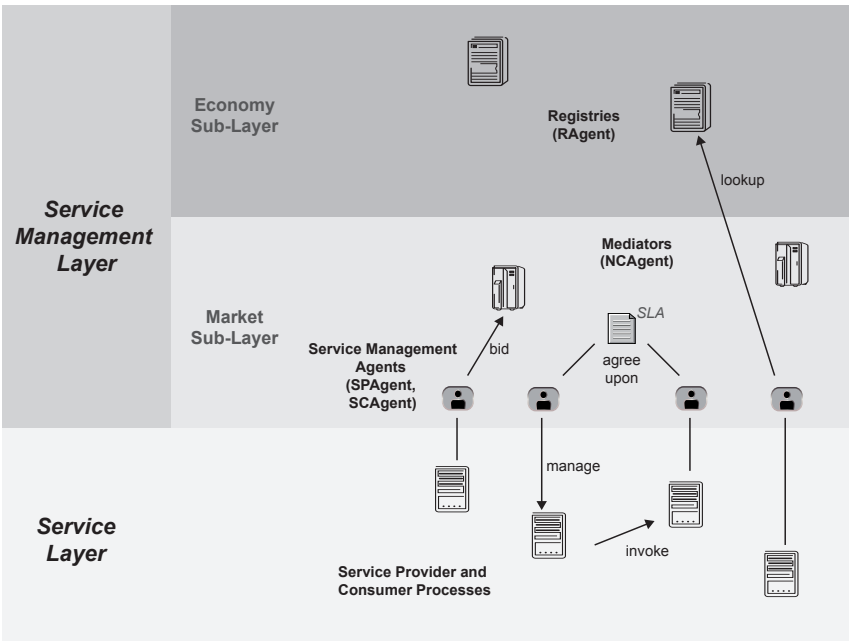


Figure 3: System Architecture

6 Findings

The major deliverables achieved up to now are a comprehensive functional architecture of our infrastructure, structured schema descriptions of the involved data types as well as initial implementations of the designed software components. All of these will shortly be sketched in the following.

6.1 Design Idea

The basic design idea underlying this work is to offer a given product (SLA for an electronic service) independently from the way an agreement concerning this product can be attained (negotiation protocol). This allows for flexible combinations of product and protocol to be chosen for each market situation individually.
Since software agents will be employed for the service management within our systen, the negotiation protocol applied for a given service is not only decoupled from the actual SLA but must also be made explicit in terms of its communication rules. This allows for run time adaption of the SC agents to the respective protocol.
To this end, the designed infrastructure will build on a conceptual architecture of machine-readable description documents, as described in the following.

6.2 Service Description Documents

As just introduced, a set of service description documents is needed enabling a) the discovery of an appropriate service or respective SLA (template) and b) the description of the negotiation protocol used to reach an agreement. For that purpose three different data structures were designed:

- Service Type (ST): definition of the functional and non-functional aspects, a given class of services can offer.
- Extended SLA Template (EST): definition of initial QoS guarantees (building on the non-functional aspects given in the respective ST) as an input for the subsequent negotiation as well as the applied negotiation protocol.
- Service Identificator (SI): identification of an individual service instance along with links to the associated ST and EST documents.

Figure 4 gives a short overview on these documents and their relations:

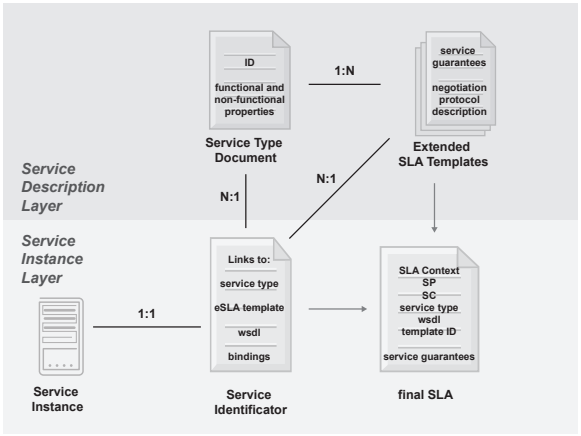


Figure 4: Document-based Architecture

6.3 Role-based Architecture

The developed prototype infrastructure builds on a defined set of roles, the service management agents can adopt: The two basic roles present in this system are the SC and the SP, representing buyer and seller agents.
Additionally a set of registry services / agents (RA) are needed for supporting the publication and discovery of service description documents. Finally, the NC role represents the agents mediating negotiation processes as a broker agent.
Both SP and SC agents mainly offer an interface to one another, allowing them to send and receive messages related to the discovery and negotiation phases. Additionally, each of theses roles offers one routine to external users of the infrastructure: A SP agent offers a method for publishing and selling and a SC one for discovering and purchasing a service on a user's behalf, respectively.
A RA only accepts discovery related messages as it does not take part in any other phases of the life cycle. Finally, NC agents are responsible for admission of SCs or SPs to and potentially mediation of a given negotiation. Hence, they again offer methods for exchanging respective messages.

7 Next Steps

The next step of our project proceed in two basic directions:

- Porting the SimIS toolkit to the HPI Future Soc resources in order to allow the large-scale evaluation of our mechanisms.
- In a second step to further develop our internal components and, once scalable and effective, port them to a real-world service management middleware.

References

[1] I. Brandic, S. Venugopal, M. Mattess, and R. Buyya. Towards a meta-negotiation architecture for sla-aware grid services. Techreport, University of Melbourne, August 2008.

[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[3] I. Foster. Service-oriented science. *Science*, 308(5723):814–817, 2005.

[4] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15:2001, 2001.

[5] R. D. Group. Repast home page. Website.

[6] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[7] S. König, S. Hudert, and T. Eymann. Socio-economic mechanisms to coordinate the internet of services – the simulation environment simis. *Journal of Artificial Societies and Social Simulation (JASSS)*, 13(2), 2010.

[8] A. Ludwig, P. Braun, R. Kowalczyk, and B. Franczyk. A framework for automated negotiation of service level agreements in services grids. In *Lecture Notes in Computer Science, Proceedings of the Workshop on Web Service Choreography and Orchestration for Business Process Management, 2006*, volume 3812/2006, 2006.

[9] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. A service level agreement language for dynamic electronic services. *Journal of Electronic Commerce Research*, 3:43–59, 2003.

[10] D. Neumann, J. Stoesser, C. Weinhardt, and J. Nimis. A framework for commercial grids - economic and technical challenges. *Journal of Grid Computing*, 6(3):325–347, September 2008. ISSN: 1570-7873.

[11] M. J. North, N. T. Collier, and J. R. Vos. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation*, 16:1–25, 2006.

[12] S. Paurobally, V. Tamma, and M. Wooldridge. A framework for web service negotiation. *ACM Trans. Auton. Adapt. Syst.*, 2(4):14, 2007.

[13] K. Peffers, T. Tuunanen, M. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2008.

[14] R. Ruggaber. Internet of services sap research vision. In *WETICE '07: Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, page 3, Washington, DC, USA, 2007. IEEE Computer Society.

[15] C. Schroth and T. Janner. Web 2.0 and SOA: Converging concepts enabling the internet of services. *IT Professional*, 9(3):36–41, 2007.

[16] W. Ziegler, O. Waeldrich, P. Wieder, T. Nakata, and M. Parkin. Considerations for negotiation and monitoring of service level agreements. Technical Report TR-0167, CoreGRID, June 2008.

Simulation of Physical Growth Models

Tobias Friedrich
Max-Planck-Institut für Informatik
Campus E1.4
66123 Saarbrücken, Germany

Abstract

Based on recent advances in random walk theory we have developed a new highly-efficient parallel algorithm to simulate several physical growth models. We use the infrastructure of the HPI Future SOC Lab to examine properties of these physical growth models up to four orders of magnitude larger than previously possible. Our main application is internal diffusion limited aggregation which models for example solid melting around a heat source. Obtaining precise estimates for the resulting shape is a long-standing open problem in statistical physics. We are also interested in how much the amount of randomness in the simulation influences the process.

1. Introduction

How do corals grow? Successive particles wander in "from infinity" and stick when they reach some arm of the coral. The resulting growths appear dendritic and fractal-like, but rigorous results are extremely hard to come by. This model is called *diffusion limited aggregation (DLA)* and was introduced in 1981 by Witten and Sander [21, 22] to model aggregates of condensing metal vapor. It has since then attracted much interest both among mathematicians and physicists. It models not only the growth of corals, snowflakes, vascular networks, neurons, and crystals, but also the path taken by a lightning, coalescing of dust or smoke particles, dielectric breakdown, electrochemical deposition, viscous fingering, watershed formation. Figure 1 gives a typical example of two-dimensional diffusion limited aggregation (Thanks to Paul Bourke from the Centre for Astrophysics and Supercomputing, Swinburne University for providing the image).

The cousin of DLA is *internal diffusion-limited aggregation (IDLA)*. This is a cluster growth process in which particles start at one or more sources within a cluster, diffuse outward, and are added to the cluster at the first site outside it they reach. This process is sometimes also called anti-DLA or diffusion-limited erosion as by reversing figure and ground, one can see this as a hole being hollowed out by particles which remove sites from a surrounding material. It has been introduced in 1990 by Diaconis and Fulton [7] as a variant of classical DLA. IDLA models electrochemical polishing, viscous fluid displacing an inviscid one in a porous medium, solid melting around a heat source, and other physical phenomena.

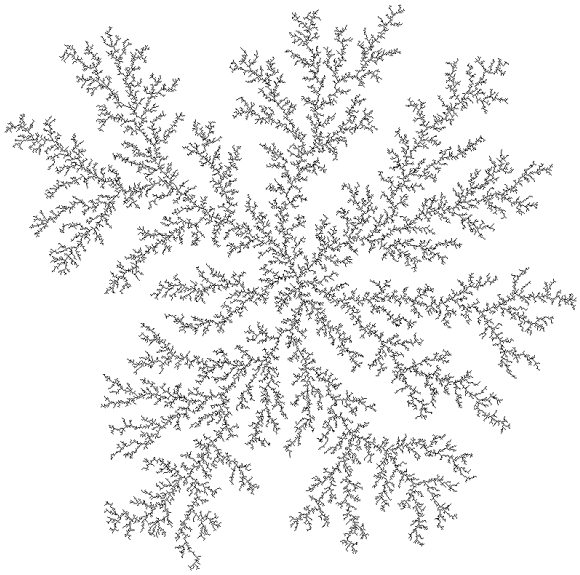


Figure 1: Example for diffusion limited aggregation (DLA).

Though DLA is a very popular model, rigorous results on its shape remain very hard to prove as it tends to amplify irregularities in the cluster's boundary. On the other hand, IDLA tends to smooth them out. Lawler, Bramson, and Griffeath [16] showed that the limiting shape of n particles is a Euclidean ball as $n \to \infty$ and that with high probability the fluctuations around this limiting shape are bounded by $O(n^{1/3})$ [15]. Moore and Machta [20] observed experimentally that up to $n = 10^{5.25}$ these error terms were even smaller, namely of roughly logarithmic size. Determining the size of the boundary fluctuations in IDLA is a long-standing open problem in statistical physics. Our aim is understanding this physical process better by simulating it in a clever way up to four orders of magnitude further than previous methods.

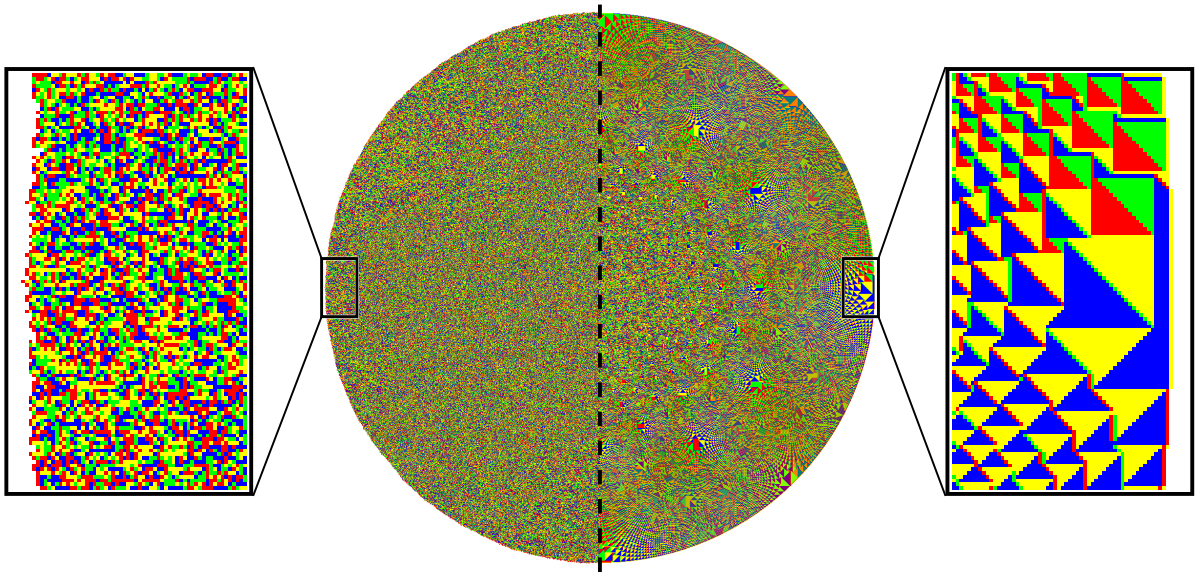


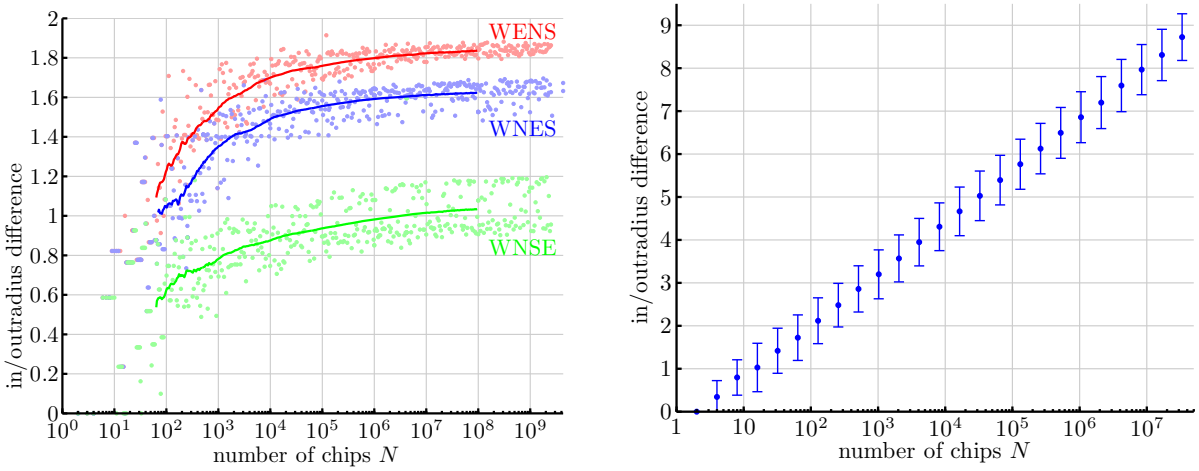
Figure 2: IDLA cluster (left) and rotor-router aggregation with counterclockwise rotor sequence (right) of $N = 10^6$ chips. Half of each aggregate is shown. Each site is colored according to the final direction of the rotor on top of its stack (yellow=`W`, red=`S`, blue=`E`, green=`N`). Note that the boundary fluctuations of the rotor-router aggregation are much smoother than for IDLA. Larger rotor-router aggregates of size up to $N = 10^{10}$ can be found on [1].

2 Derandomization and Quasirandomness

With the Copenhagen interpretation of quantum mechanics by Bohr and Heisenberg in the early 20-th century, it became widely accepted that the fundamental understanding of the world can only be statistical. However, it remains widely open *how much* randomness is needed. This is not only a fundamental question in philosophy and physics, but also a very practical one in computer science.

Coming back to the aforementioned growth models, we are interested in how much the resulting structure depends on the amount of involved randomness. In the late 1990s, James Propp from the University of Massachusetts Lowell came up with a quasirandom aggregation as a way of derandomizing IDLA [14]. To do so, he adds a rotor to each position and this rotor can point up, right, down, or left. Now the particles do not anymore go into a random direction, but in the direction of the respective rotor and afterwards the rotor is rotated one step further (e.g. clockwise or counterclockwise). This way the previously random walk of the particles becomes a *quasirandom walk*, which is deterministically determined after the initial directions of the rotors have been fixed. Hence the property that the neighboring positions are served with equal chance is kept while the variance of the process is removed. It is known from combinatorics that this quasirandom walk closely resembles a random walk in several respects [4–6, 8, 13]. The concept of quasirandom walks has lead to improvements in algorithmic applications. Examples include external mergesort [3], broadcasting information in networks [9], and iterative load-balancing [12].

Starting with all initial rotors pointing in one direction, the quasirandom IDLA reveals very interesting patterns. The right half of Figure 2 shows a quasirandom IDLA of one million chips. Every site's rotor begins pointing east and is rotated counterclockwise. The color indicates in which direction the rotor is eventually pointing. The clearly visible fractal-like patterns are barely understood, but it is known that the limiting shape is also a Euclidean ball and that the inner fluctuations are provably logarithmic in the radius [17, 18] in this case. However, the true fluctuations appear to grow even more slowly, and may even be bounded independent of the number of particles. Kleber [14] observed that for up to $n = 3 \cdot 10^6$ particles the difference of the radius of the largest inscribed and the smallest circumscribed circle is at most a small constant. With better algorithmic tools and the resources of the Future SOC lab we could recently confirm these measurement up to $n \approx 2 \cdot 10^{10}$ particles. Another motivation is to generate more fine-scaled examples of the intricate patterns of the final rotors at the end of the aggregation process. As said above, these patterns remain poorly understood even on a heuristic level. A four-color picture of the final rotors of an aggregation of $n = 10^{10}$ particles can be found on our website [1]. To make this picture with over ten Gigapixels accessible, this page uses a Google Maps overlay to allow the user to zoom and scroll through the image.



(a) Quasirandom IDLA. **(b)** IDLA.

Figure 3: Difference between inradius and outradius for different numbers of chips N for quasirandom IDLA (depending on the rotor sequence) and the classic IDLA (with standard deviations). It is clearly visible that the fluctuations for classic IDLA are of logarithmic order while they are sublogarithmic for the quasirandom variant.

3 Algorithmic Techniques

It is known that computing DLA is P-complete and therefore inherently sequential [19]. We now discuss how to calculate a classic IDLA or quasirandom IDLA cluster of n particles. Straight-forward sequential simulation of IDLA takes on average $\Theta(n^2)$ steps as every particle needs roughly $\Theta(n)$ steps to go from its source to the boundary at distance $\Theta(\sqrt{n})$. Moore and Machta [20] showed that on a parallel machine with k processors this can be reduced to $O(n^2/k + n \log k)$ steps. In [11] we developed a substantially faster sequential algorithm based on recent findings in random walk theory. With this we achieve sequential runtimes of about $\Theta(n^{1.5})$ for classic IDLA and $\Theta(n \operatorname{polylog}(n))$ for quasirandom IDLA.

The main trick of the new algorithm is to compute the final state of various growth models without computing all intermediate states. For this, we analyze the odometer function of the growth process which counts for every position how often a particle has passed. Our technique is based on a least action principle [10] which characterizes this function. Starting from an educated guess for the odometer, we successively correct under- and overestimates and *provably* arrive at the correct final state. The degree of speedup depends on the accuracy of the initial guess. As this successively correcting the under- and overestimates can be done independently in different regions, this algorithm scales very well on many-core architectures.

4 Experimental Setup

After initial experiments with the machines available at the Max-Planck-Institut für Informatik in Saarbrücken, we used the resources of the HPI Future SOC Lab to perform more involved experiments. As especially the runtime for quasirandom IDLA is very close to linear, indeed memory, rather than time, is the limiting factor. Therefore especially the usage of the Hewlett Packard DL980 G7 with 2048 GB main memory greatly enhances the applicability of this method. Figure 3 (a) gives some results on the fluctuations of quasirandom IDLA depending on the number of chips/particles.

Because of the larger fluctuations, the simulation of classic IDLA takes significantly longer and has to be repeated many times to get statistically significant estimates for size and fluctuations. For this, we have used some of the HPI Future SOC Lab resources with many cores, but not necessarily large main memory. Figure 3 (b) shows some results for the classic IDLA model.

5 Outlook

Combining recent results from random walk theory and recent hardware from the HPI Future SOC Lab, we have analyzed several physical growth models and gained statistical data of up to four magnitudes higher quality than previously possible. As next step we want to analyze higher moments of classic IDLA and other quasirandom variants of IDLA. We also hope to extend this analysis to other types of abelian distributed processors, for example to the well-known Bak-Tang-Wiesenfeld abelian sandpile model [2].

References

[1] `http://rotor-router.mpi-inf.mpg.de`.

[2] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: An explanation of the $1/f$ noise. *Physical Review Letters*, 59(4):381–384, 1987.

[3] R. D. Barve, E. F. Grove, and J. S. Vitter. Simple randomized mergesort on parallel disks. *Parallel Computing*, 23(4-5):601–631, 1997.

[4] J. Cooper and J. Spencer. Simulating a random walk with constant error. *Combinatorics, Probability & Computing*, 15:815–822, 2006.

[5] J. Cooper, B. Doerr, T. Friedrich, and J. Spencer. Deterministic random walks on regular trees. *Random Structures and Algorithms*. To appear, preliminary version appeared in *19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 766–772, 2008.

[6] J. Cooper, B. Doerr, J. Spencer, and G. Tardos. Deterministic random walks on the integers. *European Journal of Combinatorics*, 28: 2072–2090, 2007.

[7] P. Diaconis and W. Fulton. A growth model, a game, an algebra, Lagrange inversion, and characteristic classes. *Rend. Sem. Mat. Univ. Pol. Torino*, 49:95–119, 1990.

[8] B. Doerr and T. Friedrich. Deterministic random walks on the two-dimensional grid. *Combinatorics, Probability & Computing*, 18(1-2):123–144, 2009.

[9] B. Doerr, T. Friedrich, and T. Sauerwald. Quasirandom rumor spreading. In *19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 773–781, 2008.

[10] A. Fey, L. Levine, and Y. Peres. Growth rates and explosions in sandpiles. *Journal of Statistical Physics*, 2010. To appear, arXiv:0901.3805.

[11] T. Friedrich and L. Levine. Fast simulation of large-scale growth models, 2010. arXiv:1006.1003.

[12] T. Friedrich, M. Gairing, and T. Sauerwald. Quasirandom load balancing. In *21th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 1620–1629, 2010.

[13] A. E. Holroyd and J. Propp. Rotor walks and Markov chains, 2009. arXiv:0904.4507.

[14] M. Kleber. Goldbug Variations. *Mathematical Intelligencer*, 27(1):55–63, 2005.

[15] G. F. Lawler. Subdiffusive fluctuations for internal diffusion limited aggregation. *Annals of Probability*, 23(1):71–86, 1995.

[16] G. F. Lawler, M. Bramson, and D. Griffeath. Internal diffusion limited aggregation. *Annals of Probability*, 20:2117–2140, 1992.

[17] L. Levine and Y. Peres. Spherical asymptotics for the rotor-router model in $\mathbb{Z}^d$. *Indiana Univ. Math. J.*, 57(1):431–450, 2008.

[18] L. Levine and Y. Peres. Strong spherical asymptotics for rotor-router aggregation and the divisible sandpile. *Potential Analysis*, 30:1–27, 2009.

[19] J. Machta and R. Greenlaw. The parallel complexity of growth models. *Journal of Statistical Physics*, 77:755, 1994.

[20] C. Moore and J. Machta. Internal diffusion-limited aggregation: Parallel algorithms and complexity. *Journal of Statistical Physics*, 99(3–4):661–690, 2000.

[21] T. A. Witten and L. M. Sander. Diffusion-limited aggregation, a kinetic critical phenomenon. *Physical Review Letters*, 47(19): 1400–1403, 1981.

[22] T. A. Witten and L. M. Sander. Diffusion-limited aggregation. *Physical Review B*, 27(9):5686–5697, 1983.

Towards Scalable and Self-Optimizing Software for Multi-Core and Cloud Computing

Sebastian Wätzoldt, Stephan Hildebrandt, Andreas Seibel, Gregor Gabrysiak and Holger Giese
System Analysis and Modeling at Hasso Plattner Institute for IT Systems Engineering
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
sebastian.waetzoldt@student.hpi.uni-potsdam.de
[stephan.hildebrandt|andreas.seibel|gregor.gabrysiak|holger.giese]@hpi.uni-potsdam.de

Abstract

Developing software that really exploits the potential of multi-core and cloud computing is inherently more difficult than traditional software development as the essentially required development of parallel behavior is much more difficult than for sequential behavior. However, multi-core and cloud computing is more about scalability of service-oriented architectures than only performance as in case of parallel computing and instead of a given fixed hardware configuration the possibility to exchange the underling hardware or provider to handle even higher loads is key. We propose to approach these new challenges by a model-driven approach where the higher-level abstraction of the software description enables to derive several optimized platform-specific solutions for different as well as changing hardware settings. In order to ensure that the system operates always with a good solution, the software should be able to adapt itself such that in the spirit of autonomic computing the software takes care of the permanent self-optimization of its execution strategies to ensure scalability. To evaluate different initial static options for our related currently developed self-adaptive model-driven approach, we employed the HPI Future SOC lab as a test bed.

1. Introduction

Moore's law today still holds concerning the increase in number of gates we can integrate on a chip and it will probably still hold for a few generations of chips before we hit the atom size as limiting factor. However, the related speedup for sequential processing we could also observe in the last decades has already come to a halt. The increase in number of gates is today used to for *multi-core computing* [2] that integrate multiple cores on one CPU rather than to speedup the sequential processing of a single program. In addition, virtualization and massive parallelization of computational tasks using *cloud computing* [1] has become popular. Here server farms running many standard PCs rather than dedicated high-performance computers with special hardware are employed to achieve required computations in a cost efficient manner.

On the one hand, both trends promise to speed up the computing by doing it in parallel. On the other hand, developing software that really exploits the potential of multi-core and cloud computing is inherently much more difficult than traditional software development due to the need to do parallel processing. At first, today's development languages have been optimized for sequential processing on single-core systems. In addition, decades of research in the field of parallel computing have shown that generic, easy to program, and platform independent solutions usually result in severe performance penalties.

But the raised challenge is not the same as parallel computing. We see two characteristics that distinguish developing software for multi-core and cloud computing from parallel computing: (1) the former is more about scalability of service-oriented architectures than only performance as from a business perceptive the option to add resource to serve more customers and not optimal performance is what counts. (2) In the parallel computing view the goal is to optimize the software for a given fixed hardware configuration and a given task, while in the considered case the possibility to handle unpredictable loads and the option for the provider to exchange the underling hardware is key.

We propose to approach these new challenges by a model-driven approach where the higher-level abstraction of the software description enables to derive several optimized platform-specific solutions for different as well as changing hardware settings. In order to ensure that the system operates always with a good solution, the software should be able to adapt itself such that in the spirit of autonomic computing [6] the software takes care of the permanent self-optimization of its execution strategies to ensure scalability.

We at first want to the support *control flow parallelism* given either explicitly in the specification or implicitly by detecting it by analyzing the specification. In addition, we also exploit *data parallelism* when the

specified behavior is executed on large data sets. Here, highly abstract specification of the data queries and data manipulation only implicitly permit to parallelize their execution.

To achieve this goal, a high-level modeling approach for the software and its deployment, a suitable runtime environment to support the monitoring as well as adaptation, and alternative execution strategies as well as selection strategies for them have to be developed. To evaluate different static options in a first initial step for our related currently developed self-adaptive model-driven approach, we employed the HPI Future SOC lab as a test bed in a series of tests that only started recently.

To report on the project, we at first explain the employed high-level specification language in Section 2. Then, the studied parallelization strategies are presented in Section 3. In Section 4, the preliminary evaluation of the strategies is presented and final conclusions and an outlook on planned additional work closes the report.

2. High-Level Specification Language

Story diagrams are an established high-level specification language for the definition of behavior in diverse contexts. They have been employed in several applications ranging from behavior specification [3] of software systems, reverse engineering [7], consistency checking [8], and as implementation technique for model transformations with triple graph grammars [5].

Story diagrams are basically a combination of UML activity diagrams and UML collaboration diagrams. The activities of a story diagram are called story pattern. A story diagram defines the control flow between story patterns. A story pattern is interpreted as a graph transformation and can be used for data queries and manipulation. The notation of story pattern is to some extent similar to UML collaboration diagrams. In the following, the control flow of story diagrams and story patterns are explained in more detail.

The content of a story pattern is basically a single graph query and graph manipulation rule (called graph rewriting rule) suitable for systems, whose structure has been defined by UML class diagrams. The story pattern defines one or more instance graphs that can be matched on a given host graph (a specific instance situation). The story patterns contain a left-hand side (LHS) and a right-hand side (RHS). The LHS defines the graph query that has to be found in the current host graph. The resulting graph manipulation is defined as follows: All elements only included in the LHS but not the RHS are deleted and all elements present in the RHS but not in the LHS are created.

Figure 1 shows an exemplary story pattern A. The LHS contains the instances x:X, y:Y, z:Z, t:T and g:G. Modifications to the instance graphs are defined by additional annotations to the instances, which are - - for

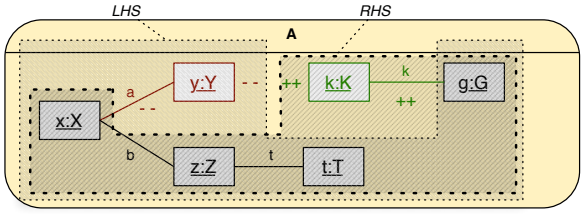


Figure 1. Exemplary story pattern

deleting an instance or a reference and ++ for creating an instance or a reference. All elements that have no or a ++ annotation are part of the RHS. All elements that have no annotation are part of the LHS and RHS. y:Y, for example, is part of the LHS only because it has to be matched before it can deleted. When deleting a node, all references pointing to this node are implicitly deleted too.[1]

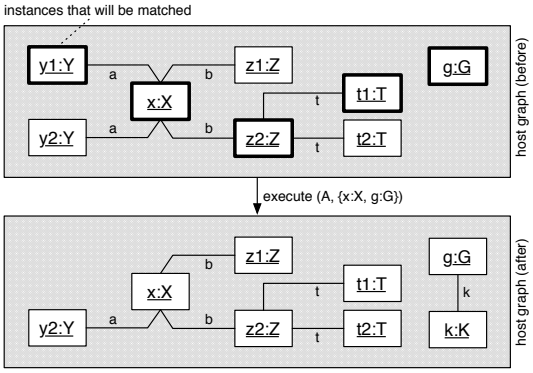


Figure 2. Executing the story pattern

Figure 2 depicts the execution of the story pattern A on a specific host graph, which is shown at the top of Figure 2. First, the interpreter will find a match for the story pattern on this host graph (executing the LHS), which can be y1:Y, x:X, z2:Z, t1:T and g:G. Second, the match in the host graph is modified according to the RHS, which results in deleting y1:Y and creating k:K that is connected to g:G via reference k.

If the story pattern A would be defined as being a foreach story pattern, all matches that can be found in the host graph would be modified according to the RHS. When executing a story pattern, at least one instance of the current instance graph has to be already bound. In Figure 1, x:x and g:G are declared to be already bound.

The control flow of a story diagram is pretty similar to a control flow in UML activity diagrams. The elements of a story diagram are start nodes, decision nodes, story patterns for the regular activities, fork nodes, join nodes, join specifications, final nodes and edges connecting these nodes. Figure 3 shows an exemplary story diagrams to give an overview about the graphical notation.

[1] In terms of graph transformation systems this behavior of implicitly deleting references is called single push-out (SPO).

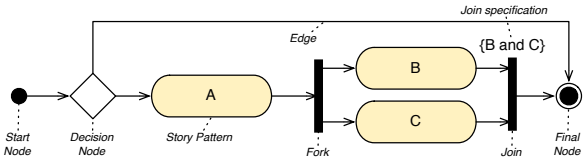


Figure 3. Exemplary story diagram

Executing a story diagram always starts from a start node and terminates whenever a final node is reached. This may include parallel processing in case of fork and join nodes. Edges between story patterns transport the resulting bindings from one story pattern to another. Thus, executing the example story diagram would start in the start node and then deciding whether continue executing story pattern A or directly terminate the execution by executing the final node. After activity A has been executed, the activities B and C have to be executed in parallel. These two nodes could be executed sequentially or even in parallel, denoted by the fork and join nodes. After the join, the execution terminates.

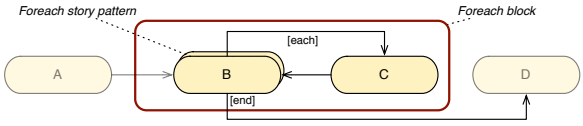


Figure 4. Exemplary foreach block

An important concept of story diagrams are foreach loops. The loop condition of a foreach loop is a single story pattern, whose query determines for which bindings the loop is executed. The body is given by the set of activities that are reached via an [each] edge and subsequent edges. The body is then executed for each binding determined by the loop condition. Figure 4 shows a foreach loop whose body contains a single story pattern C. The loop condition of the foreach loop is given by a foreach story pattern B. The annotated edge [each] indicates the start of the foreach loop. The other edge [end] denotes the termination of the foreach loop.

Each valid result found for the query specified by the loop condition in form of a foreach story pattern initiates the execution of the whole foreach body (taking the [each] edge). When all valid query results of story pattern B have been processed, the edge [end] is traversed. It is to be noted that in principle the execution of the loop bodies for different query results could happen in parallel.

3. Parallelization Strategies

In the previous section, we explained the basic story diagrams and its operational semantics. The current story diagram interpreter [4] only supported a sequential execution of story diagrams and thus did not support fork and join nodes. Therefore, we developed a new interpreter that supports explicitly defined concurrency (fork and join nodes; see Figure 3), exploits that foreach loops allow a concurrent execution and also parallelize the query processing. In the following, we will briefly explain the parallelization capabilities of the newly developed interpreter for story diagrams and story pattern.

The new story diagram interpreter introduces the concept of a scheduler for threads and a task pool as primary concepts for parallelization. Tasks are used to capture possible next steps in the execution of story diagrams and story patterns. In the context of executing a complete story diagrams, a pool of current tasks and a pool of worker threads is used to dynamically assign task to threads by a scheduler.

The scheduler manages a fixed number of threads, which are currently dynamically created depending on the given platform.[2] The interpreter provides a pool of tasks to the scheduler, which should be concurrently executed. Thus, the scheduler also manages a pool of tasks, which have to be mapped on threads for execution.

Tasks that are waiting for execution are passive tasks whereas tasks that are currently executed by a thread are active tasks. Any task can have dependencies to other tasks. Thus, the scheduler has to adhere to these dependencies to ensure a proper execution.

An important capability of tasks is that during their execution by a thread other tasks may be created and added to the task pool. Each thread executing a task has read access to a global variables context managed by a variables context manager. Additionally, it has read/write access to a local variables context.

Basically, the control flow of story diagrams is a proper sequence of story pattern. Nevertheless as UML activity diagrams, join nodes and fork nodes can be employed to explicitly define concurrency in the control flow.[3] In addition, the UML permits that synchronization of the control flow at join nodes can be explicitly specified by means of join specifications, e.g., (B and C) in Figure 3 will force to stop the execution of the story diagram until B and C have finished their execution (see Figure 5.

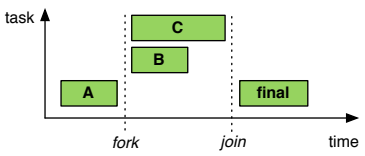


Figure 5. Execution of the fork/join

[2] Currently we create two threads per physical CPU core.

[3] Usually, a modeler will manually specify fork and join nodes. However, also an automated analysis could be envisioned that automatically add fork and join nodes to maximize concurrency in a given story diagram. Such an automated analysis would require a detailed analysis of individual story pattern in the control flow concerning possible data dependencies.

Story Diagrams additionally provide semaphores to provide a more sophisticated synchronization concept. UML activity diagrams had a similar construct called sync states that were skipped with the transition from UML 1.4 to UML 2.0.

In the old interpreter foreach nodes and foreach loops were executed sequentially. In the new interpreter we can execute them in parallel exploiting implicit control flow and data parallelism. Each foreach story pattern is related to a task. When a thread executes this task, it creates subtasks for each valid query result. In case of a foreach loop, the subtask initiates the execution of the story patterns in the loop body. When all subtasks have been executed, the parent task synchronizes the results of its subtasks and terminates (see Figure 6).

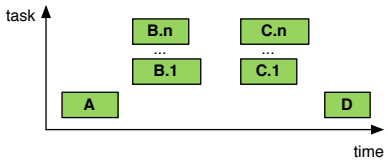


Figure 6. Execution of the foreach loop

Not only the control flow of story diagrams is subject to parallelization, but also the query processing for a story pattern. Tasks are used to separate the query processing of a story pattern, so that each task defines a part of a processing that can be executed in parallel by other tasks. The interpreter first analyzes if there are parts of the query that are completely independent. These are assigned to different tasks. Each task can itself be split again into several additional tasks. The top of Figure 7 shows how a single task has been split into two tasks.

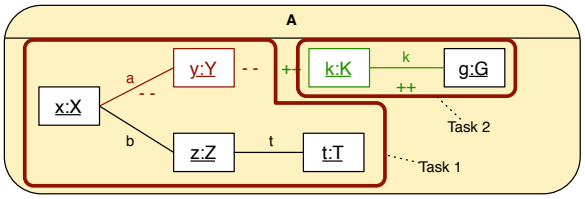


Figure 7. Split a task into two tasks

4. Preliminary Evaluation

We only recently started with the evaluation of the outlined parallelization strategies. In this report, we can only report on first, small experiments done to study the performance for the parallelization strategies for concurrency explicitly specified via fork and join nodes as well as implicit concurrency for foreach loops.

As test platforms we use our sequential interpreter running on a 1,87 GHz core, a dual core with 2,4 GHz (windows 7), a quad core with 2,5 GHz (windows server 2003) and the future soc lab with 24 cores (with hyper threading 48) with 1,87 GHz. Therefore, a slightly better speedup could be expected for the dual and quad core as the sequential processing is faster.

As test scenarios we consider different simple cases for story diagrams that work on data sets of different complexity. It holds that E1, E2 and E3 increase in complexity. The same applies for I1, I2 and I3.

A simple case for explicit parallelism resulting from a combination of fork and join nodes is depicted in Figure 8. The story pattern permits to parallelize the queries, which is not always the case. The details of the story pattern are omitted to meet the space constraints we have for the report.

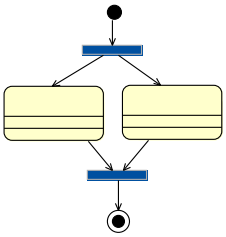


Figure 8. Story diagram with explicit parallelism via fork/join nodes

The results of running the story diagram for the three data sets are depicted in Figure 9. The time required by the sequential interpretation is used to normalize the results. The presented speedup factor is the time required for the sequential case divided by the time required on the specific hardware platform. Therefore, the sequential processing always has a speedup equal to one.

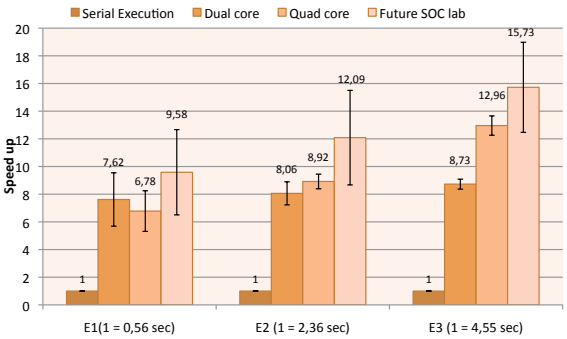


Figure 9. Results for explicit parallelism

The results presented in Figure 9 for the explicit concurrency due to fork/join nodes indicate that at first the new interpreter with its parallel processing has general performance benefits as superlinear improvements for the dual core in particular cannot result only from the higher chip frequency. In addition, we can observe that speedup for the 24 cores is only moderate. Here only the effects of parallelizing the matching matter and the

two parallel story pattern are equally well exploited already on a dual core machine.

The case of a foreach loop studied in our first experiments is shown in Figure 10. Also in this case the story pattern permits to parallelize the queries. We omit the details of the story pattern due to the limited space we have for the report.

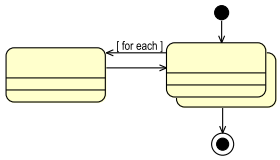


Figure 10. Story diagram with implicit parallelism of a foreach loop

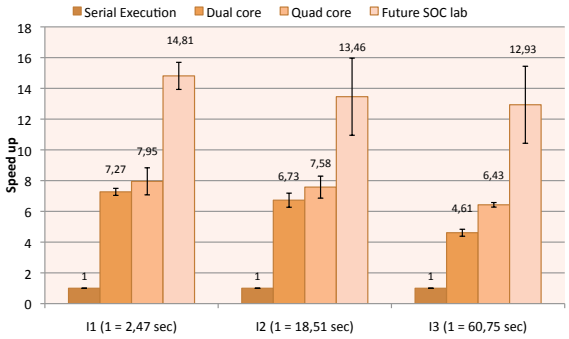


Figure 11. Results for implicit parallelism

The achieved speedup for the three test data sets are reported in Figure 11. Here also the data indicates that at first the new interpreter with its parallel processing has general performance benefits due to the superlinear improvements for the dual core. We can further observe that for more complex examples for the given test data the speedup for the 24 cores is not increasing. This is not what we expected and we hope to identify the source of this weak elasticity in further experiments.

5. Conclusions and Future Work

Currently, we can only show that there is high potential for major performance improvements by means of parallelizing the execution of story diagrams and story patterns. However, there are also cases where the overhead of parallelization in fact can result in a decrease of the performance.

Thus, we need to further investigate the characteristics for those cases where the parallel execution really outperforms the sequential execution. If we could characterize and detect these cases via heuristics, this would enable hybrid execution strategies for story diagrams and story patterns that only employ the parallel execution when major performance improvements are highly likely. In addition, the decision could be done context-dependent depending on the fact whether it is economically useful to speedup the execution of the story diagram at hand or not (e.g., if a certain response time is desirable and the current processing time indicates that it may be missed, more parallelization to achieve a speedup would make sense).

In the last month of the project it is planned to more thoroughly study the parallelization strategies for different data and therefore establish a foundation to identify the characteristics that may guide the heuristics. In the planned follow-up project we cooperate with SAP to ensure a more industrial perspective. We plan to mainly address the question which language constructs are relevant for SAP application domain and will only partially cover the needed runtime systems as well as self-adaptation. The project will be supported also by a joint bachelor project entitled ”Model-Driven Software Development for Multicore and Cloud Systems”. In later projects we then plan to also address the run-time systems and in particular self-adaptation in more depth.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009.

[2] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[3] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story Diagrams: A new Graph Rewrite Language based on the Unified Modeling Language. In G. Engels and G. Rozenberg, editors, *Proc. of the 6th International Workshop on Theory and Application of Graph Transformation (TAGT), Paderborn, Germany*, LNCS 1764, pages 296–309. Springer Verlag, 1998.

[4] H. Giese, S. Hildebrandt, and A. Seibel. Improved Flexibility and Scalability by Interpreting Story Diagrams. In T. Magaria, J. Padberg, and G. Taentzer, editors, *Proceedings of the Eighth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2009)*, volume 18. Electronic Communications of the EASST, 2009.

[5] H. Giese and R. Wagner. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling (SoSyM)*, 8(1), 28 March 2009.

[6] J. O. Kephart and D. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003.

[7] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh. Towards Pattern-Based Design Recovery. In *Proc. of the 24th International Conference on Software Engineering (ICSE), Orlando, Florida, USA*, pages 338–348. ACM Press, May 2002.

[8] R. Wagner, H. Giese, and U. Nickel. A Plug-In for Flexible and Incremental Consistency Management. In *Proc. of the International Conference on the Unified Modeling Language 2003 (Workshop 7: Consistency Problems in UML-based Software Development), San Francisco, USA*, Technical Report. Blekinge Institute of Technology, San Francisco, October 2003.

Evaluation of the CMPSs Programming Model for Consumer Applications

Ben Juurlink
TU Berlin
Einsteinufer 17
D-10587 Berlin
juurlink@cs.tu-berlin.de

Chi Ching Chi
Chi Ching Chi
TU Berlin
Einsteinufer 17
D-10587 Berlin
cchi@cs.tu-berlin.de

Michael Andersch
TU Berlin
Einsteinufer 17
D-10587 Berlin
michael.andersch@mailbox.tu-berlin.de

Abstract

Since the advent of mainstream multicore systems, developing multicore software has been an increasing concern. Many challenges arise when trying to develop multicore software that is both efficient and portable on the one hand, as well as extensible and modular on the other hand. To ease the development of multicore software, the abstraction layers between the execution hardware and the application need to be changed. In the EU Strep project Encore [1], a task-based programming model called SMP Superscalar (SMPSs) [2] is investigated. In this programming model the programmer annotates potential parallel parts of the serial base code with task directives and input and output directives that specify the inputs and outputs of each task. Based on these directives, the compiler framework and underlying runtime system ensure correctness, parallel execution, and locality-aware scheduling.

Our goal is to implement consumer applications and benchmarks using SMPSs and other more established multicore programming models such as Pthreads and compare them based on performance and code quality. An important application in this comparison is H.264 video decoding. Unlike H.264 encoding, parallelism is not obvious and hard to exploit. Our analysis of implementing H.264 decoding on the Cell processor [3] has shown that H.264 decoding requires high memory bandwidth, especially at higher resolutions. Furthermore, locality- and resource-aware scheduling could bring significant performance improvements.

Image processing and ray-tracing are also investigated to have a good coverage of the application domain. This includes both simpler image rotation and color conversion kernels as well POV-Ray [4], a widely used ray-tracer benchmark.

The HPI machines provide the necessary platforms for this analysis, as they are multicore, multisocket, SMT and NUMA capable machines. This allows us to investigate the behavior of our target applications as well as showing the capabilities of and potential areas of improvements to the SMPSs programming model. In particular, we want to investigate if function-level (pipeline) parallelism can be conveniently expressed in SMPSs and if additional directives are needed to control the placement of threads and data onto the cores.

References

[1] http://www.encore-project.eu

[2] Barcelona Supercomputing Center. SMP Superscalar (SMPSs) User's Manual. Available via http://www.bsc.es/media/1002.pdf

[3] C. C. Chi, B. H. H. Juurlink, C. H. Meenderinck. Evaluation of Parallel H.264 Decoding Strategies for the Cell Broadband Engine. Proc. Int. Conf. on Supercomputing, 2010.

[4] http://www.povray.org/

Query Processing on Prefix Trees

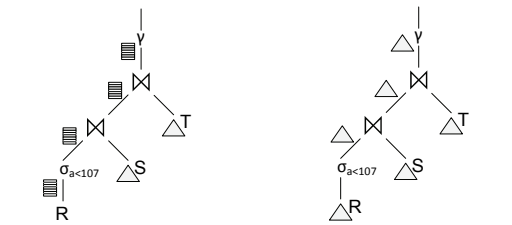
Matthias Boehm Patrick Lehmann Peter Benjamin Volk Wolfgang Lehner
TU Dresden, Database Technology Group; Dresden, Germany
matthias.boehm@tu-dresden.de

Abstract

There is a trend towards Operational BI (Business Intelligence) that requires immediate synchronization between the operational source systems and the data warehouse infrastructure in order to achieve high up-to-dateness for analytical query results. The high performance requirements imposed by many ad-hoc queries are typically addressed with read-optimized column stores and in-memory data management. However, operational BI additionally requires transactional and update-friendly in-memory indexing due to high update rates of propagated data. For example, in-memory indexing with prefix trees exhibits a well-balanced read/write performance because no index reorganization is required. The vision of this project is to use the underlying in-memory index structures, in the form of prefix trees, for query processing as well. Prefix trees are used as intermediate results of a plan and thus, all database operations can benefit from the structure of the in-memory index by pruning working indices during query execution. While, this is advantageous in terms of the asymptotic time complexity of certain operations, major challenges arise at the same time. In this paper, we sketch our preliminary project results. Efficient query processing over huge evolving data sets will enable a broader use of the consolidated enterprise data. Finally, this is a fundamental prerequisite for extending the scope of BI from strategic and dispositive levels to the operational level.

1 Introduction

Advances in information technology combined with rising business requirements lead to an exponentially growing amount of digital information created and replicated worldwide [7]. In addition to this huge amount of data, there is a trend towards Operational BI (Business Intelligence) [6, 10, 13] that requires immediate synchronization between the operational source systems and the data warehouse infrastructure in order to achieve high up-to-dateness for analytical query results. The high performance requirements imposed by many ad-hoc queries are typically addressed with read-optimized, in-memory data management.



(a) Traditional Query Plan (b) Prefix Tree Query Plan

Figure 1. Solution Overview

However, Operational BI additionally requires transactional and update-friendly main-memory indexing due to the existence of high update rates. For example, in-memory indexing with prefix trees, in combination with optimistic concurrency control, exhibits a well-balanced read/write performance [12] because no index reorganization is required. In addition, advanced data analytics [4, 5] such as forecasting [1], which go beyond traditional data aggregation and analytical query processing, also require in-memory indexing to enable efficient point and range queries.

The vision of this research project is to use the underlying in-memory prefix tree index structures, which are required for a balanced read/write performance, for query processing as well. As shown in Figure 1, prefix trees are used as intermediate results of a query execution plan and thus, all database operations can benefit from the structure of the in-memory index by pruning working indices during query execution.

Example 1 *Consider the query* $\gamma(\sigma_{a<107}(R) \bowtie S \bowtie T)$. *A traditional query execution plan as shown in Figure 1(a) might also use the underlying index structures, e.g., for index nested loop joins of* $* \bowtie S$ *and* $* \bowtie T$. *In contrast, our idea is to use the underlying index structure as intermediate results of* all *operators rather than just at the leaves of a query plan.*

While, this is advantageous in terms of the asymptotic time complexity of certain operations and thus, allows for efficient query processing on very-large amounts of data, major challenges in terms of memory management, query processing and scalability arises at the same time. The objective of this project is to leverage the HPI Future SOC Lab infrastructure in order to

evaluate our initial prototype in large-scale infrastructures and to adapt our framework with regard to these preliminary results.

With the aim to provide an overview of the project idea, we make the following contributions that also reflect the structure of the paper. In Section 2, we explain selected prefix-tree-based operators and sketch the query transformation. Subsequently, in Section 3, we present preliminary experimental results. Finally, we review the project status and planned future work in Section 4 and conclude the paper in Section 5.

2 Query Processing Overview

As our underlying in-memory index structure, we use the *generalized trie* [2] that is a prefix tree (trie) with variable prefix length of k' bits. We define that (1) $k' = 2^i$, where $i \in \mathbb{Z}^+$, and (2) k' must be a divisor of the maximum key length k. Given an arbitrary data type of length k and a prefix length k', the trie exhibits a fixed maximum height $h = k/k'$ and each node of the trie includes an array of $s = 2^{k'}$ references (node size). The trie path for a given key k_i at level l is then determined with the l^{th} k'-bit prefix of k_i. In addition, we use *trie expansion* such that subtries are only expanded if required (if multiple keys share the same prefix). This trie exhibits (1) the deterministic property such that any key has exactly one path within the trie and (2) a constant worst-case time complexity of $O(h)$. Combining this in-memory index structure and our vision of query processing on prefix trees, in this section, we present selected prefix-tree-based operators and the related query transformation.

2.1 Selected Operators

In general, we use the relational algebra but extend it on physical and logical level to a closed prefix-tree-based algebra, where the input and output of each operator, except `ixEmit`, is represented by a prefix tree. The generalized trie with its deterministic property then allows for specific optimizations.

Query Plan Leaf Operators: There are three possibilities of query plan leaf operators. The `ixCopy` simply copies the complete memory block of an index because due to pruning during query processing, we would destroy our underlying in-memory index structures. However, for tries (1) that are read-only during query processing, or (2) that can be reused from previous query executions [8], we use the `ixRef` to simply reference an existing index. Finally, for equality selection predicates, the `ixGet` operator obtains a single key partition of the underlying trie by a point query.

Conjunctive, Disjunctive, and Range Selection Predicates: While conjunctive selection predicates are realized by a sequence of `ixGet` operators, disjunctive predicates on the same attribute are addressed by the concept of query-data joins [3]. Having long `IN`-lists of disjunctive predicates in mind, the `ixSelectOr` operator builds a prefix tree over all predicates and executes a semi-join between the input trie and this predicate trie. In contrast, disjunctive predicates on different attributes require a linear scan by `ixSelectMultiple` or a union of multiple `ixGet`. For range predicates, we use the `ixSelect`:

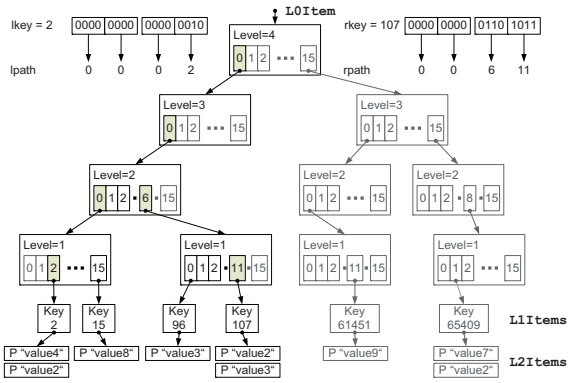


Figure 2. Example `ixSelect` Operator

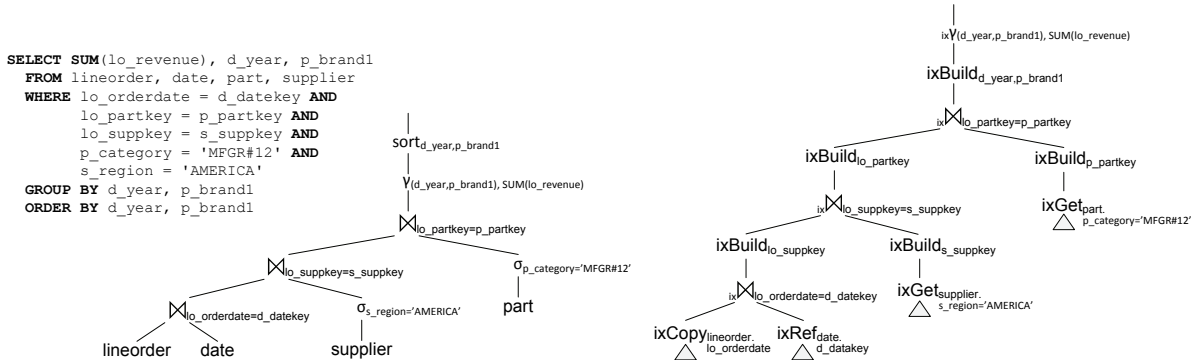
Example 2 *Assume a simple range predicate query $\sigma_{2 \leq a \leq 107} R$. Using a working index over R.a (e.g., created with an `ixCopy` operator), we use an `ixSelect` operator in order to prune the trie as shown in Figure 2. We go down the trie for the left key (2) and prune all subtries left of this path. Subsequently, we repeat this for the right key (107). Thus, the range predicate can be evaluated independent (except for the trie height) of input and output sizes of this query, while an index scan depends on the output size.*

Natural-, Equi- and Theta-Joins: For joins we compare tries by exploiting their deterministic property. There, we recursively compare the references of both join inputs. For each join input side, we distinguish the cases (1) subtrie, (2) key partition, and (3) null reference. If at least one side is a null reference, we can prune the complete subtries of both sides. Thus, we might observe sublinear asymptotic behavior, where only the worst-case complexity is linear. Similar concepts are used for the other binary operators such as `ixUnion`, `ixDiff`, and `ixIntersect`.

Group-By and Order-By: The generalized trie is inherently partitioned due to the deterministic trie paths. Thus, the `ixGroupBy` operator for multiple attributes recursively creates tries for each attribute with the scope of the current key partition only. Due to order-preserving indexing, the `ixSort` for multiple attributes uses exactly the same conceptual idea. As a result, both operators exhibit a linear time complexity.

2.2 Query Transformation

Putting it altogether, we use the query transformation procedure to show how these operators are combined in order to answer traditional analytical queries.



(a) Traditional Query Processing (b) Query Processing on Prefix Trees

Figure 3. Example Query Transformation for SSB Q2.1

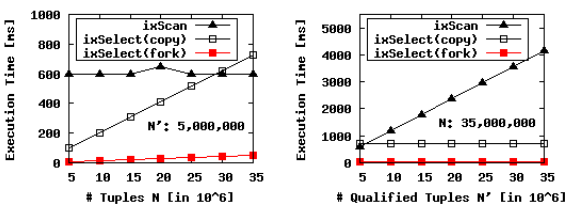
Before discussing query transformation, we need to introduce the additional `ixBuild` operator. We use secondary indices as intermediate results. If subsequent operators require different attributes, we need to invert the obtained intermediate index. This is realized by a linear scan over the input trie and inserting all tuples into a new temporary index over the required attribute. However, there are many different alternatives on logical plan level that might be used for optimization.
Query transformation (without optimization) is then realized as follows: We start with an traditional query plan. For each leaf node, we decide on the prefix-tree-based access (`ixCopy`, `ixRef`, or `ixGet`). Subsequently, we replace the logical non-leaf operators with our prefix-tree-based operators. If the attributes of input tries do not match up the operator description, we insert an `ixBuild` before this operator. Finally, we remove unnecessary operators such as single attribute `ixGroupBy` or redundant `ixSort` operators.

Example 3 *Assume the query Q2.1 of the Star Schema Benchmark (SSB) [11] as shown in Figure 3(a). The query plan consists of a left-deep join tree, two equality selection predicates as well as a final group-by and sort including two attributes. In contrast to this, the prefix-tree-based query plan includes several modifications. We create the* lineorder *index by an* `ixCopy` *operator, while the* date *index can be simply referenced by an* `ixRef` *operator because it is used read-only as the right side of a join. Equality selection predicates on* supplier *and* part *lead to the use of point queries in the form of* `ixGet` *operators. In addition, we insert five* `ixBuild` *operators in order to invert intermediate result tries. Due to the equality of group-by and order-by attributes (*d_year, p_brand1*), we finally remove the redundant* `ixSort` *operator.*

Such prefix-tree-based query plans exhibit potential for optimization. For example, we can reuse intermediate results [8], of rather static dimension tables, such as the result tries of $\texttt{ixBuild}_{p\_partkey}$ or $\texttt{ixBuild}_{s\_suppkey}$ by an `ixRef` operator because they would be used read-only as the right side of a join.

3 Experimental Evaluation

As an example, we evaluate the scalability for range queries of the form $Q : \sigma_{x \leq R.a \leq y} R$, where $N = |R|$ denotes the table cardinality and $N' = |\sigma_{x \leq R.a \leq y} R|$ denotes the result cardinality. Our prototype (implemented in C) maintains a table `R(a INTEGER(4), b VARCHAR(100))` with a secondary index on `a`. As our test environment, we used a *Desktop Intel Nehalem* with one processor (quad core Intel i7 Core at 2.67GHz), hyper-threading (2 threads per core), Fedora Core 13 (64bit) as operating system and 6GB of RAM. We inserted a sequence of N tuples and compared the traditional index scan `ixScan` (point query on x, scan until y) with our prefix-tree-based `ixSelect` ((1) with `ixCopy`, and (2) with `ixFork` for lazy copy-on-write as used for shadow paging [9]) for an query result size of N'. We observe that `ixScan` is only influenced by the number of output tuples (Figure 4(b)). In contrast, our `ixSelect` only depends on the number of input tuples (Figure 4(a)) because it is dominated by the `ixCopy` operator. Note that the reuse of allocated memory blocks for temporary indices within the `ixCopy` reduces the execution time by 52.5 % (not shown in the figures). Most importantly, the use of fork led to significant improvements. The major finding is that the percentage (N'/N) from where the `ixSelect` is beneficial decreases with increasing data size (lower slope), which makes it especially applicable for large-scale data sets.



(a) Varying Input Size N (b) Varying Output Size N'

Figure 4. Range Query Performance

4 Project Status and Future Work

Despite the just recent project start of September 1st 2010, we give a brief overview of the current project status and our planned future work.

4.1 Current Project Status

Based on our conceptual ideas and the initial prototype of the *generalized trie*, we started to build the required prefix-tree-based query processing framework for this project. This includes a runtime for prefix-tree-based operators and query processing, memory and record management as well as meta data management. We also investigated the inter-influences to transaction management and query compilation.
The current status of the project is a partially finished promising initial prototype that allows for preliminary experimental investigation. However, many aspects that offer further optimization potential are not addressed so far such that a first completely finished prototype is expected in the second quarter of 2011.

4.2 Used Future SOC Lab Resources

The objective of this project is to leverage the HPI Future SOC Lab infrastructure in order to evaluate our initial prototype in large-scale infrastructures. Thus, we use the available resources mainly with the aim of experimental evaluation. This includes, in particular (1) the scalability with increasing data size (in-memory), and (2) the scalability with increasing number of threads.
Until now, we used the HPI Future SOC Lab resource `Fujitsu RX600 S5 1, CPU: 4 x Xeon (Nehalem EX) E7530, RAM: 256 GB`. The major benefit for us is the possibility to evaluate our in-memory prototype on large-scale data sets due to the available main memory resources. This allows us to adapt our framework with regard to the obtained results, which enables a future-oriented investigation of the idea of query processing on prefix trees.

4.3 Next Steps

Beside the outstanding work for the integrated prototype, there is plenty of future work regarding further optimization potential. This includes the following three major research directions.

- Memory Management (garbage collection, optimized lazy *copy-on-write* for efficient copying)
- Query Processing (recycling intermediates, hybrid row/column storage, physical operator alternatives, cost-based query optimization)
- Scalability (scalability with increasing data sizes, inter-operator parallelism by pipelining subtries, intra-operator parallelism by task partitioning)

5 Conclusions

Based on the requirements of (1) balanced read/write performance and (2) efficient index support for point and range queries, we proposed the vision of query processing on prefix trees. We sketched the project idea as well as promising preliminary conceptual and experimental results. Efficient query processing over huge evolving data sets will enable a broader use of the consolidated enterprise data. Finally, this is a fundamental prerequisite in order to (1) extend the scope of BI from strategic and dispositive levels to the operational level, and to (2) enable advanced data analytics that goes beyond traditional data aggregation.

Acknowledgment

We want to thank Frank Dietze, Steve Reiniger, and Thomas Dedek for their efforts on implementing parts of this prefix-tree-based query processing framework.

References

[1] Deepak Agarwal, Datong Chen, Long ji Lin, Jayavel Shanmugasundaram, and Erik Vee. Forecasting high-dimensional data. In *SIGMOD Conference*, 2010.
[2] Matthias Boehm, Benjamin Schlegel, Peter Benjamin Volk, Ulrike Fischer, Dirk Habich, and Wolfgang Lehner. Efficient in-memory indexing with generalized prefix trees. In *BTW*, 2011.
[3] Sirish Chandrasekaran and Michael J. Franklin. Streaming queries over streaming data. In *VLDB*, 2002.
[4] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, and Caleb Welton. Mad skills: New analysis practices for big data. *PVLDB*, 2(2), 2009.
[5] Sudipto Das, Yannis Sismanis, Kevin S. Beyer, Rainer Gemulla, Peter J. Haas, and John McPherson. Ricardo: integrating r and hadoop. In *SIGMOD Conference*, 2010.
[6] Umeshwar Dayal, Malú Castellanos, Alkis Simitsis, and Kevin Wilkinson. Data integration flows for business intelligence. In *EDBT*, 2009.
[7] IDC. *The Digital Universe Decade - Are You Ready?* IDC, 2010.
[8] Milena Ivanova, Martin L. Kersten, Niels J. Nes, and Romulo Goncalves. An architecture for recycling intermediates in a column-store. In *SIGMOD Conference*, 2009.
[9] Alfons Kemper and Thomas Neumann. Hyper: Hybrid oltp&olap high performance database system. Technical report, 2010. TUM-I1010.
[10] William O'Connell. Extreme streaming: business optimization driving algorithmic challenges. In *SIGMOD Conference*, 2008.
[11] Pat O'Neil, Betty O'Neil, and Xuedong Chen. *Star Schema Benchmark - Revision 3*, 2009.
[12] Elizabeth G. Reid. Design and evaluation of a benchmark for main memory transaction processing systems. Master's thesis, MIT, 2009.
[13] Richard Winter and Pekka Kostamaa. Large scale data warehousing: Trends and observations. In *ICDE*, 2010.

Build Automation as a Service

Martin v. Löwis
Hasso-Plattner-Institut
Postfach 900460
D-14440 Potsdam
Martin.vonLoewis@hpi.uni-potsdam.de

Bernhard Rabe
Hasso-Plattner-Institut
Postfach 900460
D-14440 Potsdam
Bernhard.Rabe@hpi.uni-potsdam.de

Abstract

Build Automation assumes new roles in software development as availability of fast hardware increases. One wide-spread application is continuous integration, the other software production. In this project, we experiment with the usage of high-performance hardware to provide build automation as a service, in a manner where the developers of the software don't require access to the hardware, and the operators of the hardware can rely that even malicious code inject by a developer (or an adversary masquerading as a developer) will not harm the infrastructure or data of other users. The specific experiments were performed in the context of the Python language project.

1 Introduction

Build and test automation is a core aspect of Continuous Integration (CI). As developers complete the implementation of a new feature, this feature can get integrated immediately if it is known not to break the current code base. The software needs to be rebuilt, and the test suite needs to be re-run. If the test suite fails, the change needs to be reverted, or (if a patch queue manager2 is used) not be integrated in the first place. Martin Fowler recommends that the build should be quick, and that it should be performed once for every commit to the source code repository.

Unfortunately, for a larger project, building the software and running the test suite can take between several minutes and several hours. The productivity gain of continuous integration is quickly lost if the result of the automated build are available only a few hours after the change was made – the developer might be working on something else meanwhile, and other developers may get stalled by the broken build.

Fortunately, using multi-core systems, build automation can be significantly accelerated: most build processes already support parallel execution. Unit test suites often would allow independent and parallel execution of tests as well, even though the test runners currently often enforce sequential execution; developers typically don't invest into parallel testing as they do not have the necessary hardware available, anyway.

Another aspect of build automation is software production: every time a software system is released, a number of build steps have to be performed. Some are manual (e.g. updating the release notes, bumping the version number), while others can be automated. In many larger projects, production is fairly automated already.

However, in some settings, running the entire production process on a single computer is not feasible. For example, building installation packages for different processor architectures or operating systems will typically require separate computers (unless cross-compilation can be used).

In this project, we investigated some aspects of build automation on future systems, using service-oriented approaches. We investigated two different build tasks, which we will describe first. Then we elaborate the architecture used, and finally we report the results we received so far, along with identifying remaining issues.

2 Continuous Integration in Python

The oldest and most widely used implementation of the Python programming language is based on an interpreter written in C; this implementation is often referred to as CPython [1]. The interpreter along with various standard library modules that access operating system facilities has been ported to various hardware platforms and operating systems; the most common target systems are Microsoft Windows, the POSIX family of operating systems, and Apple Mac OS X. The core group of developers currently consists of about 130 contributors [2].

The CPython source code is currently maintained in a central Subversion repository [3]. For further discussion, we restrict attention to the build procedures used for POSIX target systems, in particular Linux, as this was the system we used in the Future SOC experiment. To build the interpreter, an autoconf-based configure run is followed by a make-based compilation.

The CPython source code also includes a unit test suite, consisting of about 350 test modules, each performing a varying number of tests. Developers are expected to run the test suite before every commit. However, practice demonstrates that developers sometimes skip running some or all of the tests if they are convinced that the change cannot possibly break tests cases. Also, they can only run the test suite on the hardware they have available; however, incompatibilities between operating systems may cause some change to break the test suite on some particular systems.

To improve the stability of the code base, a continuous integration process was set up by one of the authors a few years ago, using the buildbot system. As shown in figure 1, each commit operation to the source repositories triggers a notification to the build master, which in turn notifies all build slaves. Each slave then performs a full build and test suite run, and reports all output and the summary status to the master. The master, in turn, sends out email notifications if the modification has broken the build on some system (i.e. if the build was working before the change, but failed afterwards).

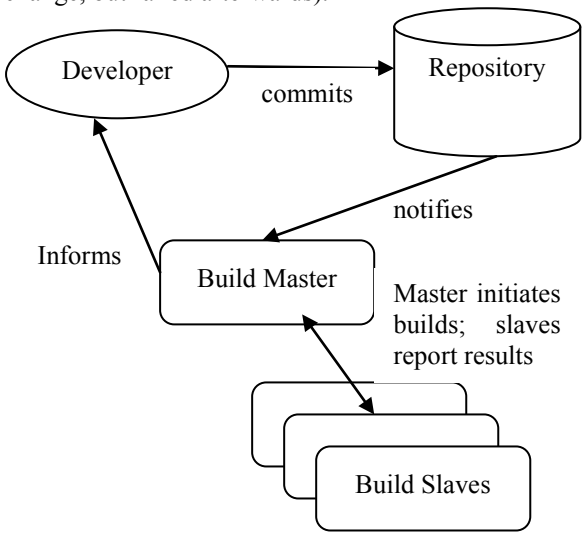


Figure 1. Information flow in buildbot.

In the past, problems specific to only selected platforms were often detected only months after the change had been made, in many cases only after a release of the interpreter had been made. With the CI process, problems are detected much earlier (for the systems which participate as build slaves).

However, a remaining issue is that the errors are detected *after* the change has been committed to the source control system. If the change causes problems, somebody would need to investigate the problem and devise a solution. Practice has shown that this is often not the original author of the change, since she felt that she had accomplished the objective (of getting the change integrated into CPython). This is sometimes viewed as unfair among contributors, as the convention says that whoever introduces the errors should also work to remove them.

One cause of this problem is that the buildbot feedback is not timely. Build times vary between 30 minutes and 2 hours. Developers will often be busy with other tasks when the results arrive, and unable to react immediately.

To improve this situation, the objective of this project is to significantly reduce build times on a selected system, to give developers an instantaneous feedback whether the change has caused problems.

2.1 Buildbot Slave Setup

In the Future SOC lab, a buildbot slave was set up on the Fujitsu RX600S5 system. Standard Debian packages were used; in the course of the project, the build area was moved from a local hard disk to the EMC storage via NFS.

2.2 Buildbot Master Changes

In addition to setting up a slave, changes to the master have been implemented, in order to facilitate the available hardware. These changes request that all build steps for which parallel execution is possible are indeed executed in parallel. While it might be more appropriate that the slave requests parallel execution (instead of this being a master-side configuration), buildbot does not currently support such slave-side configuration.

In particular, the following changes have been made:

- The make build program is invoked with the option '-j48', which requests the execution of up to 48 processes in parallel. This number was chosen to match the number of virtual (hyper-threaded) processors in the hardware.
- Likewise, the test runner (Python's regrtest.py) was invoked with the option '-j48', having the same effect on the tests as well.

The actual results of this setup are reported further below.

3 Automated Builds in the Python Package Index

PyPI, the Python Package Index [4] is a central infrastructure in Python community which allows open source authors to announce their Python-related projects, and also publish the actual source code and documentation if they desire (alternatively, they may only link to their project home page from the index).

Many authors do indeed provide source code as well as selected pre-packaged binary packages, typically for Windows. Developers typically can't provide binary packages for all systems, as they don't have access to all the systems for which users may request binary packages.

Within this Future SOC project, we experimented with automatically building binary packages for all projects listed in PyPI, for selected operating systems (namely different Ubuntu versions). The objective

would be to provide an Ubuntu package repository which would allow Ubuntu users to install PyPI packages using their regular package management tools (i.e. aptitude(8)). In principle, this approach would also work for other systems (such as FreeBSD or Solaris), however, the specific build procedure will need to be adopted for every system.
In dealing with this build task, we detected to major challenges: integration with the Debian packaging system, and security threats for the build infrastructure.

3.1 Integrating with the Debian Packaging System

A Debian package is a collection of files to be installed on the target machine, plus a number of package management metadata. These metadata primarily consist of three groups:

- human-targeted package information (name, description)
- package dependencies (which other packages are needed to use this package)
- custom installation procedures (pre/post install/uninstall scripts)

In order to automatically generate Debian packages, this information must also be obtained in an automated manner. Fortunately, Python packages in PyPI often are based on distutils [5], which then already provides much of the needed information. Missing information can then be left blank or filled with boilerplate text.
Dealing with package dependencies is more complicated for two reasons: First, many PyPI packages don't include dependencies, and if they do, they only refer to other PyPI packages, not to Debian packages which they also may depend on. Second, a number of PyPI packages are already available in Ubuntu, although perhaps not in the latest version. It would be desirable to record dependencies to these versions of the packages, so that users don't have to deal with conflicting versions on their systems.
Fortunately, Andrew Straw's stdeb package [6] already deals with much of the metadata problems for Debian packages (and is applicable to Ubuntu also); hence the automated build process relies mostly on stdeb to fill out the package metadata correctly.
The issue of custom installation procedures can be solved in a uniform manner for Python packages. Even though custom steps are indeed necessary (in particular, to create Python bytecode files on the target system), the Debian python-support library simplifies this task; stdeb uses this library by default.

3.2 Executing Untrusted Code

One key issue for build automation as a service is the trust into the foreign code. In the majority of build procedures, authors of the software can integrate arbitrary algorithms, to deal with specific tasks that arise in building the software. Unfortunately, this means that, in principle, developers could also inject malicious code that then forms a threat for the operator of the build automation service.
In the first part of the project, this was only a minor concern: the possible developers that have access to the build procedures are all well-known and trusted, and the build process runs under an unprivileged account.
In building PyPI packages, the authors are not individually known. While it is likely that they will have no malicious code in their build procedures, the threat of somebody uploading a package to specifically attack the build infrastructure is real.
To cope with this problem, we have used virtual machines to perform all build steps in a sandbox. At the end of the build, all system modifications were discarded, except that the actual build results (i.e. the Ubuntu packages) were preserved.
The details of the setup so far are described in the results section.

4 Results

The two different build projects have a different degree of completion, with different insights that we have gained.

4.1 Continuous Integration

The main objective of the project was to reduce the build time. This objective has been achieved: a typical build of the Python 3 branch now completes in ca. 8 minutes. This speedup was primarily achieved from the parallel execution of the test suite, which uses ca. 6 minutes of the total.
However, a number of questions remain:

1. Given that there are 48 processes running tests, and given that there are 350 test modules that could be run in parallel, why is the speedup so low (below 10)? Most likely, some individual tests take a long time to complete; this has not been analyzed further.
2. Some of the test cases fail on the Future SOC hardware, yet pass on other systems. By a shallow inspection, this is likely due to non-standard behavior of the storage system.

Elaborating on the second issue, we see two kinds of failures: First, file names with non-ASCII characters apparently cause problems (e.g. a file that ought to be named "Grüß-Gott" is actually reported as having the name "Gr\udcfc\udcdf-Gott"). Second, files show incorrect time stamps: a file that should have been modified at time 1041808783 is reported to have 1041808783.000001 as its modification time).

4.2 Building PyPI packages

After researching alternatives, we decided to use the Sun/Oracle product VirtualBox [7] to perform the virtualization. This product allows users to create operate the hard disk in a "immutable" mode, where

changes are discarded after the machine shuts down. In this project, we set up a virtual hard disk with all build tools in place, and then change the hard disk to be immutable. All disk changes made in an individual build can then be discarded at the end of the build, preventing some build from tampering with the files on the disk.

To automatically integrate the individual builds, we use again buildbot: a build slave will start a virtual machine, wait for the build to complete, and then upload the build results to the master.

One particular challenge is the communication between the build slave (which runs in the host system) and the virtual machine. Currently, we concentrate on the Shared Folders feature of VirtualBox: a dedicated directory of the host system is integrated into the virtual machine. Before the build starts, the build configuration is placed into that folder. As part of the boot process, the build process is launched and processes the build task. It then needs to shut down the virtual machine, leaving only the build results behind in the shared folder.

This project is still in progress; actual results on the build time for the entire package index are not yet available. In particular, the following tasks are still not done:

- Computation of dependencies between packages, and a global scheduling of the order in which builds should occur.
- Shutdown of the virtual machine at the end of the build
- Production of packages for multiple Ubuntu releases, in particular for the Intel x86 and AMD-64 architectures.

5 Future Work

As indicated above, further tasks need to be performed in each of the subprojects. For the continuous integration project, the critical path must be investigated, to determine whether build times can be further reduced. For the automated package builds, the build infrastructure needs to be completed and run at least once.

6 References

[1] Alex Martelli. Python in a Nutshell (2nd edition ed.). O' Reilly 2006.

[2] Python Software Foundation. Python Committers. http://www.python.org/dev/committers

[3] Python Software Foundation. Python Source Repository. http://svn.python.org/projects/python/

[4] Python Software Foundation. Python Package Index. http://pypi.python.org/pypi

[5] Greg Ward, Anthony Baxter. Distributing Python Modules. http://docs.python.org/distutils/index.html

[6] Andrew Straw. stdeb. http://pypi.python.org/pypi/stdeb

[7] Oracle. VirtualBox. http://www.virtualbox.org/

Parallel Aggregation and Join Computation in NewDB – Project Status October 2010 –

Christian Bensberg, Nico Bohnsack, Christian Mathis,
Kai Stammerjohann, Frederik Transier
SAP AG

Stephan Müller, Jan Schaffner, Christian Tinnefeld
Hasso Plattner Institute

Abstract

Database architecture follows hardware trends. Current hardware trends like very large amounts of main memory, multi-core processors, and big data centers for cloud computing are pioneered by SAP's NewDB—a parallel main-memory database system with multi-tenant support. In this project proposal, we apply for the support of the Future SOC Lab to investigate two algorithms specially developed for NewDB: parallel in-memory variants of relational join and aggregation. These operations are among the most frequently executed algorithms, both in OLAP-style and OLTP-style data processing. Based on the expected results, we will develop a cost model for parallel aggregation and join algorithms and a calibrator tool to automatically parameterize our algorithms for specific hardware platforms. This paper presents the intermediate status of the project obtained till October 2010.

1 Introduction

Hardware trends have always influenced database architectures. This will also be true for the following developments, from which we believe that they will heavily impact software development in general and database system design in particular:

- *The Shift in the Memory Hierarchy*: Dropping DRAM prices and exponentially growing DRAM capacities modify the classical memory hierarchy. Nowadays, a single blade can hold 1 TB of main memory. Systems with 50 of these blades can store the ERP data of the worlds largest companies [6].

- *Multi-Core Architectures*: Chip manufacturers banged against several walls known as the "heat wall", the "ILP wall" (Instruction Level Parallelism), and the speed of light [1]. These walls effectively ended the exponential CPU frequency growth. To nevertheless cope with performance demands, chip manufacturers put multiple cores inside a CPU. As a simple corollary, software vendors now cannot rely on frequency-based performance speed-up anymore. Instead, they have to parallelize their software to—ideally—scale linearly with the number of available cores.

- *Data Centers*: Many large IT companies invested in data centers, where they centralize computing power and storage capabilities to build the infrastructure for flexible computing clouds. Hardware virtualization is the major enabler for this trend. Customers do not have to invest in hardware anymore. Instead, they rent virtual computers and storage space "in the cloud".

Next generation database management systems will have to cope with these developments. Although parallelism and large main memories have always been an issue to database systems, we think the above sketched developments take data processing to the next level:

- Purely in-memory systems overcome the classical external-memory gap rendering database buffers superfluous. On the flip side, main-memory I/O becomes the limiting factor. Now, issues like the cache-miss ratio, cache alignment, cache-line splits, cache flushes, and prefetching strategies become the dominant factors influencing query execution performance.

- Database servers were almost always designed to exploit parallelism (e. g., inter-query, intra-query, inter-operation, and intra-operation parallelism). However, we will soon get machines with more than 64 cores per server node (like the machine announced in the "Call for Projects" of the *Future SOC Lab*), and we will be able to build clusters with more than 100 blades. Database vendors will have to think about how to make the most out of these highly parallel systems.

- A data store is part of every cloud platform. Many cloud vendors have chosen to develop proprietary data store implementations (e. g., Google's BigTable [3] or Amazon's S3 [2]) rather than building on traditional database systems. However, many customers will want to build their applications on the well-known and well-established features provided by classical systems, thus requiring "cloud-aware database systems".

SAP's next-generation database management system (called "NewDB") responds to these hardware trends. In short, NewDB is a highly parallel main-memory database system which can be installed on a cluster of blades. At its core, it has a columnar table store and provides ACID transactions, a SQL interface, and multi-tenancy support.

2 Research Questions

Within NewDB, the authors are working on query processing algorithms. Our goal is to design and implement algorithms that meet the above sketched hardware developments, i. e., that are aware of the memory hierarchy and scale well with the available number of cores on a blade. In the focus of the present *Future SOC Lab* project proposal are two recently developed algorithms for parallelized *join* and *aggregation* computation. Join and aggregation operations occur in almost every relational query (be it OLTP-style or OLAP-style). Due to the computational complexity and/or vast data access, join and aggregation are potentially expensive and have a major impact on any relational database system. Therefore, we think our work is not only interesting w. r. t. NewDB, but to relational database research in general.
Surrounding our new algorithms, we identified the following research topics to be tackled with the help of the *Future SOC Lab*:

1. Our algorithms are internally parallelized (intra-operator parallelism). How do our algorithms scale with the number of available cores?
2. How do our algorithms scale with the input size and what is the best degree of parallelism for a given input size?
3. How can we adjust our algorithms to the specific characteristics of the memory hierarchy (cache size, memory access time, I/O bandwidth)?
4. How does main memory consumption depend on the input size and the query complexity? How is memory allocated/freed during processing? What are the implications for memory management?
5. In multi-user/multi-tenant scenarios: how do we best schedule multiple concurrently running aggregation/join queries? Is it better to serialize the execution of concurrently issued aggregation/join operations or should we allow the scheduler to run multiple aggregation/join operations in parallel (inter-operator parallelism)?

Based on the results of our measurements at the *Future SOC Lab*, we will develop a cost model for parallel aggregation and join algorithms. Such a cost model provides valuable information for the query optimizer and the scheduler. The results will also provide the basis to develop a calibrator tool (similar to [5]), which can automatically parameterize our algorithms w. r. t. a specific hardware platform at system startup.

3 SOC Lab Experiments

As a first step, we tackled the first point in the above list of research questions: "How do our algorithms scale with the number of available cores?". To assess the performance of the algorithm under various input distributions and orderings, we generated a data set of 168 tables, each table containing roughly about 100 Million entries. The tables have the following schema (simulating a table for material management):

```
CREATE TABLE generated:<D><#(S)>(
id integer primary key,
matno integer,
value decimal(14,2))
```

Attribute id identifies each row. Attribute `matno` is a material number, drawn out of a set S of predefined materials. The order of material numbers depends on the distribution D of a table. The value attribute contains a decimal number, which indicates a flow of the material (+ for incoming, - for outgoing flows). We generated the following distributions D, following the experiment in [4]:

- Uniform – For each row, a material number is randomly chosen from S.
- Heavyhitter – 50 % of the rows contain a certain material number (the heavy hitter), all other rows are distributed uniformly.
- Sorted – The distribution of the material numbers is uniform, but the rows are sorted by the material number.
- Sequential – The distribution is again uniform, but the material numbers form a consecutive and repetitive sequence of size $\mathrm{card}(S)$. For example, if we had material numbers from 1 to 100, the sequence would be 1, 2, 3, , 100, 1, 2, 3, 100, and so on.
- Movingcluster – The material number for row i is chosen uniformly from the range $\lfloor (card(S)W)i/R \rfloor$ to $\lfloor (card(S)W)i/R + W \rfloor$, where R is the size of the table (e.g., 100 Million rows) and W is the window size (we chose

$W = 1024$). This distribution generates some "locality" within a window of size W.

- Pareto – Is a self-similar distribution that adheres to the 80-20 rule: 20% of the rows in the table contain 80% of all material numbers. The remaining 80% of the rows contain only 20% of all material numbers, whose distribution again adheres to the 80-20 rule.
- Zipf –The frequency of a certain material number is inversely proportional to its position in the frequency table, e.g., material number X at position y in the frequency table occurs twice as often as material number X at position $y + 1$.

We used the Python *numpy* package to generate the Pareto and the Zipf distributions. For the different distributions, we generated tables with 100 Million rows and the following cardinalities: $2, 2^2, 2^3, \ldots, 2^{24}$ resulting in 168 tables. These tables were imported into NewDB, resulting in a 250 GB database.
On each table, we executed the following three queries 10 times and took the median execution time of the aggregation in milliseconds (see [4]):

- Q1:
```
select count(*), sum(value),
sum(value*value), matno from
generated:<D><S> group by matno
```
- Q2:
```
select max(value), min(value),
min(value), matno from
generated:<D><S> group by matno
```
- Q3:
```
select distinct matno from
generated:<D><S>
```

The additional min aggregation in Q2 facilitates the comparison with Q1. To make the results comparable, we calculated the throughput (number of rows per millisecond) of the aggregation algorithm, rather than the absolute computation time. We used the NewDB default parameters for the parallel aggregation algorithm, except for the number of cores, which we varied to asses the aggregation algorithm.
The benchmark was conducted on Fujitsu RX600S5 – a machine with 32 Cores (Intel(R) Xeon(R) CPU X7550) and 1024 GB main memory and OpenSUSE 11. The benchmark client was implemented in Python and ran on the server machine. We used a preliminary version of NewDB. Note, with 64 cores, we rely on Intels Hyper Threading.
We executed each query 11 times. The first run was omitted, because this run contains the time to load the queried table into main memory. Of the remaining 10 times, we recorded the median run time. The run time was measured with an internal stop watch, which only recorded the time for the aggregation (i.e., no communication overhead was recorded).

4 Findings and Next Steps

The findings of the experiments not available for publication yet. The above described synthetic data set helped to identify the behavior of the algorithm in corner cases.
As next steps, we would like to proceed our experiments by further elaborating on question 1 and tackling question 2 from Section 2. To do so, we plan the following experiments:

- Repeat the experiment on a real-world data set (instead of the synthetic one used).
- Repeat the experiment on standard benchmarks (TPC-H) with different scaling factors.

Help from the SOC lab with finding a real-world data set would be appreciated. Ideally, this data set would already be available in a star schema.

References

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[2] D. Barth. An update on amazon s3. `http://developer.amazonwebservices.com/connect/servlet/KbServlet/download/254-102-510/amzns3_6-14-06.pdf`.

[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *In Proc. Conf. on Usenix Symposium on Operating Systems Design and Implementation*, pages 205–218, 2006.

[4] J. Cieslewicz and K. A. Ross. Adaptive aggregation on chip multiprocessors. In *VLDB*, pages 339–350, 2007.

[5] S. Manegold. *Understanding, Modeling, and Improving Main-Memory Database Performance*. PhD thesis, Universiteit van Amsterdam (UvA), 2002.

[6] A. Zeier, M.-P. Schapranow, and C. Tinnefeld. Impact of column-oriented main-memory databases on enterprise applications, October 2009. `http://epic.hpi.uni-potsdam.de/pub/Home/MatthieuSchapranow/091015_BI123_SAP_TechEd09_Impact_of_Column-Oriented_Main-Memory_Databases_on_Enterprise_Applications.pdf`.

IDS Alert Correlation using In-Memory and Multi-Core

Sebastian Roschke
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam
sebastian.roschke@hpi.uni-potsdam.de

Feng Cheng
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam
feng.cheng@hpi.uni-potsdam.de

Christoph Meinel
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam
meinel@hpi.uni-potsdam.de

Abstract

Intrusion Detection Systems (IDS) have been widely deployed in practice for detecting malicious behavior on network communication and hosts. The problem of false-positive alerts is a popular existing problem for most of IDS approaches. The solution to address this problem is correlation and clustering of alerts. To meet the practical requirements, this process needs to be finished as fast as possible, which is a challenging task as the amount of alerts produced in large scale deployments of distributed IDS is significantly high, due to the deployment of IDS sensors in Cloud computing and open network designs (e.g., SOA). We identify the data storage and processing algorithms to be the most important factors influencing the performance of clustering and correlation. We propose the utilization of memory-optimized algorithms and a column-oriented or In-memory databases for correlation and clustering in an extensible IDS correlation platform. The utilization of the column-oriented database, an In-Memory Alert Storage, and memory-based index tables leads to significant improvements on the performance. The platform can be distributed over multiple processing units to share memory and processing power, i.e., it can make use of both paradigms: multi-core and In-memory. The efficiency of the proposed platform is tested by practical experiments with several alert storage approaches and different simple algorithms.

1. Project Description

The problem of false positive alerts is a well known problem for many IDS approaches [14]. Suboptimal patterns or insufficient thresholds for pattern-based and anomaly-based IDS approaches are the main reasons for a huge number of false-positive alerts. By deploying the IDS sensors in a distributed environment, the number of false positive alerts increases as a single event may be detected and reported multiple times by different involved sensors. The popular solution to address this problem is correlation and clustering of alerts. To improve the efficiency of clustering and correlation, several techniques have been proposed [3]. As high security requirements of networks need real-time reporting of attacks and malicious behavior, the performance of these techniques is a critical factor. In particular in large scale distributed IDS (DIDS), providing high-performance correlation and clustering remains to be a difficult challenge, due to a huge amount of alerts. The performance of alert correlation can be improved by using table indexes in main memory for hyper alerts [13], i.e., clusters of alerts with the same properties. Furthermore, correlation in real-time is often based on filtering and clustering of alerts to hyper alerts [12], which reduces the number of processed alerts significantly. The approach reaches a correlation rate on the order of $100,000$ alerts per second based on the massive reduction of alerts by clustering them. However, a general approach that can handle a higher amount of alerts per second without the need of alert reduction is considered to be useful.

In-Memory and column-based databases are usually used for costly analytical processing of huge amounts of data [5]. A column-based organization of the database improves analytical operations, which often consist of comparison of all values from a single or multiple columns. As described in [6], database systems can benefit from the use of main memory. In-memory and column-based databases are suitable for future computing paradigms, such as multi-core systems, which are supposed to have more CPUs and a huge amount of main memory. By storing a database

in the main memory, analytical operations can be processed in parallel by several CPUs with direct access to the main memory. There are many implementations for column-based database systems, such as MonetDB [7, 8], which is an open-source database system for high-performance analytical operations, e.g., Online Analytical Processing (OLAP), Geographic Information Systems (GIS), XML Query, text and multimedia retrieval. MonetDB often achieves a significant speed improvement for SQL over other open-source systems, e.g., MySQL[9] or PostgreSQL[10]. The general benefits of in-memory and column-based databases can be useful to correlation and clustering of IDS alerts.

In large scale deployments of DIDS, huge number of alerts are produced in a short time. To fulfill the challenging task of fast correlation and clustering, we identified the data storage and processing algorithms to be the most important among several other influential factors for the performance. To improve those factors, we propose the utilization of improved algorithms using a memory based index table and the deployment of In-Memory or column-oriented databases for correlation and clustering. A flexible correlation system is needed, which synchronizes, unifies, and analyzes all the security related events produced by the integrated sensors. To meet these requirements, we implement an extensible IDS correlation platform in this paper, which consists of several *Correlation Handlers*, a unified *Alert StorageController*, a unified *AlertUpdateController*, and a *RequestController* for visual presentation and network communication. The utilization of a *Column-oriented Database* and an *In-Memory Alert Storage* in connection with *Improved Algorithms* using *Memory-based Index Tables* for correlation and clustering lead to significant improvements of the performance. Different types of correlation modules can be easily integrated and compared on this platform. A new plugin concept for *Receivers* provides flexible integration of various sensors and additional IDS management systems. The platform can be distributed over several units to share memory and processing power. The IDMEF [4] standard is used to represent and exchange the alert information. A standardized interface is designed to provide a unified view of result reports for users. The efficiency of the proposed platform is evaluated by practical experiments for various alert storage approaches and simple algorithms, within local or distributed deployment of the platform.

We believe that research in the area of IDS and network security as application for multi-core and In-memory based platforms can provide new paradigms for conducting security. Correlation and clustering is currently only done in a limited way using filtered data sets. Using the multi-core and In-memory platforms, it might be possible to do correlation and clustering on an unfiltered data set. Thus, it might not be necessary to fine tune (e.g., exclude certain detection rules) the IDS sensors anymore, as the correlation and clustering can do meaningful reasoning on all alerts in a short time. Furthermore, we expect correlation and clustering services offered in the Cloud. A flexible and extensible correlation platform can provide the foundation work for a new paradigm in security.

2. Results and Achievements

During the last few month, we have been able to achieve multiple results by using the system in the Future SOC infrastructure. We fully implemented the correlation platform with In-Memory and column-based DB support. We conducted practical experiments using an IDE integrated performance measurement framework. Apart from the practical achievements, we have been able to publish papers on the correlation platform [1] and started research on a complex correlation algorithm using attack graph data and environmental information for IDS correlation [2]. Some results are summarized in the following subsections.

2.1 Implementation

The architecture of the correlation platform is shown in Figure 1. It consists of four major components: the *Controller*, the *RequestController*, the *AlertUpdateController*, and the *AlertSourceController*. The *Controller* is responsible for starting the platform with all other controllers, loading the *Correlation Modules*, and initializing the *In-Memory Alert Storage* and the *Index Tables*. The *AlertSourceController* provides the interface to the source storage, e.g., a row or column-oriented database. By using *Source Adapters* as plugins, the *AlertSourceController* provides an easy and flexible mechanism to connect different types of databases. The *UpdateSourceController* provides the interface to a running IDS management system. The plugin concept of *Alert Receivers* offers the possibility to connect different management systems as well as different IDS sensors directly [11]. The *RequestController* provides the interface to the *Correlation-Frontend*, which is responsible for presenting the correlation results to the user. The *Correlation Modules* implement the different correlation algorithms. Each module is working independently based on a data source, which is either the database provided by the *AlertSourceController*, or the *In-Memory Alert Storage* created at startup. The *In-Memory Alert Storage* can also be disabled if needed, as it is memory consuming. Furthermore, each module can update and read the *Index Tables* to cluster and correlate the alerts. We deployed the prototype of the correlation platform a FutureSOC VM (1 CPU, 4 GB Ram) and developed multiple features to improve performance and usability:

- Snort alert generator that generates IDS alerts using a network description

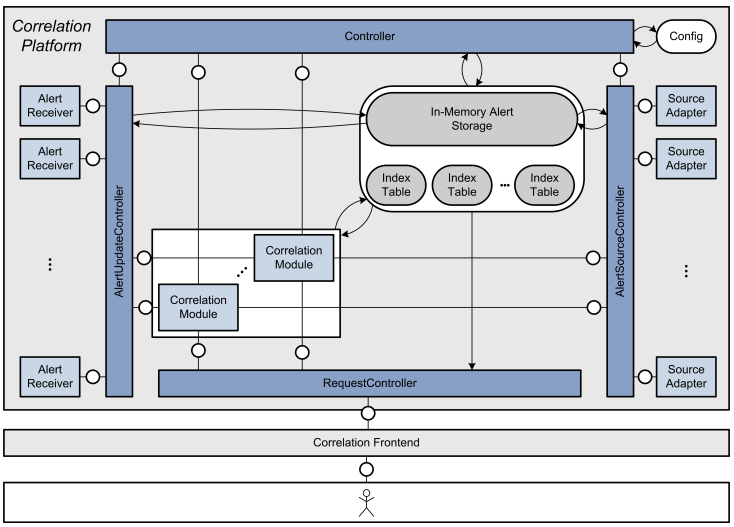


Figure 1. IDS Correlation Platform

- Dynamic module loading by uploading a module through the frontend
- Usability and performance improvements for the GUI
- Integration of environmental data into the platform that can be used for correlation (network and system descriptions, attack graph data)
- Development of the information pool concept that enables access to correlation results and environmental information for all correlation modules

2.2 Experiments

We conducted practical experiments to perform a performance analysis on the different approaches. The analysis has been performed on a system with two *Intel Core(TM)2 Duo* CPUs running on $1.4GHz$ with a cache size of $3,072kB$ each. The system possesses $2GB$ RAM and Solid-State-based hard drive with a size of $128GB$. The running operating system (OS) was a Gentoo Linux. The time and memory consumption are measured by the Eclipse Test & Performance Tools Platform Project (Eclipse TPTP). Therefore, the measurements have not been done on the FutureSOC VM. The following additional software packages were used for the experiments:

- MySQL version *5.0.70*
- MonetDB Release *Aug2009-SP2*
- Sun Java Development Kit (JDK) version *1.6.0.15*
- Snort version *2.8.3*

We used an alert data set collected by running a Snort IDS sensor connected to the backbone of the university network. The sensor generated $1,391,520$ real alerts in six month of runtime. Based on this data set, we generated three databases: one with $43,485$ alerts (called DB_1), one with $695,760$ alerts (called DB_2), and one with $1,391,520$ alerts (called DB_3). DB_1 and DB_2 are a part of the basic data set and have a chosen size (i.e. exactly $1/16$ and $1/32$ of the original data set). We created these databases based on *MySQL* and *MonetDB* to conduct the experiments. We measured the inserts within the creation process, the clustering, and the correlation based on the improved simple algorithms using a row-oriented database (MySQL), a column-oriented database, and the *In-Memory Alert Storage*. The index tables are used in connection with the *In-Memory Alert Storage* to clearly separate memory-oriented and database-oriented approaches.
By comparing the results, we conclude that a row-oriented database shows poor performance for clustering and correlation. It handles between $2,034$ and $2,784$ alerts per second for the simple clustering, and between $3,018$ and $5,156$ alerts per second for the simple correlation. However, with approximately $16,000$ alerts per second, the creation of the database is as fast as the creation of an *In-Memory Alert Storage*, which can be important for an IDS management system that needs to insert many alerts in a short time frequently. The column-oriented database shows better performance for correlation and clustering. It handles between $4,714$ and $17,177$ alerts per second for the simple clustering, and between $49,018$ and $102,792$ alerts per second for the simple correlation. With approximately 63 alerts per second, an important problem is the poor performance for database creation, which makes it difficult to use as main alert database for IDS management. The best performance is shown by the *In-Memory Alert Storage*. It han-

dles between $153,188$ and $779,672$ alerts per second for the simple clustering, and between $261,367$ and $725,600$ alerts per second for the simple correlation. By using index hash tables, the aggregated clustering can handle between $153,188$ and $5,288,716$ alerts per second. A major issue is the memory consumption of this approach. It uses 144 MB, 884 MB, and 1.6 GB of memory for an *In-Memory Alert Storage* with the databases DB_1, DB_2, and DB_3.

3. Future Work

Within the next few months, we prepared the correlation platform for further research and experiments. We would like to work towards our vision with the following steps:

- Conduct performance experiments on the improved platform
- Research on storage mechanisms: fast access and persistence
- Research on correlation algorithms that are using environment information and attack graphs
- Research on multi-core support for IDS correlation
- Research on collaboration approaches for IDS correlation
- Research on visualization techniques for correlation results

References

[1] S. Roschke, F. Cheng, Ch. Meinel: *A Flexible and Efficient Alert Correlation Platform for Distributed IDS* In: Proceedings of 4th International Conference on Network and System Security (NSS'10), IEEE Press, Melbourne, Australia, pp. 24-31 (September 2010).

[2] S. Roschke, F. Cheng, Ch. Meinel: *Using Vulnerability Information and Attack Graphs for Intrusion Detection* In: Proceedings of 6th International Conference on Information Assurance and Security (IAS'10), IEEE Press, Atlanta, United States, pp. 104-109 (August 2010).

[3] R. Sadoddin, A. Ghorbani: *Alert Correlation Survey: Framework and Techniques*, In: Proceedings of the International Conference on Privacy, Security and Trust (PST'06), ACM Press, Markham, Ontario, Canada, pp. 1-10 (2006).

[4] Debar, H., Curry, D., Feinstein, B.: *The Intrusion Detection Message Exchange Format, Internet Draft*, Technical Report, IETF Intrusion Detection Exchange Format Working Group (July 2004).

[5] H. Plattner: *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 (SIGMOD'09), ACM Press, Providence, Rhode Island, USA, pp. 1-2 (2009).

[6] P. A. Boncz, S. Manegold, and M. L. Kersten: *Database Architecture Optimized for the New Bottleneck: Memory Access*, In: Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99), Edinburgh, Scotland, UK, pp. 54-65 (1999).

[7] MonetDB: WEBSITE: *http://monetdb.cwi.nl/* (accessed Nov 2009).

[8] P. Boncz: *Monet: A Next-Generation DBMS Kernel for Query-Intensive Applications*, PhD Thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, 2002.

[9] MySQL: WEBSITE: *http://www.mysql.com/* (accessed Nov 2009).

[10] PostgreSQL: WEBSITE: *http://www.postgresql.org/* (accessed Nov 2009).

[11] S. Roschke, F. Cheng, Ch. Meinel: *An Extensible and Virtualization-Compatible IDS Management Architecture*, In: Proceedings of 5th International Conference on Information Assurance and Security (IAS'09), IEEE Press, vol. 2, Xi'an, China, pp. 130-134 (August 2009).

[12] Tedesco, G. and Aickelin, U.: *Real-Time Alert Correlation with Type Graphs*, In: Proceedings of the 4th international Conference on Information Systems Security (ISS'09), Springer LNCS 5352, Hyderabad, India, pp. 173-187 (2008).

[13] Ning, P. and Xu, D.: *Adapting Query Optimization Techniques for Efficient Intrusion Alert Correlation*, Technical Report, North Carolina State University at Raleigh, 2002.

[14] Northcutt, S., Novak, J.: *Network Intrusion Detection: An Analyst's Handbook*, New Riders Publishing, Thousand Oaks, CA, USA (2002).

Report on the Project: Enlargement of the Search Domain of the tele-TASK Portal

Maria Siebert, Franka Moritz and Christoph Meinel
Hasso-Plattner-Institute
University of Potsdam
maria.siebert|franka.moritz|christoph.meinel@hpi.uni-potsdam.de

Abstract

The tele-teaching project tele-TASK of the Hasso-Plattner-Institute is a large-scale lecture recording system with increasing popularity amongst the students and external people. Searching within the easily generated and quickly growing video archive is not an easy task. All available meta data sources have to be taken into account and additional meta data will be created by processing the available data streams. Therefore the search domain is quite large and powerful engines are needed to compute search request in nearly real-time. Several experiments with the futureSOC server in connection with the tele-TASK project, their results and drawbacks are described in this paper and the next steps are outlined.

1 Introduction

This report explains the experiments conducted with the futureSOC server on account of the tele-teaching project tele-TASK by the chair of internet technologies and systems of the Hasso-Plattner-Institute in Potsdam. At first we will introduce the tele-TASK project and explain bottlenecks of the current system. Second the two parts of the project - the tele-TASK recording system and the tele-TASK web portal are explicated in more detail. Next we describe preliminary experiment results that limit the current system and motivate why a futureSOC project was applied for and what our vision about the outcome of the futureSOC experiments is. Furthermore the actual experiments that we conducted are explained and their drawbacks and problems stated. We conclude this report with an outlook to further research we plan with the futureSOC server.

2 Project Description

The tele-teaching project tele-TASK is being developed since about 2002 [4]. Tele-TASK is a shortcut for Tele-teaching Anywhere Solution Kit and was originally the name for the recording system. In the last years the recording system has been continuously developed. The good usability and growing demand for lecture recordings has resulted in a requirement of a management solution for all lectures recorded. Therefore the tele-TASK portal (http://www.tele-task.de) was developed, which serves the purpose to manage, organize and provide the recordings.
The tele-TASK project provides a large amount of lectures and videos recorded at the HPI. It is widely adopted and enjoys great popularity among our students. Also a large number of external persons are using the portal. More than 3000 lectures can be found in the portal held by about 750 lecturers. These lectures are grouped in over 200 series. For a better usage of the lectures, they are split into smaller scenes, to create handy video clips which can also be used through the iTunes U portal. Some details of the two parts of the tele-TASK project, the recordings system and the portal, are explained in the next paragraph.

2.1 tele-TASK Recording System

The tele-TASK recording system is a unique innovation which received the Einstein Award in 2002. Its outstanding feature is the synchronous recording of one audio and two video streams [4]. This enables its users to record the speech of a lecturer, a video of the lecturer as well as a screencast of everything that happens on the presenters laptop, like the digital presentation or the presentation of a website, simultaneously. The capturing system allows to create a huge number of recordings of lectures without much effort.

2.2 tele-TASK Portal

The tele-TASK portal (http://www.tele-task.de) is a web portal programmed in Python, based on the Django framework. It is implemented using MySQL as database backend. The database currently consists of over 100 tables. The last version of the portal has been online since one year ago, having thousands of users each month. The portal is set up using a plug-in architecture [6]. This means that all modules are

Figure 1. Advanced Search Form in tele-TASK Portal

loosely coupled and the portal can easily be extended with further functionality. Further meta data sources and search options can therefore be integrated easily so that the complexity of the portal may increase rapidly without disturbing the structure and the core functionalities.

2.3 Research and Development

The current development focus of the portal is to gain more meta data and use it for better usage experiences. The following structured meta data are available or will be available soon:

- Manually inserted meta data for the media, like title, description, lecturer, etc.
- Data from the users: Enhancing the portal with community functions like rating or tagging [1], new meta data is created.
- Data from the audio stream: Using speech recognition, the spoken text can be extracted [2].
- Text from the slides: Using OCR tools, the text of the slides can be parsed [3].
- Usage data from the logs: Evaluating the logs more data about related content can be found [7].

In order to find the most appropriate content, all available meta data should be used to browse through the content and extract all content items which fulfil the required search query. Afterwards filters can be applied to reduce the number of search results. Also it is possible to order the content according to different criteria.
Through the new plug-in architecture of the portal [6], it is an easy task to enhance the search functions with new filters and order by criteria [5] to make the new meta data searchable. But these possibilities cannot be utilized, because of constraints connected with the resources available. For generating complex search request, big database table joins are required.

3 Preliminary Experiment Results

In the development version of the portal, the following search requests where tested:

- Search for all titles: Searching inside all titles of the different content types, like series, lecture, scene or even playlist is possible in the actual development version, but needs a lot of ram and cpu power, increasing the search time and sometimes resulting in crashes of the system. That is why it is not possible to use this function in the live application.
- Usage of user data: Ordering the search result by rating results is possible in the development version, but results in the same performance problems.
- Search for audio: Searching for content of the audio stream is available in the development version, but due to a lack of generated data, could not be used in the live application so far. In future

a lot of data, which has to be analyzed in search requests, will be generated and therefore needs a lot of cpu power and ram to be processed.

- Other search of meta data will be added soon, enhancing the search domain even more. These queries however cannot be executed with the hardware utilized at the moment as cpu and memory resources are exhausted.

4 Vision

The search amongst the broad variety of learning content is one of the main challenges for learners in our society nowadays, where content can be produced so easily. Therefore it is a key task for us as providers of tele-teaching content to also offer a most convenient and comprehensive search to all users of the tele-TASK portal.
Before starting this project it was not possible to include a search amongst all currently available meta data into the search query, let alone all meta data that will be generated in the near future. This constraint exists due to ram and cpu limitations of the server the database is currently running on. Therefore only limited search queries can be executed and only simple joins can be permitted for all users at present. This constraint shall be erased with the help of the new in-memory technology.
A first application of the new server technology that we suggest is to keep the whole database of the tele-TASK portal in the memory. Using this new set-up we expect all currently implemented search query, filter and order options to be executable at once without performance problems. As second step the meta data that will be generated in another part of the tele-TASK project will be included as query options in the search. A much larger query set will have to be executed in this option. Performance issues will be taken into consideration as well, but we still expect a fast and comprehensive processing of combined joins.
Finally it is also planned to use more search request for the detection of related content to the shown data. This should result in a recommendation system for the users of the portal.

5 Experiments

The inital idea for an application for a futureSOC experiment platform was to store the database for the live tele-TASK portal in the ram of the futureSOC server, in order to test the performance of the search functionality in the portal under real circumstances. The idea for the setup was to store the database of a clone of the tele-TASK web portal on the futureSOC server and access it by the Django instance running on the Apache web server (see figure 2). But a remote database access from our live system is not possible, as we will

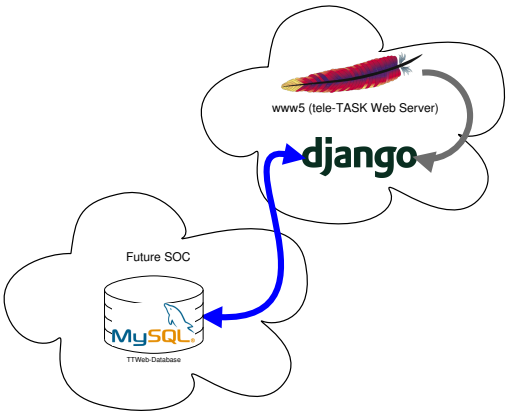


Figure 2. Idea of Communication Between tele-TASK Webserver and future-SOC

explain in more detail in section 5.2. Therefore our first desired experiment setup could not be tested.
A second option was to store the whole tele-TASK system including database and framework on the futureSOC server and then do a performance test on the system. But of course those performance tests would work by inserting search queries via the search interface of the tele-task website. Accessing the tele-TASK website that is hosted on the futureSOC via webbrowser is not possible with the current security setup of the futureSOC either, which is why that option could not be tested as well.
A third option was the pre-processing of meta data sources in order to provide data base tables to the system that can be queried much faster. This pre-processing would involve a transformation of for example audio data in ready-made keywords. Once all keywords for existing content have been generated, the database tables can be indexed and the query performance can be improved a lot with this procedure. This is an option we would like to explore in the future. But with the first trial of this method we also experienced difficulties.
A first test that was executed on the futureSOC server was to process audio transcription on that machine instead of processing it on a standard desktop computer or another server. Unfortunatly we had to make the experience that no increase in speed could be observed during that test. More to the contrary the test took nearly double the time than on a standard computer. Therefore we possibly still need to work on the parameters of our futureSOC virtual machine and find out which settings work best for our purpose. The setup we used initially will be explained in the following section.
Afterwards we tried to use the futureSOC server for pre calculation of search data. Therefore we combined the existing meta data in one index database table and

searched this table. This approach allowed us to search for search terms in one table column using indices provided by MySQL. Using that approach we were able to search all searchable database fields in one search request. It costs the time, which is needed to combine all these fields in the search field.
The problem with that approach is, that the calculated data could not be used on the live system, because on that system the data is changing frequently and therefore must be updated often. That is why we used the futureSOC for testing the combined algorithm, but have to write a final algorithm, which works on the live systems web server.

5.1 Used FutureSOC Lab Resources

On the futureSOC Lab we tried to use a similar setup than what we have on the tele-TASK server in terms of the operating system. That was why we decided to use a Debian Lenny package. Furthermore we put much hope in operating the tele-TASK database in the RAM of the futureSOC server. The initial RAM we asked to be set up for us therefore was 4GB which is enough to hold the database in this cache.

CPU	4 Cores
RAM	4GB
HDD	50GB
OS	Debian Lenny
Database	MySQL

Table 1. Overview of FutureSOC Resources Used in the Experiment Setup

The CPU power was adjusted according to the initial requirements for out database trials and was therefore set to 4 cores. In order to achieve more increase of speed for the third experiment option with the preprocessing of meta data sources, we would need to increase the number of CPUs and test if this would result in the desired speed-up.

5.2 Problems During the Experiment

A major disadvantage for our ongoing experiments is the mode of accessing the server. Our idea to utilize the futureSOC resources that we received in order to store the whole tele-TASK database in ram memory cannot be successful with this kind of setup (see figure 2). The reason for that is that we are not able to run our portal on its regular server while just remotely accessing the database on the futureSOC server. The access restrictions only allow us to access the server via a VPN connection with using a personal login and a certificate. That is why it cannot be used from our remote server which hosts the Django framework with the tele-TASK web server. Also an access of the website via a web browser is not possible.
Therefore we were not able to use the futureSOC server for live tests with a larger number of users and real database queries, which would be one desired experiment for our project goal. This means that we will only ever be able to use this setup for single tests with simulated users and not a real-life performance test.

6 Outlook

For being able to simulate the real user behaviour on the futureSOC server we are collecting search request by users. It is also planned to use the apache web server log for evaluating the behaviour of the users after sending a search request. With this data the search function can be expanded with new algorithms.
It is planned to provide additional data to the user, when he starts a search request, like proposing related search terms and doing spelling corrections. With the help of semantic data, which can be gained from different data sources like DBpedia, the search function can become even more powerful. With this data it is possible to decide what the user is looking for and to handle synonyms and homonyms of search terms.
Having knowledge about the search behaviour of the user will help here too, because the interest of a user can influence his search goals. For example, someone who is interested in human computer interfaces wants to get different results for the search term interface than a user interested in software architecture.

References

[1] F. Moritz, M. Siebert, and C. Meinel. Community Tagging in Tele-Teaching Environments. In *2nd International Conference on e-Education, e-Business, e-Management and E-Learning (to appear)*, Mumbai, India, 2011. IEEE Computer Society.

[2] S. Repp and C. Meinel. Automatic Extraction of Semantic Descriptions from the Lecturer's Speech. In I. Press, editor, *Proc. 3rd ICSC*, pages 513–520, Berkeley, CA, USA, 2009.

[3] H. Sack. Automated Annotation of Synchronized Multimedia Presentations. In *In Workshop on Mastering the Gap: From Information Extraction to Semantic Representation, CEUR Workshop Proceedings*, Berkeley, CA, USA, 2006.

[4] V. Schillings and C. Meinel. Tele-TASK – teleteaching anywhere solution kit. In *Proceedings of ACM SIGUCCS*, Providence, USA, 2002.

[5] M. Siebert and C. Meinel. Realization of an Expandable Search Function for an E-Learning Web Portal. In *Workshop on e-Activity at the Ninth IEEE/ACIS International Conference on Computer and Information Science Article*, page 6, Yamagata/Japan, 2010.

[6] M. Siebert, F. Moritz, and C. Meinel. Establishing an Expandable Architecture for a tele-Teaching Platform. In *Ninth IEEE/ACIS International Conference on Computer and Information Science Article*, Yamagata, Japan, 2010. IEEE Computer Society.

[7] L. Wang. *X-tracking the Usage Interest on Web Sites*. PhD thesis, University Potsdam, 2009.

Workload Management for Main Memory Databases in Data Clouds

Jan Schaffner, Alexander Zeier, Hasso Plattner
Hasso Plattner Institute for IT System Engineering
University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

Tim Kraska, Michael J. Franklin, Michael I. Jordan, David A. Patterson
UC Berkeley
465 Soda Hall MS-1776
Berkeley, CA 94720, USA
{franklin,jordan,pattrsn,kraska}@berkeley.edu

Dean Jacobs
SAP AG
Dietmar-Hopp-Allee
Walldorf, Germany
dean.jacobs@sap.com

Abstract

In this report we give a formal description for the problem of assigning tenants to servers of an on-demand system, which is one of the key problems in our Future SOC Lab project. We will pose this problem as an optimization problem and omit database specifics and pose the problem in an abstract fashion, using the metaphor of assigning tokens to baskets (i.e., tenants to servers). We present a first set of greedy algorithms and describe how we used the computing resources of the Future SOC Lab in the process of experimentation. We outline the next steps for our project .

1. Introduction

For traditional data warehouses, mostly large and expensive server and storage systems are used. For small- and medium-sized companies, it is often too expensive to implement and run such systems. Given this situation, the SaaS model comes in handy, since these companies might opt to run their OLAP at an external service provider. The challenge is then for the analytics service provider to minimize total cost of ownership by consolidating as many tenants onto as few computing resources as possible, a technique often referred to as multi-tenancy.

The *Rock* project at the HPI [6, 4, 7] seeks to maximize throughput in a cluster of main memory column databases: The goal is to support the highest possible number of concurrently active users while guaranteeing hard service level objectives on end-user response times (e.g. "99 percent of all queries have to complete in less than 1 second"). We suggest different data placement strategies for deciding which tenants are co-located on which servers in order to minimize the number of servers when running a given number of users. This problem is at the heart of large-scale Internet services trying to minimize the cost of their data centers.

In this Future SOC Lab project we seek to compare several different placement strategies for Analytic Databases in a Cloud Computing environment. Rock uses an active/active load balancing scheme in the presence of multiple replicas. If a server goes down, the workload which was handled by this server is re-distributed to the servers holding the other copy of the tenants' data. The re-distribution of workload in the event of a server failure differs depending on how the tenant replicas are assigned to the servers in the cluster. Using the off-the-shelf replication capabilities offered by most modern databases would result on replicating the data on the granularity of a whole server. In doing so, all tenants appearing together on one server will also co-appear on a second server in the cluster. This technique is often referred to as *mirroring* (cf. Figure 1). The downside of mirroring is that in case of a failure all excess workload is re-directed to the other mirror. In doing so, the mirror server is

a local hotspot in the cluster until the failed server is back online. A technique for avoiding such hotspots is to use *interleaving*, which was first introduced in Teradata [8]. Interleaving entails performing replication on the granularity of the individual tenants rather than all tenants inside a database process. This allows for spreading out the excess workload in case of a server failure across multiple machines in the cluster.

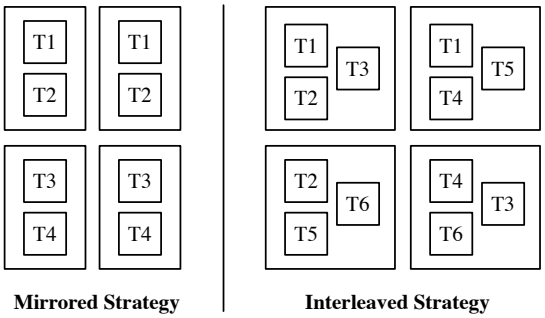


Figure 1. Example Layouts of Tenant Data

This report is structured as follows: Section 2 will formulate the above placement problem described in the previous paragraph as an optimization problem. Given the length restrictions of this report, we will omit database specifics and pose the problem in an abstract fashion, using the metaphor of assigning tokens to baskets (i.e., tenants to servers). In Section 3, we present a first set of algorithms for solving this placement problem and describe how we used the computing resources of the Future SOC Lab in the process of experimentation. In Section 4, we outline the next steps for our project. Section 5 concludes this report.

2. Problem Statement

Let $N = \{i_1, \ldots, i_{|N|}\}$ be a set of baskets and $T = \{t_1, \ldots, t_{|T|}\}$ a set of tokens. An assignment of tokens to baskets is given, as shown in Figure 2. All tokens have a radius $r(t)$ and a color $c(t)$, which are also known for all tokens. Each color occurs exactly twice (or, in other words, each token occurs exactly twice, which means that there are $2|T|$ tokens in an assignment). The problem to be solved is to find a new assignment of tokens to baskets with the following properties:

- No color occurs twice in the same basket.
- The sum of all token radiuses in each basket does not exceed a fixed upper bound $cap(i)$.
- The sum of all token radiuses should have a similar value for all baskets.
- Any two colors appearing together in one basket should preferably not appear togther in a second basket.

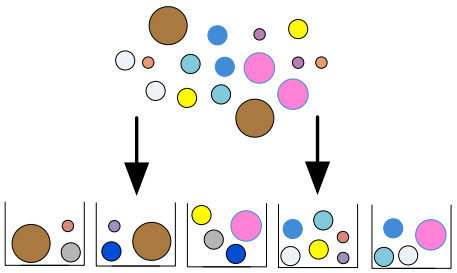


Figure 2. Assignment of tokens to baskets

2.1. Formal Description

To denote the assignment of tokens to baskets we define a decision variable y as follows:

$$y_{t,i}^{(k)} = \begin{cases} 1 & \text{if token } t \text{ is in basket } i \\ 0 & \text{otherwise} \end{cases}$$

$$\text{with } k \in \{0,1\},\ t \in T,\ i \in N$$

The index k identifies the first and the second token of the same color, respectively. An assignment of tokens to baskets Y', the number of baskets N, and the set of tokens T with their respective radiuses and colors is given as an input for this problem. The goal is to devise an algorithm which calculates a new assignment Y (i.e. the transformation $f : Y' \rightarrow Y$).
Token radiuses increase and decrease as time progresses, although they are fixed for any given instance if the problem. The model is that a new instance of the problem is created each time one or more changes in token radius have been observed. We are looking for an algorithm that—when invoked—balances the sum of token sizes across all baskets and, at the same time, tries to minimize color co-appearance across the baskets. An optimal assignment in this respect would be such that no two colors appearing together in one basket appear together in any other basket at the same time. To do so, the algorithm moves tokens between the baskets.
The sum of all radiuses $r(s)$ of the tokens in a given basket i is defined as

$$R(i) := \sum_{t \in T} \sum_{k=0}^{1} y_{t,i}^{(k)} r(t),\ i \in N.$$

To describe how good (or bad) a given assignment of tokens to baskets is w.r.t. color co-appearance, we introduce a penalty function $P(i)$. It is computed from the perspective of an individual basket and is defined

as the sum of all token radiuses of co-appearing tokens in one of the other $N-1$ baskets. Since this value depends on which of the other $N-1$ baskets is chosen as a partner in this binary comparison, $P(i)$ is defined relative to the partner yielding the maximum value:

$$P(i) = \max_{j \in N} \left(\sum_{t \in T} \sum_{k=0}^{1} y_{t,i}^{(k)} y_{t,j}^{(1-k)} r(t) \right)$$

$$\text{with } i, j \in N, i \neq j$$

Constraints

1. A valid assignment Y must contain each color exactly twice.

$$\sum_{i \in N} y_{t,i}^{(0)} + y_{t,i}^{(1)} = 2, \ \forall t \in T$$

2. No basket must contain any two tokens of the same color.

$$y_{t,i}^{(0)} + y_{t,i}^{(1)} \leq 1, \ \forall t \in T, \ \forall i \in N$$

3. The sum of all token sizes in a basket i must be less than or equal to the capacity of the basket.

$$R(i) \leq cap_i, \ \forall i \in N$$

Objective Functions

1. All baskets shall be balanced w.r.t. aggregate token size (in addition to constraint no. 3, which only specifies an upper bound for the sum of all token radiuses within one basket R_i). One way of progessing towards a similar value for the different $R(i)$s is to minimize their variance:

$$\min \ Var(R(1), \ldots, R(|N|))$$

2. Co-appearances of colors in the baskets shall be minimized:

$$\min \ \sum_{i \in N} P(i), \ \forall i \in N$$

3. In addition to minimizing the co-appearance penalty per basket (the previous optimization goal), all baskets should have a similar penalty. One way of progessing towards a similar value for the different $P(i)$s is to minimize their variance:

$$\min \ Var(P(1), \ldots, P(|N|))$$

2.2. Possible Extensions

For simplification, the following extensions of the problem will be left out when devising a first set of solutions. They will, however, be considered at later stages in this project.

Varying number of baskets For the problem stated above we assume a fixed number of baskets N. It might be the case that the observed changes in token size create a situation in which the current number of baskets is not sufficient for finding an assignment of tokens to baskets such that none of the above constraints is violated. Given such a situation, the algorithm is allowed to create a new basket. Similarly, when the changes in token size result in a situation where all of the above constraints could be satisfied using fewer baskets, then the algorithm can empty a basket by migrating its tokens to other baskets and delete the basket. It would also be conceivable to trade-off the the number of baskets against the balancing of the $R(i)$s as well as the values and the balancing of the $P(i)$s.

Minimizing the number of migrations So far we have not imposed a limit on the number of movements of tokens between baskets necessary to provide the transformation $f : Y' \rightarrow Y$. However, it would be conceivable to try to minimize the number of movements in this sequence. It would also be interesting to study how fast the other goal functions converge to an optimal value, varying the number of allowed migrations in f.

3. Preliminary Results

In this section, we will present three greedy placement algorithms, which make simplifying assumptions. We will then describe our implementation of a brute force solver, which enumerates all possible combinatorial placements for a given number of servers and tenants. Due to space restrictions, we omit an evaluation of our greedy placement algorithms against the optimun solutions obtained from brute force placement, which is thus left for the next report.

3.1. Greedy Algorithms

All algorithms presented in this section are concerned with placing a set of tenants to servers such that the requirements discussed in Section 2 are met. The work imposed on a server by adding a tenant to a server (called *workload*) is known a priori for all tenants. Therefore, the tenants can for instance be ordered by workload size prior to placement by the algorithms. As a ground rule, all layouts must be able to absorb one server failure at any point in time.

3.1.1 Naïve Best Fit with Mirroring

This algorithm servers as a baseline against which the other algorithms will be evaluated. It varies the number of servers during execution and starts out with a two empty servers to which it adds the a single copy of the first tenant to be placed. It then continues to fill

up this server with more tenants until the capacity of the server is reached, at which point a new server is started. To determine whether a server i has enough capacity to fit a tenant t, the algorithm simply evaluates the condition $R(i)+r(t) < \frac{cap(i)}{2}$. The reason for allowing only half of a server's capacity to be used is the rule that all assignments produced by the alorithm must be able to absorb a server crash in the cluster in the sense that there is enough capacity to serve all tenants. In the presence of multiple servers, a given tenant is assigned using a best fit strategy on this condition (which behaves similar to first fit since the number of servers is increased only in case no server has enough capacity to fit an additional tenant). After all tenants have been assigned, the servers are mirrored thus doubling the number of servers and providing each tenant with a second copy. The number of servers produced in the end by this algorithm is assumed the worst-case by the other algorithms, in the sense that they try to find assignment such that a server crash can be absorbed but less servers are required.

3.1.2 Best Fit Considering Penalty Only

This algorithm takes the co-appearance penalty into account, which was defined in Section 2. Unlike the previous algorithm, this variant is run with a fixed number of servers, and the copies of each tenant are placed in a single step. Since there might be multiple valid assignments satisfying the constraints of the problem for different numbers of servers, the algorithm in run multiple times while varying the number of servers N. The range across which N is varied is determined by the result of the previous algorithm, naïve best fit, taking the number of servers it produces as an upper bound for N. The lower bound for N is set to half the numbers of server produced by naïve best fit.
Within a run with a given number of servers, the algorithm performs what we call a local bruteforce. In each step when trying to place a tenant, it tries all possible assignments of the two copies of the tenant to be placed to the available servers.
One possible way of enumerating all possibilities two assign the two copies of a tenant to N servers will now be briefly discussed. Given that both copies of a tenant must be on different servers and that both copies are equivalent, all possibilities to assign a single tenant to two servers can be done by creating a matrix with dimensions $N \times N$. The elements of this matrix are defined as follows:

$$(i,j) = \begin{cases} 1 & \text{if } i < j \\ 0 & \text{otherwise} \end{cases}, \; i,j \in N$$

An example for $N = 4$ servers:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

The number of possibilities for assigning two copies of a tenant to N servers is thus $N(N-1)/2$. We refer to all 1-valued elements of this matrix as possible *configurations* of a tenant.
Section 2 stated three different (possibly conflicting) objectives for a tenant placement algorithm. Without resorting to multi-criterial optimization methods (which we plan to do in future work), we need to build a single objective function combining all the objectives we want to cover. This algorithm combines the three objectives into one by defining the capacity of a server using both $R(i)$ and $P(i)$. The basic idea is that the failure of a single server is masked, in which case the following constraint must be met:

$$R(i) + P(i) < cap(i) \; \forall i \in N$$

In addition to that, we define the objective function by minimizing the difference of value for $P(i)$ before and after assigning the tenant to be placed. Since the co-appearance function is sensitive to tenant co-location, this difference must be computed for both servers to which a copy of the tenant is to be assigned:

$$Min \; P(i) + P(j) - P(i)' - P(j)'$$

$P(i)'$ denotes the penalty before assigning tenant t to server i. If multiple servers have the same value w.r.t. this objective, the first two servers are picked.
Since all tenants are ordered by workload prior to invoking the algorithm, this variant achieves strong interleaving of the large tenants. Given this strong interleaving and the fact that actual server load is not taken into account, this algorithm leaves servers ununtilized as N progresses towards its upper bound. An example of this is omitted due to space restrictions.

3.1.3 Best Fit Considering Penalty and Workload

This algorithm is similar to the previous variant, although it also takes server load into account. The objective funtion is defined as follows:

$$Min \; P(i) + R(i) + P(j) + R(j)$$

Since this algorithm strives to evenly spread load across servers it utilizes all available servers.

3.2. Brute Force

For being able to compare how far the values of the objectives described in Section 2 deviate from the optimal values, we developed for enumerating all possibilities to assign T tenants to N servers. As we have already established in Section 3.1.2, there are $N(N-1)/2$ possibilities to assign two copies of one tenant to N servers. The number of combinatorial possibilities to assign T tenants to N servers is thus:

$$\left(\frac{N(N-1)}{2}\right)^T$$

We encode each combination using a sequence of length T with each element of the sequence being an integer encoding a tenants *configuration* (cf. Section 3.1.2):

$$(a_0, \ldots, a_{T-1}),\ 0 \leq a_i \leq \frac{N(N-1)}{2} - 1$$

This sequence obeys lexicographic ordering in the sense that the next combination can by computed simply by incrementing the rightmost element of the sequence which is smaller than $(N(N-1)/2)-1$. Brute force enumeration can nicely be parallelized: In our implementation, we use recursion to generate multiple ranges of the sequence with equal sizes. Each of those ranges is then enumerated in parallel. Our implementation uses Scala actors [5] and fits well with the large number of available cores on most Future SOC Lab resources.

4. Next Steps

All algorithms presented so far fall into the category of *offline algorithms*, in the sense that all tenants and their workload is known a-priori. Much more challenging (and also more relevant) is to deal with changing workloads given a current assignment of tenants to servers. To capture changes in workload we are currently in the process of obtaining production logs from a large on-demand service capturing changes in request rates over time. In an *online* setting, we need to make incremental changes to the existing assignment, since there is a high cost associated with re-organization during normal operations. Instead, we want to investigate algorithms which make local changes to the placement, such as in Amazon Dynamo [3] or Google Bigtable [2]. In doing so, we plan to minimize the number of tenant migrations required to transform one layout into another.

Other directions of future work include creating relaxed variants of the complete multi-dimensional optimization problem. One approach to try are genetic algrithms, which can be set up such that they incorporate the number of migrations as part of the model. Another promising approach for relaxation is semidefinite programming [9].

A more long-term goal is to create an algorithm which leverages statistical machine learning techniques to model workload spikes [1]. We plan to build a predictive algorithm which learns based on production logs and request rate changes in the past. Training machine learning algorithms is often compute intensive and could be done using HPI Future SOC Lab resources.

5. Conclusion

In this report, we have given a formal definition of the tenant placement problem, which we have identified as one of the key problems in workload management for in-memory database clusters. We presented three greedy algorithms for solvong this problem, although we leave their evaluation for the next report. We have illustrated how we use resources of the HPI Future SOC Lab to solve the compute-intensive aspects of this study. While we focused on the algorithm design aspect of our project in this report, this project also has a large experimental component, which will involve running tests on Future SOC Lab resources. Specifically, we need to experimentally explore how the differences for the objectives of our multi-criterial optimization problem translate back to actual throughput in the Rock clustering framework.

References

[1] P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*, pages 241–252, 2010.

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.*, 26(2), 2008.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 205–220, 2007.

[4] M. Grund, J. Schaffner, J. Krüger, J. Brunnert, and A. Zeier. The effects of virtualization on main memory systems. In *Sixth International Workshop on Data Management on New Hardware*, 2010.

[5] P. Haller and M. Odersky. Actors That Unify Threads and Events. In *Coordination Models and Languages, 9th International Conference, COORDINATION 2007, Paphos, Cyprus, June 6-8, 2007, Proceedings*, pages 171–190, 2007.

[6] J. Schaffner, D. Jacobs, B. Eckart, J. Brunnert, and A. Zeier. Towards enterprise software as a service in the cloud. In *Second IEEE Workshop on Information Software as Services in Conjunction with ICDE 2010, Long Beach, CA, USA*, 2010.

[7] J. Schaffner, J. Krüger, S. Müller, P. Hofmann, and A. Zeier. Analytics on historical data using a clustered insert-only in-memory column database. In *16th International Conference on Industrial Engineering and Engineering Management (IEEM), Beijing, China*, 2009.

[8] DBC/1012 Database Computer System Manual Release 2. Teradata Corporation Document No. C10-0001-02, 1985.

[9] L. Vandenberghe, S. P. Boyd, and S.-P. Wu. Semidefinite Programming and Determinant Maximization. In *Encyclopedia of Optimization, Second Edition*, pages 3375–3380. 2009.

An Architecture-Aware Compaction Process

Jens Krueger, Martin Grund, Alexander Zeier, Hasso Plattner
Hasso-Plattner-Institut
{firstname.lastname}@hpi.uni-potsdam.de

Abstract

In mixed workload environments with OLTP[1] and OLAP[2] queries, the process of periodically compressing the data becomes essential when using read-optimized compressed in-memory column stores. To achieve the best analytical performance usually separate systems are used for transactional and analytical purposes and the modification from the transactional system are applied in batches into the analytical system, where they are stored highly compressed and optimized for fast reading.

A next generation enterprise application database, suitable for a mixed workload has to keep up with the ongoing data changes in transactional systems while still providing support for fast analytical queries as depicted in [3]. For optimal query execution in this future system data is stored compressed. In order to also support transactional queries we keep a small write optimized store, which gets periodically compressed and merged with the read optimized store. A compaction process accomplishes this transformation. The process has to encode data efficiently while affecting running queries as little as possible. We plan to study different implementation of this compaction process, improve it to benefit from the concretely used hardware by optimizing the cache behavior, prefetching mechanisms and thread level parallelism and data level parallelism.

1. Introduction

We propose a compaction process that works online without downtime for the transactional system and has a minimal effect on the performance of running queries. For an transactional enterprise application it is crucial that the current data is always available and in a 24/7 environment system down-times are not acceptable and as such the merge must be run online without interference to other components. Traditionally, loading new data into an OLAP system is done offline — new data is not immediately visible and/or the system is not available during load. This is due to the fact that during load cleansing, reconciliation, aggregation, and even transformations are executed. Those operations are very expensive and even require own business logic. The importance of the merge is intensified in an enterprise system with hundreds or even thousands of tables that need to be scheduled for merge so that the overall system performance is kept optimal.

Applications like real-time stock level calculation, price calculation and online customer segmentation will profit from up-to-date data. A combined database for transactional as well as analytical workloads saves ETL costs and reduces the level of indirection between the different systems in the enterprise environment. However, we do not advocate a complete bonding of OLAP and OLTP systems. The requirements of data cleansing, system consolidation and very high selectivity queries cannot be met with our system approach and require additional systems. Our approach can significantly increase the value of OLTP systems by closing the described abstraction gap and level of indifference.

2. Architecture

For our evaluation we use HYRISE, an in-memory compressed vertical partitionable database engine.

The main storage type in HYRISE is a read-optimized column-oriented table that uses dictionary encoding on each column. The dictionary is a vector of all distinct values in the column and the offset of a value in the vector is its corresponding value id. HYRISE stores each value id in the document vector bit-compressed with only the number of bits needed to store the maximum value id. This results in an improved compression ratio and still guarantees fixed length value ids. Inserting in such a compressed persistence is as complex as inserting in a sorted column like shown in [2], because the whole compression has to be rebuild. A common approach to still achieve good update performance, is a technique called *differential updates*. A small and for fast writes optimized second storage stores the difference to the read optimized storage [4], thereby decreasing the read performance. To avoid the problem of constantly rebuilding the compression for every modification, all modifications are accumulated and merged from time to time.

[1] Online Transactional Processing
[2] Online Analytical Processing

3. Merge Description

We assume the architectural and design choices stated in [1]. The read optimized store is dictionary encoded and keeps its dictionaries in a sorted order. Furthermore bit compression and valid bit vectors are used. The write optimized store uses the same techniques with the difference that the dictionary is not sorted, so that new values can be appended without reorganizing the table. Additionally, a CSB+ tree is maintained for the write optimized dictionary for enabling binary searches and sorted iterating for the merge.

An unoptimized merge algorithm is outlined as pseudocode in Algorithm 4.1, consisting of two steps.

First, the dictionaries are combined into one resulting sorted dictionary and a value id mapping is created. This is achieved independently from the document vectors, by iterating simultaneously over the dictionaries. The dictionary iterators enable a sorted iteration over the dictionary values by leveraging the CSB+ tree for the write optimized dictionary. The current values of both dictionaries are compared in every iteration, the smaller value is added to the result and the corresponding iterator is incremented. In the case that both values are equal the value is only added once and both iterators are incremented. In order to be able to later update the document vectors to the newly created dictionary, every time a value is added to the merged dictionary the mapping information from the corresponding old dictionary to the merged one is also added to the mapping vectors. In 1) and 2) one of the two dictionaries is already worked off and empty, so that the values can be added directly without comparisons.

Second, the values from the two document vectors are copied into a combined document vector and the mapping is applied on the fly.

The merge process runs asynchronously. At the beginning of the merge the table is locked and an empty write optimized store is created. As soon as the merge has finished, the lock is released and all write access queries go against the newly created write optimized store.

When the merge of the stores is finished, the merged version is committed by locking the table and switching the old read and write optimized parts with the new merge read optimized version. During the merge, the process consumes additional resources (CPU and main memory), but should affect running queries as little as possible.

Merge by Example Figure 1 shows the content of a sample table. The table is internally structured in one read optimized and one write optimized part and consists of one column. The read optimized part is outlined at the top, whereas the write optimized part is located at the bottom of the figure. On the very left, the logical view of the table is outlined, internally divided into read and write optimized parts. Externally, the separation into read and write optimized parts is not communicated. Each storage is dictionary encoded and consists of one attribute vector and one dictionary. However, in the current implementation the dictionary of the read-optimized part is order-preserving, while the write-optimized dictionary is order-indifferent. Both attribute vectors are bit compressed, which is not pictured in the figure. The attribute vector references the actual values in the dictionary, which are variable length strings. We use country names as an example.

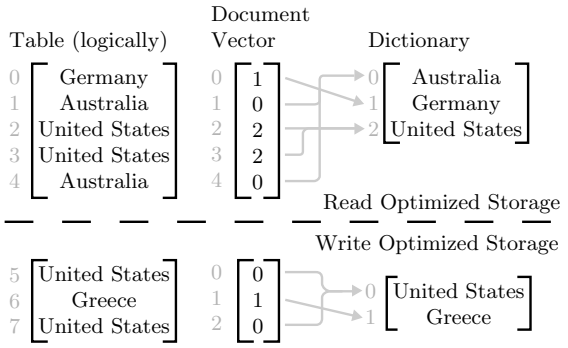
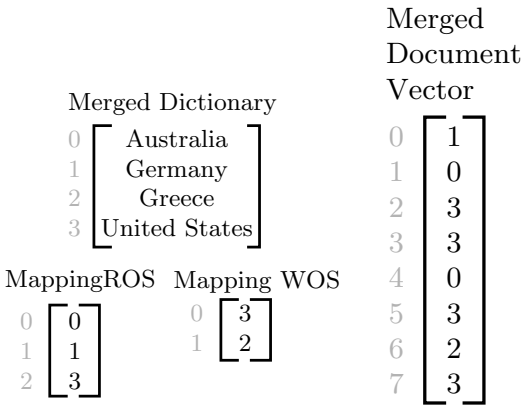


Figure 1: Merge by Example. Table Layout.



(a) Tail of the dictionaries

(b) Number of tuples. Cycles per Row.

Figure 2: Merge by Example. Step 1 and Step 2

After the last reorganization, three new rows have been inserted - "United States", "Greece", "United States". These inserts are accumulated in the write optimized store, which manages its own unsorted dictionary. Therefore, "United States" gets the value id 0 and "Greece" the value id 1. For the third inserted row, the value "United States" is already contained in the dictionary and therefore uses the existing id 0.

Figure 2 shows the results of the two merge steps. First, the dictionaries of the write and the read optimized parts are merged into one resulting dictionary. A mapping for each part's dictionary is created, translating the old value ids from the original dictionary to the corresponding value ids in the merged dictionary. In the example, "Greece" is inserted between "Germany" and "United States", so the mapping for the read optimized storage maps the old value id 2 (en-

coding "United States") to 3 in the merged dictionary. The mappings created in the first step, are used in the second step when the document vectors are merged. The document vectors are merged by appending the write optimized after the read optimized and translating the old value ids for the merged dictionary using the mapping vectors.

3.1 Parameters for Merge

The total costs of the merge process are determined by a) n_r the number of tuples in the read optimized storage, b) n_w the number of tuples in the write optimized storage, c) d_ι the intersection of the dictionaries and d) d_τ the tail of the dictionaries.

	Description
M	number of tuples in the ROS
N	number of tuples in the WOS
Md	size of the ROS
Nd	size of the WOS
u	compressed value size in the WOS (bits)
l	compressed value size in the ROS (bits)
U	uncompressed value size (bits)
i	intersection of the dictionaries (percent)
t	tail of the dictionaries (percent)

Table 1: Parameters influencing the merge.

The total number of tuples that have to be touched during the merge is the most dominant parameter for the costs of the merge process. For more tuples the costs per tuple increase logarithmic. The intersection d_ι of the dictionaries describes the amount of values that occur in both dictionaries, but will occur only once in the resulting dictionary. The bigger the intersection of the dictionaries the faster the merge, although the impact compared to the overall merge costs is fairly small. d_τ is the size of the so called tail. The tail of two dictionaries are all values of one dictionary that are greater than the last and therefore biggest value of the other dictionary. The tail values can be added as a bulk to the merged dictionary and do not have to be compared with values from the other dictionary which results in performance improvements. Therefor a big dictionary tail results in a faster merge. Similarly to the intersection of the dictionaries, the total impact is fairly small.

4 Status

The merge as described in the previous section was implemented in our main-memory research prototype HYRISE[1]. Currently we are optimizing the different steps of the algorithms based on the underlying latest Intel hardware. Special focus during this phase is to exploit special properties of the hardware by using e.g. Intel SSE operations. Furthermore, our goal is to evaluate the impact of the different parameters during the next phase of the project and to increase the performance of the merge process. The next steps are to parallelize the merge process and to observe the impact to the running system.

References

[1] M. Grund, J. Krueger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. Hyrise - a hybrid main memory storage engine (to appear). *PVLDB*, 2011.

[2] S. Héman, M. Zukowski, N. J. Nes, L. Sidirourgos, and P. A. Boncz. Positional update handling in column stores. In *Proceedings of the ACM SIGMOD International Conference on Management of Data 2010*, pages 543–554, 2010.

[3] H. Plattner. 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*, pages 1–2, 2009.

[4] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-Store: A Column-oriented DBMS. *VLDB*, 2005.

Algorithm 4.1: MERGE($main, delta$)

comment: Step1: Merge the dictionaries

$it_{main} \leftarrow main.dict.begin()$
$it_{delta} \leftarrow delta.dict.begin()$
$mapping_{main} \leftarrow Array()$
$mapping_{delta} \leftarrow Array()$
$merged \leftarrow Array()$

while not $it_{main}.end$ **and not** $it_{delta}.end$
do
- **if** $it_{main}.value < it_{delta}.value$
  - **then**
    - $mapping_{main}[it_{main}.pos] \leftarrow merged.size$
    - $merged.push\_back(it_{main}.pos)$
    - $it_{main}.increment()$
  - **else if**
  - **then**
    - $mapping_{delta}[it_{delta}.pos] \leftarrow merged.size$
    - $merged.push\_back(it_{delta}.pos)$
    - $it_{delta}.increment()$
  - **else**
  - **then**
    - $mapping_{main}[it_{main}.pos] \leftarrow merged.size$
    - $mapping_{delta}[it_{delta}.pos] \leftarrow merged.size$
    - $merged.push\_back(it_{main}.pos)$
    - $it_{main}.increment()$
    - $it_{delta}.increment()$

while not $it_{main}.end$
do
- $mapping_{main}[it_{main}.pos] \leftarrow merged.size$ (1)
- $merged.push\_back(it_{main}.pos)$
- $it_{main}.increment()$

while not $it_{delta}.end$
do
- $mapping_{delta}[it_{delta}.pos] \leftarrow merged.size$ (2)
- $merged.push\_back(it_{delta}.pos)$
- $it_{delta}.increment()$

comment: Step2: Merge the document vectors

for $row \leftarrow 0$ **to** $main.size$
do
- $valueId = main.getValueId(row)$
- $valueId = mapping_{main}[valueId]$
- $merged.setValueId(row, valueId)$

for $row \leftarrow main.size$ **to** $main.size + delta.size$
do
- $valueId = delta.getValueId(row)$
- $valueId = mapping_{delta}[valueId]$
- $merged.setValueId(row, valueId)$

return $(merged)$

Pro-Active Virtual Machine Migration in the HPI FutureSOC Lab

Peter Tröger, Matthias Richly
Hasso Plattner Institute
Prof.-Dr.-Helmert-Str 2-3
14482 Potsdam, Germany

Felix Salfner
Humboldt University
Rudower Chaussee 25
12489 Berlin, Germany

Abstract

Next generation technologies such as multi-core processors and large memory modules will result in tremendously increased computing power. However, this comes at a price: Due to the growing number of transistors and increased complexity, overall system reliability of future server systems is about to suffer significantly.

The HPI FutureSOC Lab project „Towards an Architectural Pattern for Pro-Active Virtual Machine Migration“[1] investigates an architectural blueprint for managing system dependability in a pro-active fashion. The approach is based on virtual machine live migration technologies and new failure prediction approaches.

Within this article, we present initial experimental results of the project, with a focus on virtual machine live migration capabilities.

1 Introduction

Achieving system dependability by employing replication in space and time is a traditional approach in distributed and cluster- based systems. Middleware implementations such as CORBA, .NET or DCOM have implemented various protocols to cope with transient and permanent faults above operating system level through redundant resources. High-performance computing (HPC) environments and large computing clusters were extended by similar redundancy concepts in the past, with special consideration of their tight integration and high number of components. Example analyses of large-scale HPC systems have shown a mean time between failures (MTBF) in the order of 6.5 to 40 hours, depending on installation maturity. Google for example experiences a MTBF in the order of one hour, although hidden from the users through fault-tolerant middleware and file systems.

With the advent of multi-core and many-core CPUs in commodity clusters such as blade centers, problems and challenges that once were of interest only to a small community of researchers and HPC users will now seriously impact the computing environment of tomorrow's average server environments.

One commonly agreed problem with smaller structural sizes, extreme memory increase (as in the FutureSOC lab with 2TB machines) and dynamic frequency / voltage scaling in the CPU is the overall dependability of hardware components. Industry reacted on this upcoming challenge - which is already well-known in the Exascale computing community - with a set of new fault monitoring and fault tolerance solutions.

One interesting layer of reactive fault tolerance are distributed virtualization-based failover clusters (see Figure 1). This machine-level approach adds to an existing set of solutions on hardware, firmware, operating system, middleware, and application level.

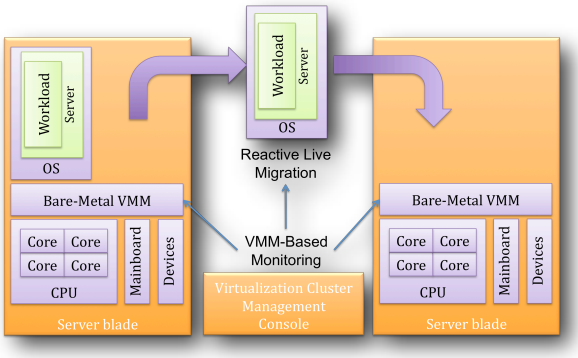


Figure 1: Reactive Live Migration of Virtual Machines

[1] The project is a cooperation of the *Operating Systems and Middleware Group* (Prof. Dr. Andreas Polze) at Hasso Plattner Institute, and the *Computer Architecture and Communication Group* at Humboldt University.

2 Approach

Our FutureSOC lab project investigates an approach where virtualization is used in a pro-active fashion for increased system dependability. Running virtual machines should be moved away from unreliable hosts, without disrupting running applications or the virtualized operating system. The given technical foundation is the recent availability of live migration capabilities in virtualization products. So far, these features only operate in a reactive fashion, leaving a relatively small time window for recovery activities in the face of a system failure. Current approaches are also solely based on performance counter threshold analysis at one level of the system stack (usually the VMM), and a subsequent reaction.

The pro-active solution for the migration decision is intended to rely upon a system health indicator, which is based on short-term online prediction of upcoming failures. Such anticipation requires the continuous monitoring and investigation of a system's state, in order to detect anomalies that indicate an upcoming failure. One key concept is the integration of status assessments from all system levels, in order to foster a maximum amount of system state information and domain knowledge.

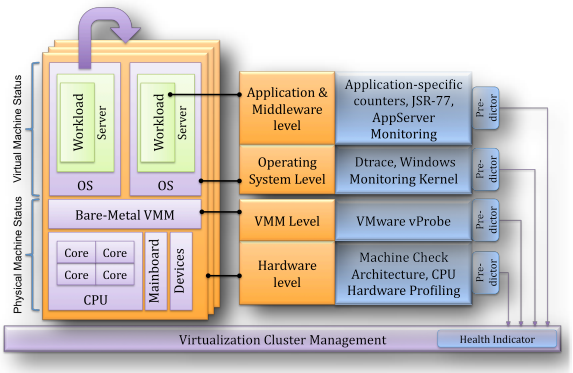


Figure 2: Pro-Active Virtual Machine Migration through Failure Prediction

Until today, such an approach would have had a serious performance impact due to the monitoring and prediction computation overhead. However, the identified problem source – complex and powerful parallel hardware in every server – becomes part of the solution here. In our proposed architecture, spare computational resources are utilized for all prediction activities. This allows combining existing approaches for system monitoring and failure prediction, given by our own earlier research results, into a new architecture for anticipatory virtual machine migration.

Our concept relies on the fact that a standard computer system can be divided into different layers of hardware and software, each with its own set of performance-related monitoring parameters. For each layer, we suggest the identification of relevant performance and / or health indicators that can be used in an online failure prediction facility. In contrast to threshold-based reactive patterns, this variable selection can be realized in a semi- automated training phase. Most prediction approaches demand this training phase in which a functional system is profiled for the 'normal' pattern of monitored events. The prediction approach then utilizes this data for online pattern matching. With the huge variety of different information sources, we focus our discussion on four major prediction layers for the system health indication – the hardware level, the virtual machine monitor level, the operating system level, and the application level.

It should also be noted that our investigation focuses on bare-metal virtualization only, since today's most capable live migration implementations are based on this model.

3 Initial Experiments

We started the FutureSOC lab project with an in-depth analysis of virtual machine (VM) migration capabilities, since this technology forms one of the cornerstones of our approach. We already developed a testing framework consisting of several load generators and test scripts which implement the measurement procedure. We applied the tests to VMware vSphere 4.0 and Citrix XenServer 5.6 so far, further tests for KVM and Microsoft Hyper-V are under way.

We focus on two metrics in the live migration investigation - the overall migration time and the blackout time of the VM. The migration time is the time from requesting the hypervisor to migrate the VM until it reports the successful migration. The blackout time is the period were the VM is not responsive to network I/O due to the migration. The latter can be significantly less than the migration time, because most hypervisors support minimizing the blackout time by ballooning, pre-copying or incremental copying the VMs state.

Load generation is done in the following ways:

- **CPU load generator (CLG)**: To generate a certain CPU load we use *burnP6* from the *cpuburn* suite in conjunction with *cpulimit*.
- **Locked pages generator (LPG)**: To increase the physical memory utilization of the VM, we created a new load generator tool that allocates a given amount of memory, writes random data to it and locks it in physical memory using the accordant system call (such as *mlock* in Linux). The migration is started when the tool is waiting (after the tool has allocated and locked the memory).
- **Dirty pages generator (DPG)**: Another tool was created to generate dirty pages during the migration by continuous memory writes. This increases the load for live migrations and therefore influences blackout time, because the VMs state continuously changes during the migration.

All tests were executed on hardware provided by the HPI FutureSOC lab: 2x FUJITSU PRIMERGY RX300 S5, with Intel Xeon E5540 @ 2.53GHz (4

cores + HT), 12GB RAM, and 2 Gigabit-NICs (migration link, external link).

The migration was performed with a Debian 5.0.4 "Lenny" virtual machine (kernel 2.6.26-2, 64 bit),a CentOS 5.5 virtual machine (same kernel) and a Windows Server 2008 R2 virtual machine. All machines were configured with one virtual CPU and a varying amount of (virtualized) physical RAM. In all cases, the virtualization guest tools / drivers were installed. Native operating system swapping was activated, but not aggressively in use due to the explicit limitation of the allocated amount of memory. Despite the CLG test cases, the virtual machine was having no CPU load from running processes.

The result graphs all show the average measurement results from 10 runs, together with error indicators in a 95% confidence interval.

3.1 VMWare VMotion - Selected Results

We ran a set of migration tests under different load conditions with an VMWare ESX 4.0.0 (build 208167) installation.

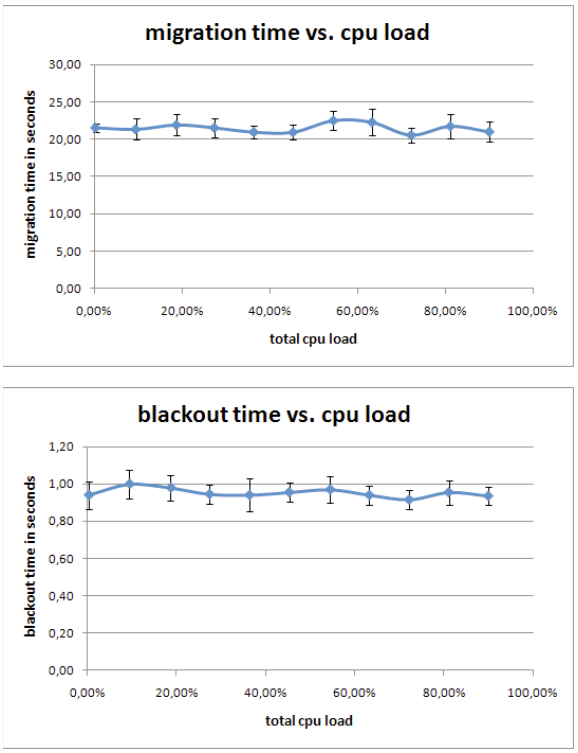


Figure 3: VMWare Migration and Blackout Time vs. CPU Load (CLG tool)

Figure 3 shows that the CPU load inside of the running virtual machine has no impact on the migration resp. blackout time of VMotion. This underlines the hypothesis that live migration performance is mainly influenced by memory copying efforts.

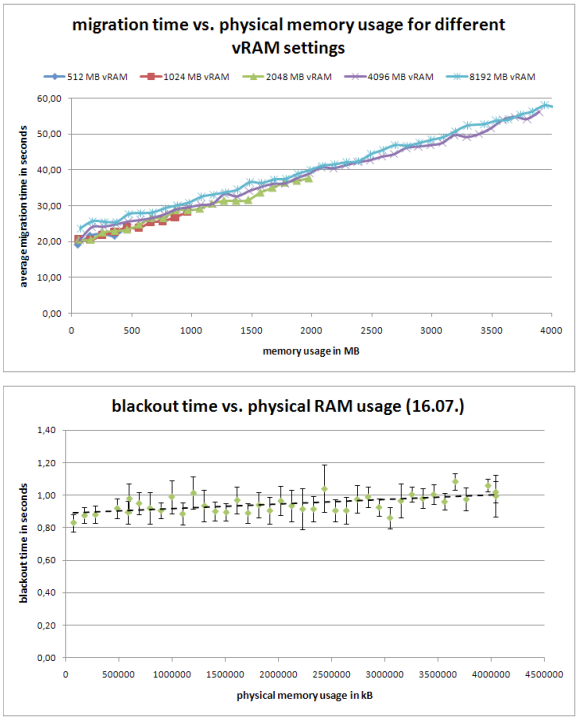


Figure 4: VMWare Migration and Blackout Time vs. Physical Memory Usage (LPG tool)

Figure 4 shows he average migration / blackout time plotted against physical memory utilization of the guest OS with different settings for the (virtual) RAM of the migrated VM. There is a linear functional correlation of the migration time and physical memory usage of the guest OS in VMWare. Blackout time is almost not influenced and still less than one second in the majority of cases with VMWare.

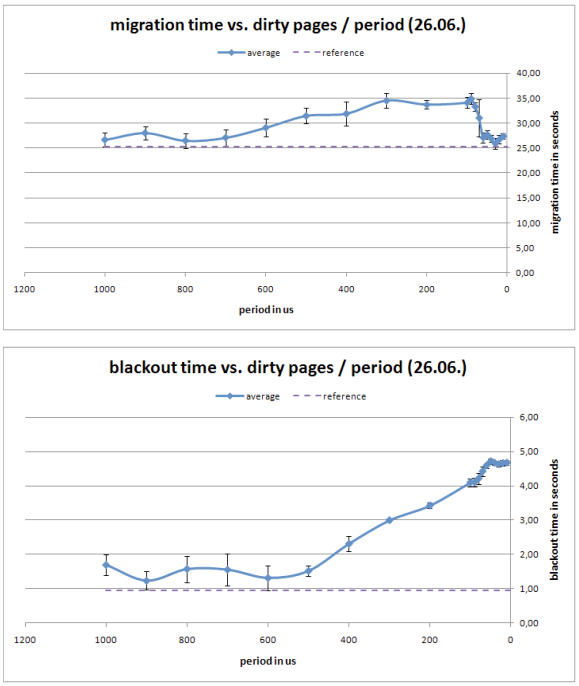


Figure 5: VMWare Migration and Blackout Time vs. Dirty Pages Rate (DPG tool)

Figure 5 shows migration & blackout time with a varying dirty pages generation rate. The rate is determined by the period value plotted on the x-axis in the charts (specifying the interval in which the tool is active) and the number of page changes within one period (constantly 10 for both charts). The reference value in the charts is determined by allocating memory that is not changed during the migration, but written only once at the beginning for triggering the lazy allocation mechanisms of the operating system.

As expected, both migration and blackout time are influenced by the amount of dirty pages generated during migration. The interesting aspect here is the impressive stable behavior of the VMWare solution, even under high memory modification load during migration. The dropping of the migration time with extremely high allocation periods is currently not understood, and might be either reasoned by measurement errors, or even by a transparent shift of the migration mechanisms on load.

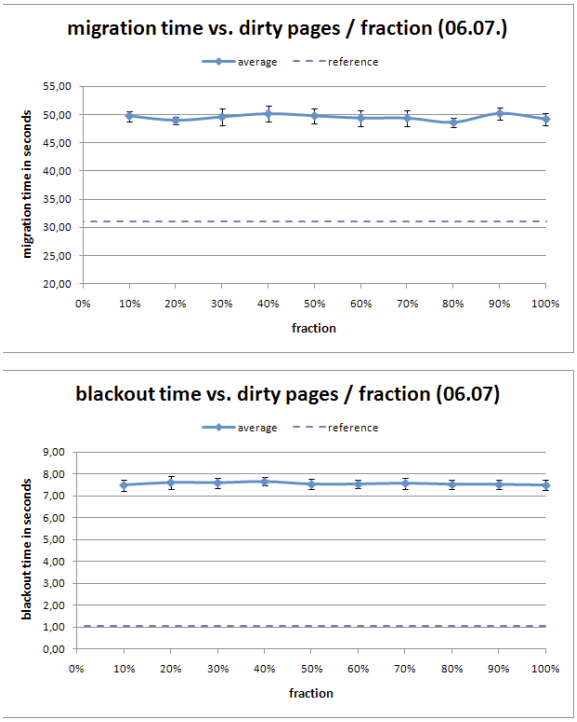


Figure 6: VMWare Migration and Blackout Time vs. Dirty Page Filling Rate (DPG tool)

Figure 6 proves the theory that live migration is operating on a page basis. In the experiment, the varying parameter was the *fraction value*. It determines how much of a block resp. page is changed in relation to the total block size (%). To clearly recognize potential effects, we chose a short period value, which causes the migration and blackout times to differ significantly from the reference value. The tests show that the fraction parameter has no influence on the measurement results. As expected, pages are copied on a whole, no matter how much of them is changed.

3.2 Xen Live Migration - Selected Results

The first set of comparative tests were performed with the Citrix XenServer 5.6 product.

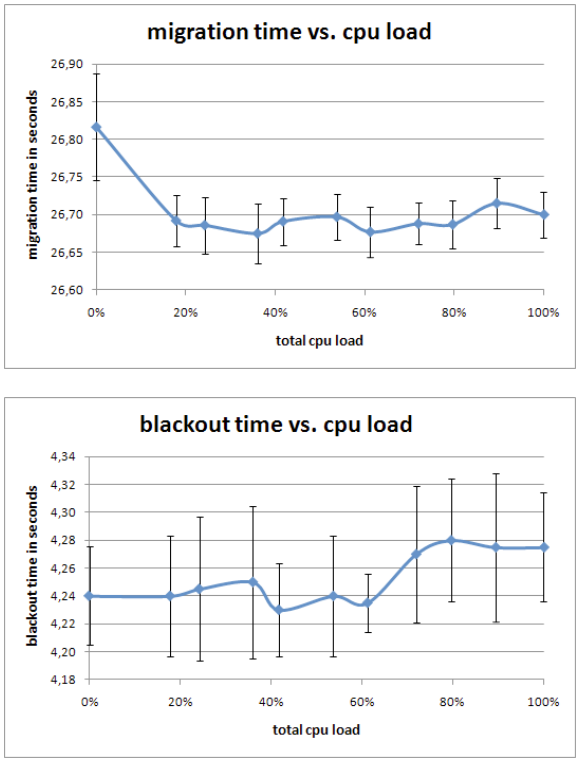
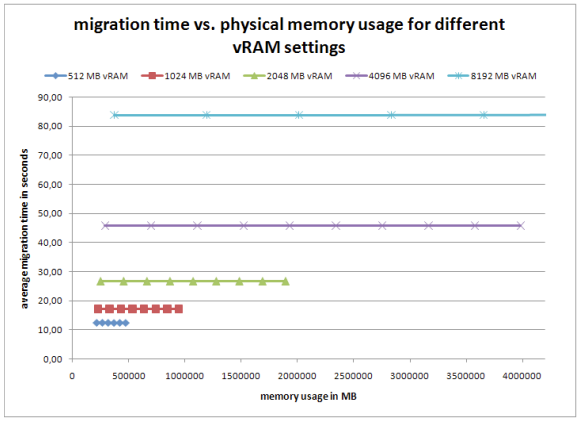


Figure 7: Xen Migration and Blackout Time vs. CPU Load (CLG tool)

Figure 7 demonstrates that in the Xen case, CPU load also does not influence migration performance. Migration time is similar to VMotion (~26s here vs. ~22s for VMotion). However, it must be noted that the blackout time is about 4 times higher than with VMotion (~4.3s vs. <1s). This is true in all tests, i.e. the minimal possible blackout time is always >4s for Xen.



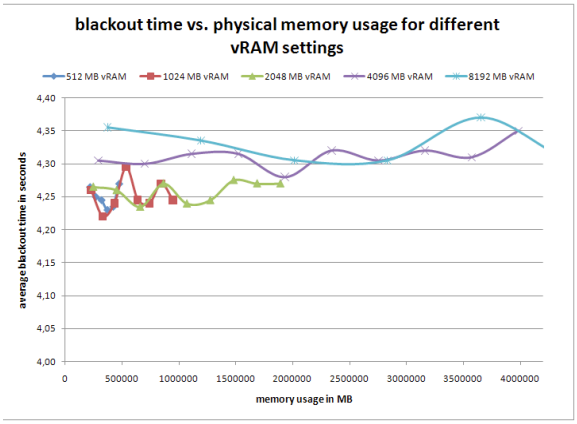


Figure 8: Xen Migration and Blackout Time vs. Physical Memory Usage (LPG tool)

Figure 8 shows he average migration time plotted against physical memory utilization of the guest OS with different settings for the (virtual) RAM of the migrated VM in the Xen case. The blackout time is not influenced, as for VMware. But in contrast to VMware VMotion, migration time is independent of memory usage of the guest OS for Xen. Our initial interpretation is that the Xen migration approach seems to copy all memory allocated for the VM, no matter if it is actually used or not. This results in a very stable but slow migration performance. Depending on the amount of guest memory being allocated, VMWare can truly outperform Xen in such cases.

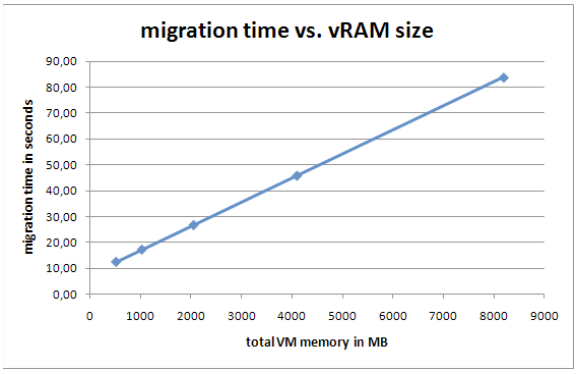


Figure 9: Xen Migration Time vs. Virtual Machine Memory Size (LPG tool)

Figure 9 proves that that there is a linear correlation between migration time and the configured VM memory size in Xen. The migration time was measured without any load on the guest OS.

For the dirty pages generation and filling rate experiments, the results were comparable to the ones obtained with VMotion.

4 Failure Prediction

A key idea of our project is to employ failure prediction at several layers of the system/software stack (c.f. Figure 2). There exists a plethora of prediction methods covering the hardware layer, operating system layer and application layer. However, existing work has very rarely, and if so very vaguely investigated the peculiarities of a system with bare-metal virtualization. Hence one of the questions to be answered in this project is to what extent existing failure prediction approaches can be applied to a virtualized environment. More specifically the question is whether there are faults and fault classes that can only be predicted (or at least predicted best) on the hypervisor level. In order to do so, we have identified three groups of input data to failure predictions that are only available at the VMM layer. This serves as a strong argument that VMM-layer failure prediction is a necessity for proactive virtual machine migration.

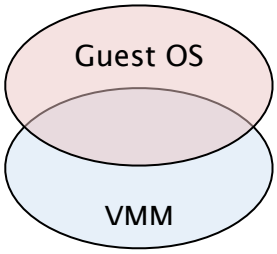


Figure 10: Overlap of Monitorable System Variables

In general, the situation can be characterized as follows: Some variables (types of input data) can only be monitored at the VMM layer, some are visible to both layers and some are visible only to the guest operating system (c.f. Figure 10). We have investigated three different types of input data:

- *EsxTop*, a monitoring tool available in the VMware ESX server
- SMART disk monitoring
- Intel Machine Check Architecture (MCA)

The management console on the ESX Server provides a tool called “esxtop” to monitor 120 system variables. We have classified those variables into the ones capturing properties that are only observable from the VMM, and the ones that are observable both from the guest OS and the VMM.

VMM-only	VMM and GuestOS
Physical CPU Util time Physical CPU load	Guest CPU % Idle
Physical memory Free MB Total swap space used	Guest VM swap space used
Physical disks queued commands	

Several failure prediction algorithms have been developed that perform disk crash prediction based on SMART values. However, such data is only accessible at the VMM layer as, for example VMware does not provide this information to guest OSes. This is especially important for predicting failures of local disks, which are used, e.g., as swap space. For this reason, SMART-based failure prediction is covered as one of the topics in the upcoming project work. A similar case exists with the correctable and uncorrectable error notification from Intels MCA feature, which is also solely accessible on Hypervisor level. Since a corrected bit does not lead to any malfunction of the system but can serve as a valuable input to an assessment of the system's health, such input is very valuable for failure prediction. However, at least VMware ESX does not supply guest virtual machines about such misbehavior.

5 Next Steps

The initial investigations have shown that stable virtual machine migration performance seems to be achievable. This is a cornerstone for continuing the overall work on a pro-active virtual machine migration environment, since the demanded time window for feasible predictions is given by the migration capabilities. We are currently investigating KVM and Hyper/V live migration features, in order to complete the performance picture of virtual machine live migration. As a second foundation, we are intensifying the work on hypervisor-level failure prediction, based on the achieved initial insights.

Rule based Business Matrix Processing (RBM) – Business Rules for real-time Process Management based on In-Memory Technology

Dr. Andreas Hufgard
Dipl.-Kff. Stefanie Krüger
IBIS Labs
Hufgard@ibis-thome.de
skrueger@wiinf.uni-wuerzburg.de

Prof. Dr. Rainer Thome
Chair in Business administration
and business computing
Joseph-Stangl Platz 2
97070 Wuerzburg
thome@wiinf.uni-wuerzburg.de

Abstract

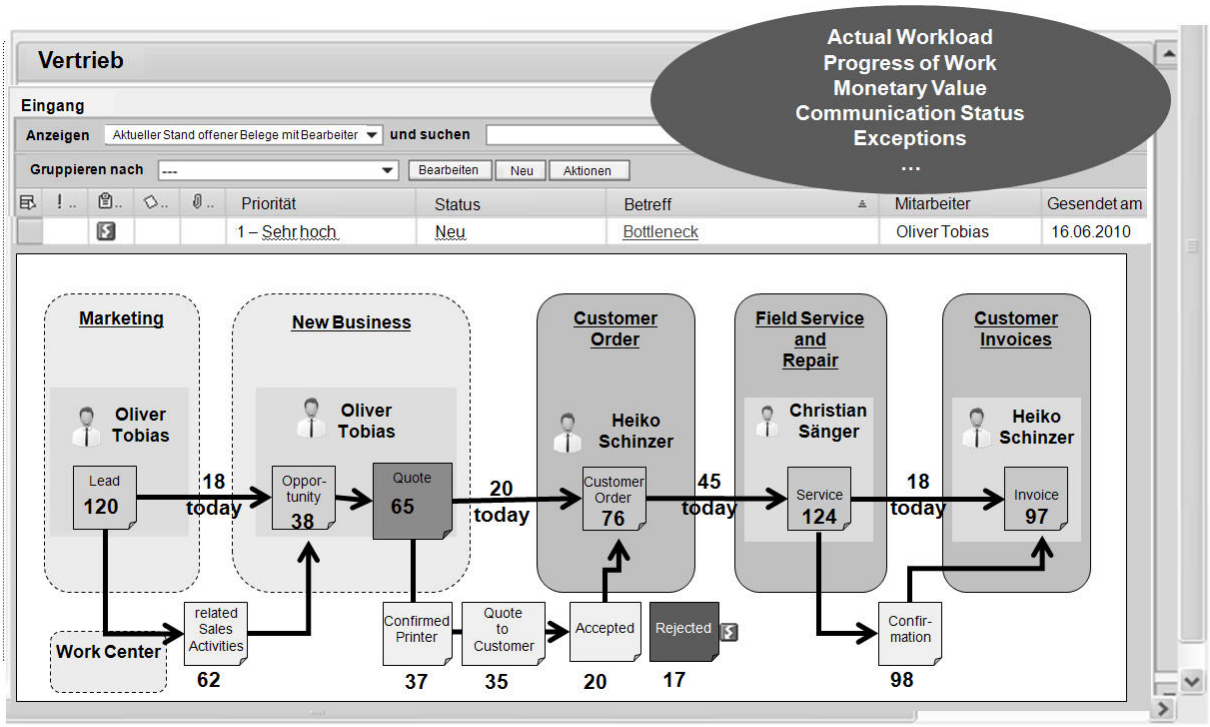
Rule based Business Matrix Processing is an approach to combine Consultative Information Technology with In-Memory Data processing. A holistic and real time Process Cockpit, a Recommendation Model + Engine and a Code of Good Practice with Business Rules have to be linked together to reach the target. As a result a substantiated decision making is possible requiring less effort and the productivity of employees can be increased dramatically.

Implementing a rule based business matrix processing and integrating it into an In-Memory based enterprise system like SAP Business ByDesign requires three separate elements that depend on one another.

1 Holistic and Real-Time Process Cockpit

For managers to be able to monitor a section of the process chain, they must have an overview and real-time model, like a cockpit, which gives them insight into the overall status and the dynamics of relevant business process transactions they are responsible for. This is the idea behind the Holistic and Real-Time Process Cockpit.

Figure 1: Holistic Real-Time Process Cockpit as a precursor of rule based Business Matrix Processing



In this context, holistic means

- it examines business process transactions in their entirety along business process chains, rather than analyzing individual cases out of context,
- it illustrates various perspectives of the business process that focus on management or decision-making tasks,
- simulation includes information on follow-up activities.

Exceptions in the process chain are made transparent by real-time displays of delays in work progress, anticipated bottlenecks, monetary values and other issues critical to prioritization.

In contrast to current monitoring approaches, real-time BPM enables new methods of management and analysis to leverage positive impact on processes while they are in progress. It should be possible to incorporate not only retrograde information from documents but also associated and progressive information from downstream sections of the process chain or from the environment; they can then be placed in context with the data in the business process transaction.

A further challenge lies in bringing together the semantic diversity of information stored in ERP systems in a way that is meaningful and that places it in interrelation with the environment and Internet databases.

2 Recommendation Model and Engine

The second element is the Recommendation Engine. Its structural model comprises a separate layer based on the data and process model and the adaptation and configuration model, respectively. A structure model, one component of a process cockpit, is responsible for rapidly furnishing and consolidating the relevant data fragments into "the bigger picture". The structure model can then provide notifications, recommendations and actions via an intelligent knowledge base. Plus, preventive steps or corrective actions, for instance, can be taken to positively influence the outcome.

The new type of engine being researched generates a modeled knowledge base of recommendations for good – or at least not incorrect – business practice, the code of good practice. Its rules need to show the user undesirable consequences, for example, or avoid these autonomously.

The anticipated edge gained through the speed of in-memory technology and the rapid association of information fragments can provide the employee at specific points in the process chain a kind of radar or braking assistant unprecedented in ERP systems.

Business matrix rule	**Protection**	**Productivity**	**Activity**
Action	Reject follow-up processing	Additional calculation/ proposal (in UI)	Trigger follow-up activities
Trigger	Profit margin of sales quote	New sales quote	New high-volume customer order created
Level of importance: • Notification (is) • Recommendation (should) • Action (must)	Profit margin of sales quote **should** exceed 10%. Profit margin of sales quote **must** be positive or must be approved by manager.	When a new sales quote is entered and customer defaults selected, values **should** be suggested automatically for other fields (pattern identification for customer and user).	Management **must** be informed if even one risk parameter (liquidity, currency, ROI) exceeds a critical threshold.

Figure 2: Classification of Business Matrix Rules (BMR)

This degree of foresight and effective protection must be based on active rules provided during runtime that immediately place occurring factors in context and trigger an appropriate reaction when a rule is violated. In contrast to configuration rules that only apply during configuration or reconfiguration, these rules are active at all times when employees trigger certain situations; or they work in the background as permanently active checks on undesirable constellations in the cockpit.

The Recommendation Engine requires fundamental architecture that enables interdependencies to be derived between events and information. An action must then be generated and forwarded via a user interface to the employee, or the correct subsequent actions must be triggered automatically.

The Recommendation Engine must also be capable of generating Business Matrix Rules that can be adapted to the company's structure and the current situation. This can be attained through self-learning networks or knowledge-based methods.

To reign in complexity and ensure parallel and independent development, the Recommendation Model's event and action rules must be based on process chain objects and structured according to these. In addition, the rules should include heuristic strategies, phase-by-phase evaluations and semantic concepts rather than merely declarative model structures. This will ensure that the rule model is able to evolve continuously.

3 Code of Good Practice – Reliable vs. Best Practice

In addition to developing the architecture and fundamental methodology, it's essential to create a code of good practice – to collect, organize and formulate a set of business rules structured by content. This is not a one-time activity. Similar to software development, it is an ongoing process that changes and develops in turn with legal stipulations, the latest trends and new insight or advancements in hardware and software. At the same time, it must be possible to gear these rules toward and configure them to company-specific requirements.

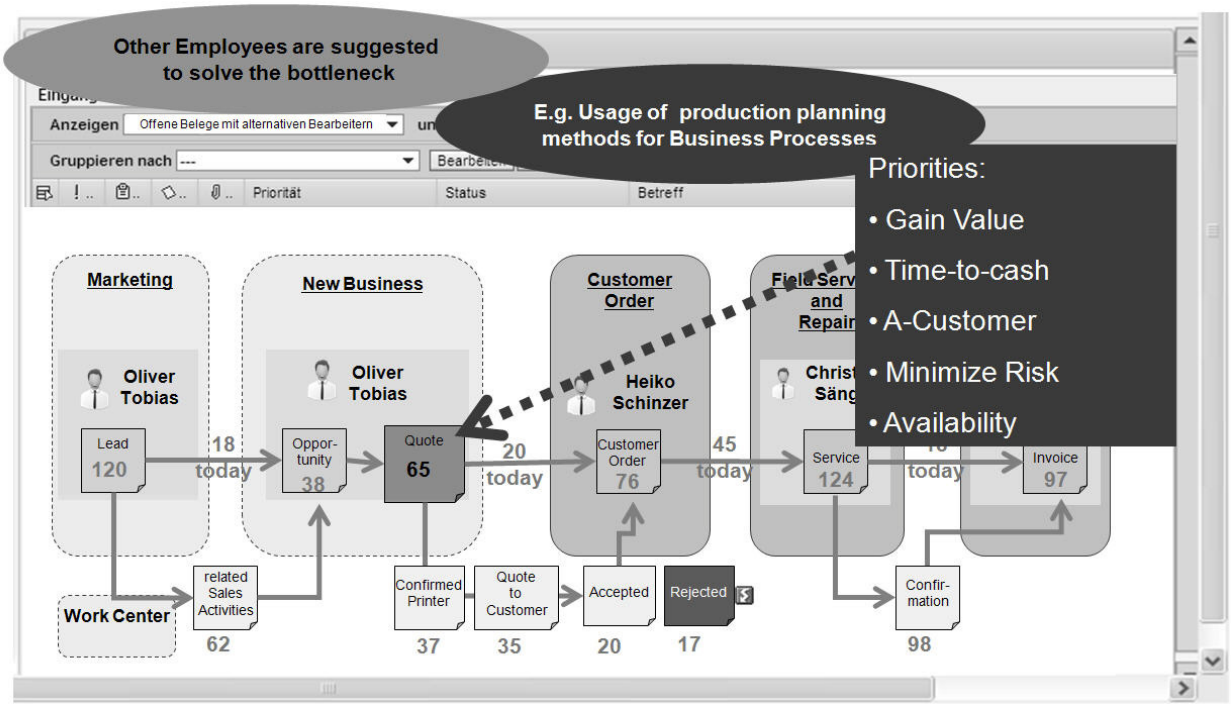
A code of good practice can be a collection and formulation of business process dependencies on the one hand, and it can assimilate and structure a company's or a consultant's rules, ideas and creativity, on the other. To better visualize the idea of a code of good practice, put yourself in the shoes of an employee who is executing a process in the system for the first time, or one who does so very rarely. Or take the viewpoint of a manager who needs to supervise her area as well as and efficiently as possible.

The employee has the disadvantage of knowing only the data shown on the screen, or will have executed a report on it at best. He has no further information that clues him in on the current goings-on unless the system or a colleague informs him of the interdependencies of his actions. The code of good practice takes on the role of an experienced colleague who can monitor the relevant interactions and tell the employee whether his actions will cause problems.

From a business viewpoint, a sales order may be influenced by several things. Earning a positive profit margin may be a priority. And there's always the question of whether there are foreseeable problems with the customer or with order processing – whether a product will be available when the customer wants it or whether a quotation for a specific quantity will be able to be filled, or whether the quantity fluctuates. The Recommendation Engine advises the employee, when in doubt, to offer more or less of a product or propose a different delivery date. It even suggests additional products the customer may want, based on customer data. It can also perform real-time priority changes for the customer; it can determine liquidity and cash flow, depending on the size of an order or on previous business contacts. If a customer is willing to accept longer delivery times, more flexible and cost effective alternatives can be added to the logistics chain.

On the one hand, there is the linking of information from each employee's part of the process to other sections in the chain, and on the other, there is the overview and analytical solving of decision-making issues and organizational difficulties at **manager level**. Management's task is to provide resources, identify and dissolve bottlenecks and set priorities. These kinds of issues are best solved by comparing and combining various scraps of information rather than through fixed, pre-formulated rules. For this reason, the manager requires a very specific type of rule – one that is capable of filtering relevant fragments of data and compiling them so decision-making can be recommended or made a high priority (e.g. like scorecards). The action to be taken may not be clear, but the constellation and the need for information is.

Figure 3: Manager view for prioritization



References

[1] A. Hufgard: *ROI von SAP Lösungen verbessern*. Galileo Press, Bonn, 2010.

[2] R. G. Ross: *Principles of the Business Rule Approach.* Addison Wesley, Boston, 2003.

[3] R. Thome; A. Hufgard: Continuous System Engineering. Discovering the Organisational Potential of Standard Software. 1st revised edition in English, Oxygon, München **2006**.

Software-Implemented Fault Injection in the HPI FutureSOC Lab

Peter Tröger
Operating Systems and Middleware Group
Hasso Plattner Institute
Prof.-Dr.-Helmert-Str 2-3
14482 Potsdam, Germany

Abstract

Software-implemented fault injection (SWIFI) is an established method to emulate hardware faults in computer systems. Existing approaches either extend the operating system by special drivers, modify the runtime environment, or change the application under test.

The FutureSOC project on software-implemented fault injection investigates new ideas for adding fault triggering and simulation as standard dependability assessment tool in modern server environments. The following report summarizes the results of the initial analysis phase.

1 Introduction

An in-depth study of fault tolerance capabilities of a system design can be accomplished at various stages including the conceptual design phase, system design, or prototype phase. In order to address the problem that true failures are a rare event during experimental runs, fault injection is frequently applied. System components are intentionally modified in order to trigger single component failures. Once a component is in an erroneous state, subsequent reactions of the system are monitored and evaluated in order to assess and quantify fault tolerance capabilities of the system under test.

Several approaches to fault injection have been proposed in the past. They vary by the layer where faults are injected and how the injection is performed. Major criteria for selecting an appropriate fault injection technique are the assumed fault model, the number of tests that have to be performed in a given time, controllability of the fault injection, and the number of changes necessary to implement fault injection in the tested system.

The representativeness of fault injection experiments is limited if the system under test has to be modified. In such cases, tests might not be performed under conditions similar to the run-time environment of the productive version. This can be a relevant aspect in safety-critical applications, such as control systems for critical infrastructures. Such systems frequently rely on certified software that must not be modified. Hence, the alterations necessary to implement fault injection should be minimal. A second limitation of most injection techniques is their non-portability across operating systems and/or hardware platforms.

In order to cope with portability and non-intrusiveness aspects of fault injection, our FutureSOC project works on novel approaches to fault injection that operates on or below the operating system level.

2 Software-Implemented Fault Injection

Software fault injection describes the approach of triggering hardware and software faults by programmatic modifications. Software fault injection can target different layers of software, such as the operating system, a runtime environment, or the application itself. Beside the generation of faulty software binaries (compile-time injection), most such approaches rely on the injection at runtime of the system. A special variation is software-implemented fault injection (SWIFI), which emulates hardware component outages by special software activities, such as changing registers and memory cells.

Our broader analysis of existing SWIFI approaches showed that they always modify system software to some extent. Either the tested application has to run in a special trace mode, the operating system has to be modified or extended by a driver, or the application itself has to be modified.

The new concept investigated by this FutureSOC project addresses the interference problem by establishing the fault injector as part of the lower layers in the hardware / software stack. Beside a traditional approach of adding fault injection capabilities as operating system kernel module resp. driver, the new idea is to add them as part of the computer firmware. This allows to leave operating system and application stack untouched, while still having the benefits of a SWIFI approach. Since many relevant server infrastructures - even in critical environments - rely on X86/X64 computers today, we focus on realizing this

approach with the modern hardware systems available in the HPI FutureSOC Lab.

3 Solution Space for Firmware-Level Fault Injection in X86

The standard firmware in X86/X64 systems is still the *Basic Input Output System (BIOS)*, which was invented over 20 years ago for the first IBM personal computer. The BIOS is responsible for initializing and abstracting the computer hardware, in order to allow an operating system boot loader to work on different hardware platforms. With modern operating systems, the BIOS interfaces are no longer used after startup, since native software drivers take over control. Extending BIOS firmware with a fault injection functionality would demand some source-code modification – but any standard BIOS software is subject to strict licensing. One possible alternative is open-source firmware, with *Coreboot* as most prominent example.

A completely different option recently became available with the introduction of Itanium processors. Intel and other companies started to develop an alternative firmware concept in order to circumvent classical BIOS limitations such as the 16 bit code base and the proprietary implementation strategy. The result of this effort is the *Extensible Firmware Interface (EFI)* standard. The EFI specifications define programming interfaces that allow extension and configuration of a computer's firmware in a more flexible way than it was the case with traditional BIOS. Beside the exclusive usage of EFI in Itanium systems, it is also the default firmware for all recent Apple computers and many other recent X86/X64 systems.

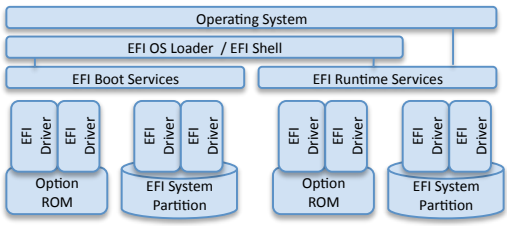


Figure 1: Extensible Firmware Interface Stack

Figure 1 shows the relation between EFI and other system components. All services are implemented by EFI drivers, which are either stored in the motherboard option ROM or can be loaded from a dedicated hard disk partition. EFI drivers provide the firmware's hardware abstraction, e.g., for console interaction, PCI bus access, or memory access of the boot loader. EFI-based applications, especially the operating system boot loader, use the driver-implemented functionality.

EFI *boot service drivers* are only available until the initiation of an operating system startup has been signaled. EFI *runtime service drivers* remain in memory to provide firmware functionality to the operating system.

A second dimension of the solution space, beside the firmware extension strategy, is the execution mode of fault injection code.

A fault injector implementation always runs as 'out-of-order' code during normal processor operation, for example as interrupt-triggered driver, trap handler, or debugger code. Since our goal is to inject faults without any software modification on operating system level or above, we propose to run the fault injector in the *System Management Mode (SMM)*. Besides the well-known real mode and protected mode, SMM is the most privileged execution mode of X86- compatible processors. It is originally intended for special BIOS software that has to be regularly executed, such as power and fan management functions or the handling of hardware error events (e.g. memory parity faults).

The SMM processor mode is triggered by a special interrupt, the System Management Interrupt (SMI), or by sending a special APIC message to the processor. The SMI is a non-maskable interrupt that takes precedence over all other interrupts. With the switch to SMM, the CPU freezes all activities in the current mode of operation. The interrupt handler routine has now full access to computer resources. All I/O and system machine instructions are allowed, which is a difference even to the Ring 0 protected mode code. Memory is accessible in 32bit real mode addressing. When the SMI handler has completed its operation, it executes a special instruction that causes the processor to resume the operation mode that was active before.

Modern processors with hardware virtualization support can also be influenced by SMM in the sense that the currently active logical processor is interrupted and the state is saved. All software layers above firmware level are now transparently suspended, starting from the host operating system itself.

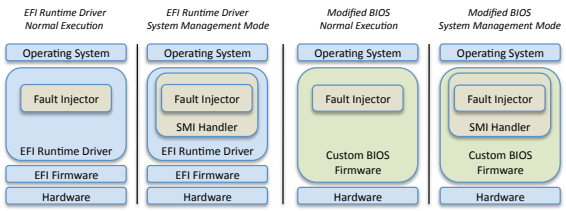


Figure 2: Solution Space

By combining firmware extension using EFI or custom BIOS with fault injection triggering using SMM or not, we end up with four proposals for firmware-based fault injection, as shown in Figure 2.

4 Initial Analysis

A. Supported Fault Locations

The fault location denotes the specific hardware part that can be set to an erroneous state under the given fault model. An analysis of related work showed that for most approaches the focus is on the main processor. Some of the SWIFI solutions also support memory cell or I/O driver fault injection.

In our firmware-based approach, both non-SMM variants support all explicit modifications (such as register changes) that are possible for kernel-mode software running in protected mode. This is also the case for driver-based related SWIFI solutions.

With the usage of SMM, the set of possibilities for processor fault injection is substantially extended. Several X86/X64 processor registers are saved before the SMI handler starts executing. Parts of the saved state map are changeable in an SMI handler implementation, which allows the manipulation of other general purpose registers, status registers, and even the instruction pointer on bit-level. Other processor registers are not automatically saved and restored, but can still be changed. This includes FPU registers, cache configuration registers, control registers and the trap con- troller state. A similar classification exists for registers in the Itanium processor.

The extended set of fault locations with the SMM approach enables a set of new opportunities for hardware fault experiments. Manipulations in the saved processor state information might even lead to machine halting on SMM exit, since the processor hardware itself tries to detect invalid conditions. This might or might not act as another possible fault type, for example to emulate a processor hardware crash fault. The injection from a SMM handler could theoretically also support processor caches, if they remain untouched when SMM is enabled by the processor. According to the architectural documentation of SMM, this is a model-specific property of the processor revision.

When memory cells are targeted as fault location, SMM approaches are limited to 32bit real-mode address- ing, whereas non-SMM approaches can access the entire protected-mode address space. SMM handlers also have write access to the memory of a potentially running virtualization hypervisor. This is possible since logical processors are also suspended on SMM entry. SMM-based fault injection therefore offers the possibility for hypervisor fault injection, a topic that is –to the best of our knowledge– not covered by dependability research at the moment.

When I/O devices are targeted as fault location, both the non-SMM and the SMM firmware approaches support the direct access by port-mapped and memory-mapped I/O. It should be pointed out that this works without any involvement of the operating system itself.

From the viewpoint of supported fault locations, the SMM-EFI approach needs to be favored. It makes the fault injector available even before the operating system loader starts to operate. SMM supports a broader range of manipulations, since neither the restrictions of the protected mode nor operating system security mechanisms are active during SMM interrupt handling.

B. Portability

From the viewpoint of portability, the usage of BIOS alternatives demands support for the very specific combination of chip set, memory controller, and other parts of the motherboard. EFI, in contrast, is a standardized interface that tries to solve the portability problem for firmware source code. With the envisioned spreading of EFI as default firmware in X86/X64 systems, more and more hardware platforms will support the EFI-based realization of a fault injector. We therefore favor these solutions for the sake of portability. The level of support for SMM handler code in EFI runtime drivers is still under investigation.

C. Fault Trigger

The fault trigger is an explicit condition that, once met, leads to the injection of a hardware fault. There is a distinction between the mechanism that checks the condition, and mechanism that subsequently activates the fault injection code.

Classical examples for SWIFI activation mechanisms are timeouts (with unpredictable fault effects, therefore suit- able for transient faults and intermittent hardware faults), exceptions resp. traps, and inserted code. Firmware-based fault injection theoretically adds the possibility for hardware interrupts as fault trigger condition, but since EFI does not support the hooking on hardware interrupts, the custom BIOS replacement provides the better solution in this case.

Exceptions, traps, and inserted code can only be implemented by extending or modifying the tested system. This is contradictory to our goal of non-intrusive failure injection.

In the case of using an SMM handler for fault injection, an SMI is the only available starting point for the injection. SMIs can be triggered from software directly or indirectly using I/O controller chips. Such controllers support a large set of SMI-triggering events including the power button, real-time clock timers, serial / USB port activities, or NMIs.

Using the I/O controller to trigger an SMI enables the re-use of well-known condition checks from other SWIFI approaches, but would require code in the operating system. Therefore a trade-off exists between minimal intrusiveness (where only hardware timers are available as trigger) and maximum flexibility in fault injection triggering (at the cost of portability).

We conclude that firmware-based fault injection would not provide any advantage if a rich set of fault triggers is the primary concern. This property is independent from the chosen implementation strategy. With respect to all investigated properties, we decided for the EFI runtime driver approach.

5 Intel Machine Check Architecture

Beside our firmware-based approach for fault injection, the investigation of the FutureSOC lab systems has lead to a new possibility for sub-OS-level fault injection. The *Intel Machine Check Architecture (MCA)* feature of modern Nehalem EX processors, ported from the Itanium world to all latest models in the Xeon series, offers the possibility to simulate hardware error events for higher layers in the software stack. The deeper experimentation with the HP DL980 has showed that the implemented, very latest, *ACPI Platform Error Interface (APEI) specification* also supports high-level monitoring and fault injection for hardware components as wrapper around the processor MCA capabilities.

A relevant aspect in this scenario is the default consideration of MCA information in the operating system. Due to its multi-platform support, Linux has direct support for the *Machine Check Architecture* of x86 processors. Since MCA reports correctable, uncorrectable and software-recoverable errors related to hardware units of the processor, the information in MCA status registers can be used to analyze whether the process context which was running before the error can be resumed or not.

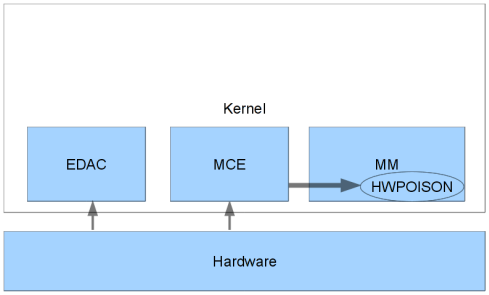


Figure 3: Fault Injection and Management Capabilities in Linux

Figure 3 shows the current handling of hardware faults in the Linux operating system kernel. There are two sets of kernel modules which are handling hardware-related errors. The first set is called *EDAC - Error Detection and Correction*. These modules are used for fetching corrected and uncorrected error notifications coming from the hardware. Information about these errors are then exported into the virtual *sysfs* file system. Initially, EDAC supported memory chipsets only to gather ECC data. The FutureSOC lab analysis showed that there is also a module for PCI Bus errors and for Core i7 MCA registers, which report issues from the integrated memory controller.

At the time of this writing, EDAC modules only had very basic support for fault recovery. One can only choose between a kernel-panic and do-nothing on uncorrectable errors.

Three years after the first inclusion of EDAC into the main kernel, Nehalem was released as first Intel x86_64 processor supporting software recovery of hardware faults. The Linux community therefore developed *hwpoison*, an interface of the memory management subsystem for triggering memory faults, in order to utilize the new hardware features for testing purposes. The tool maps the faulty reported address to a page and marks this page as poisoned through the help of the memory controller hardware. Depending on the type of page (user vs. kernel, clean vs. dirty) it will then either isolate the page or panic if it can't prevent further propagation of the fault.

Since most of the memory chipsets today have an interface for injecting some memory faults at arbitrary addresses, Linux supports this kind of fault injection too. The operating system also comes with according software tools, such as the module *mce-inject* that can be used to inject arbitrary MCE errors into the kernel. *hwpoison-inject* can be used to test the fault handling of the memory management.

6 Conclusion

The FutureSOC project on software-implemented fault injection works on novel approaches to fault injection that operate on or below the operating system level. Our initial analysis phase showed that SMM-based fault injection in the firmware can support a set of unique fault locations in the processor that are not accessible to other fault injection technologies, with the instruction pointer register as most prominent example. This enables a set of new opportunities for hardware fault experiments in X86 and Itanium operating systems. SMM handlers also have write access to the memory of a potentially running virtualization hypervisor. This is possible due to the fact that logical processors are also suspended in their execution on SMM entry. SMM-based fault injection therefore offers the possibility for hypervisor fault injection, a topic that is - to the best of our knowledge - not covered by dependability research at the moment.

Beside the clarified (and now actively developed) EFI-based fault injection approach, MCA turned out to be a promising alternative for firmware-level fault injection. We are currently investigating both the the APEI and the MCA capabilities of the different latest Nehalem EX processors available in the FutureSOC lab machines.

Project Progress Report for the Future SOC Lab Project B-HiP Business For High-Performance Computing

Andreas Emrich
Institute for Information Systems (IWi) at the German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, Campus D3.2
66123 Saarbrücken
Germany
andreas.emrich@dfki.de

Frieder Ganz
Institute for Information Systems (IWi) at the German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, Campus D3.2
66123 Saarbrücken
Germany
frieder.ganz@dfki.de

Dirk Werth
Institute for Information Systems (IWi) at the German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, Campus D3.2
66123 Saarbrücken
Germany
dirk.werth@dfki.de

Abstract

The project B-HiP investigates the applicability of high-performance computing infrastructure to support the execution of business transactions. In two major research lines organizational business processes and individual work processes are investigated and mechanisms for real-time support for such scenarios has been developed. The work in the reporting period mainly focused on creating the foundational model layer and to implement prototypes demonstrating the performance of such analysis mechanisms. Within the Future SOC Lab infrastructure, performance tests have been executed and proved a considerable performance increase. In future developments also complete business scenarios should be evaluated, in order to answer the question, whether a HPC infrastructure together with the B-HiP functionalities can provide relevant real-time support for business processes.

1 Project Idea

The project B-Hip is founded by the ministry for economy and science of the federal state Saarland. The project has started in January 2010 and will end in December 2011. It addresses new challenges in the area of enterprise applications.

B-Hip will explore potentials for the application of HPC technologies in BPM scenarios. By this means, real-time business should be enabled, i.e. analytic tasks that were traditionally performed in OLAP scenarios can be performed, when transactions occur and are thereby an integral part of transaction management.

The overall vision of B-Hip is to merge the traditional scenarios of OLTP and OLAP, in order to have information at that point of time available, when it is needed: In the actual transaction, in the actual business process. This would lead to a paradigm shift in enterprises: Enterprise do not react on events some timer later, they directly take action when the event occurs. Moreover, by the use of simulations even a proactive behaviour could be achieved: Possible consequences of a transaction can be forecasted before actually performing this transaction.

For business processes, this provides new opportunities to control business processes: Depending on the actual transactions, simulations can be run in real-time, in order to identify improvement potentials for the underlying business process. E.g. complex material requirements can be determined in real-time and respective orders can be carried out right away. By having this “information at your fingertips”, new opportunities arise to change the business process itself.

For work processes, this provides new opportunities to analyze the current work context under consideration of the focused workers skills, abilities and personal preferences. Complex analytics can be run in real-time, in order to provide the most appropriate information, that helps to perform the task in question. By this means, information are available for the respective area of work, people with similar expertise or who are experts in that area are available to contact, process execution alternatives are recommended to the user, etc.

In case studies, B-Hip will investigate the impact and potential of HPC infrastructure on business and work processes. A demonstrator will show how real-time business software can create proactive assistance in business and work processes.

In order to leverage insights gained from business transactions, actions and changes should be assessed according to their impact on business processes and related data. Not only business processes themselves, but also associated resources, organizational units, data, etc., can be affected by such changes. According to that, any change on workflow artifacts such as activities, but also changes on organizational units (e.g. staff decisions) or resources (e.g. machine upgrades) should be traced. This enables real-time analysis of the impact of such changes. In order to achieve that, a context model for BPM is needed, that is capable of describing these relationships.

2 Used Lab Resources

The context model contains all business related artifact and their relations to each other. This enables the analysis of an incoming change and moreover allows forecasting of the impact on related processes and resources. Therefore a huge amount of data has to be processed as every involved artifact of the business has to be mapped in the model.

In a first scenario ontologies are exploited to model the relations and the artifacts. By using semantic technologies one can use existing technologies to find relations between artifacts by the use of logical reasoning through the model.

Through the high level of detail in the model a huge amount of data is generated. Talking in enterprise size models with a size of several gigabytes have to be processed and analyzed by software. Usually ontologies are bulk loaded from disk as they are needed by the analyzing software. The more efficient way to handle the data would be to store the entire ontology in the main memory.

In-Memory Databases provide the ability to store the data in the main memory of a server. The advantages are higher performance in read and write operations. The disadvantages are the high costs of memory and the fact that data that has not persisted to disk is destroyed after the loss of power.

With the resources provided by the lab we want to evaluate the performance of creating and querying semantic data in disk/memory databases. The goal is to get results that fast that decisions could be made in real-time.

In this scenario we used existing business ontologies to evaluate the time to create the semantic repository on disk/memory. Through the fact that there are no huge business centric ontology sets, we connected business ontologies to other matured ontologies to get realistic evaluation data. The evaluation tests were performed on the Fujitsu RX600S5 – 1 Server. Thus the size of the dataset would not exceed the memory limitations of this server, a shared access was reasonable.

3 Findings

This section describes the project progress achieved in the reporting period. As the project B-HiP started in January 2010, the focus in the reporting period was on conceptual work. Nevertheless, certain aspects have been prototypically implemented and some of them have been evaluated regarding the performance, taking advantage of the Future SOC Lab infrastructure.

3.1 Conceptual Work

In order to provide appropriate support for the scenarios as mentioned in section 1, several improvements have to be achieved in structured as well in semi-structured or unstructured scenarios. For that purpose, a semantic context model has been developed, that allows for a formal, multi-view-enabled view on various aspects, whereas search and recommendations, as well as semantic relationship discovery enable support in less structured scenarios. The depicted approach for cross-application traceability shows, how these different approaches can be unified in a business use case.

3.1.1 Semantic Context Model for BPM

The basic knowledge infrastructure for the scenarios mentioned in section 1 requires a semantic context model, which is capable of associating process artifacts with any other related artifacts. Therefore an open, standard-based approach for specifying process artifacts has been developed based on the results of sEPC and the SUPER project. The major difference with our approach is, that we chose an upper-level ontology approach, in order to be independent from a specific process definition format and to support interoperability with other process-related ontologies such as sEPC, SUPER-EPC, WSMO, etc. and other ontologies.

The following figure demonstrates how, sEPC process artifacts are mapped to concepts from our upper-level ontology:

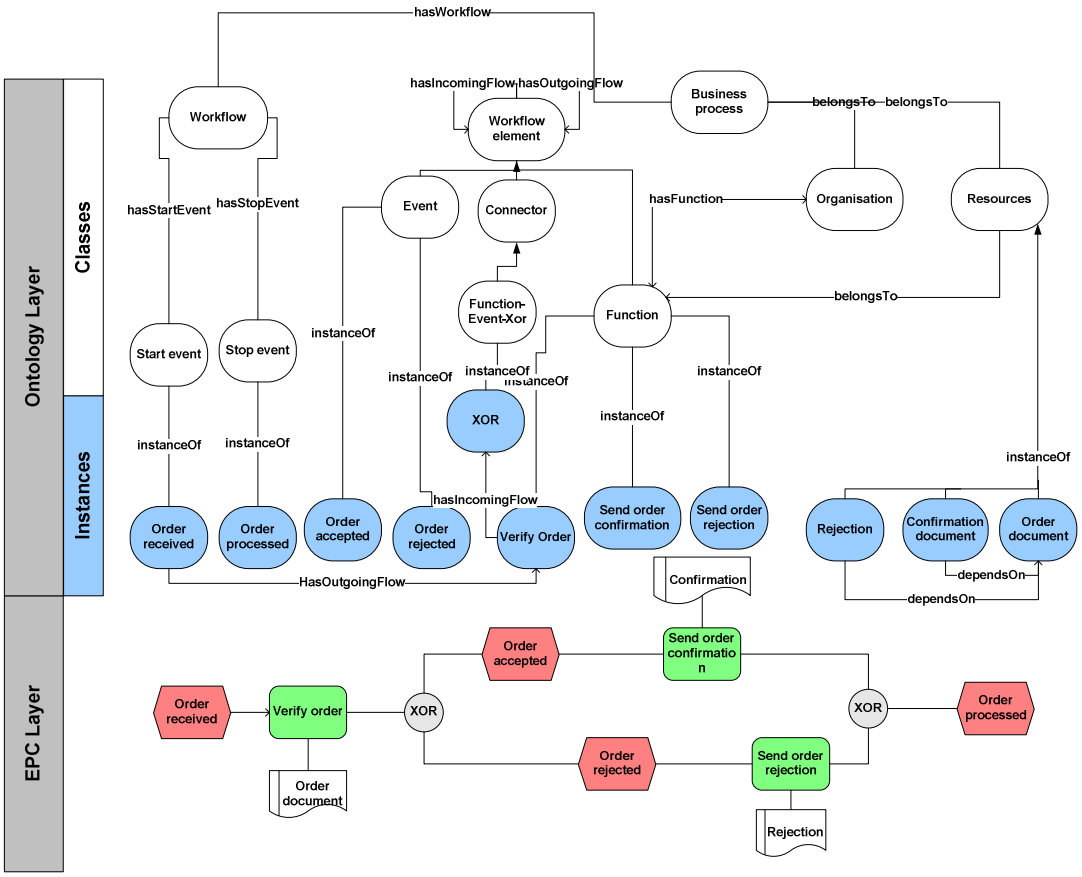


Figure 1: Semantic Context Model sEPC

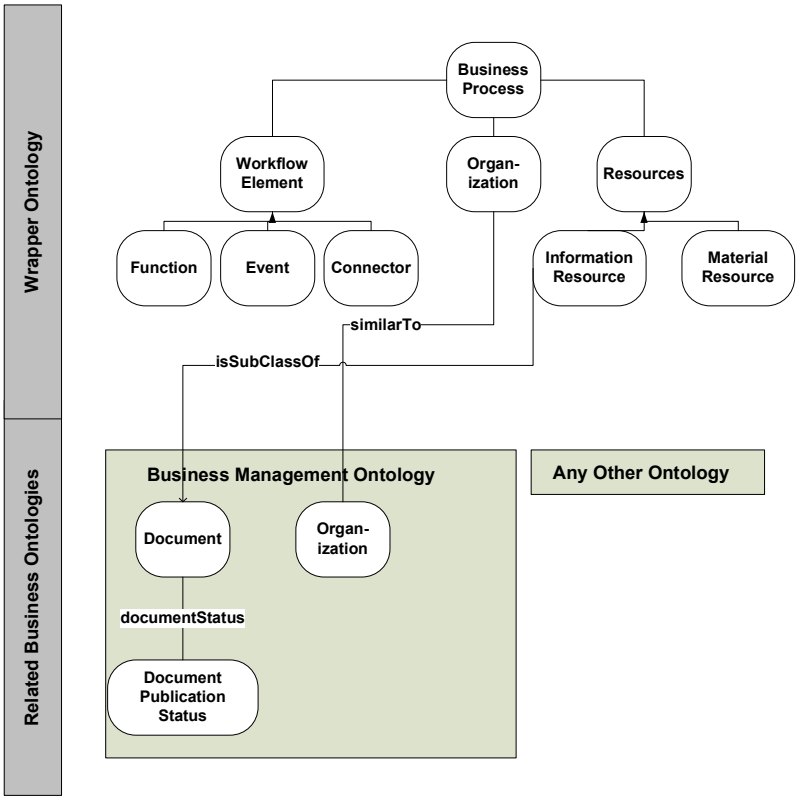


Figure 2: Ontology Mapping with B-HiP SCM

As figure 1 illustrates, the concepts in an EPC do not only relate to pure process artifacts, but could certain people in an EPC process could also be included in ontologies describing people such as FOAF, as well as documents could also specified in other business ontologies, such as the BMO (business management ontology).

Figure 2 shows, how the upper-level concepts could be linked with the BMO ontology.

3.1.2 Semantic, Context-Aware Search and Recommendations

As business context can rapidly change and a process execution does not solely depend on the model relationships that are defined for it (even when organizational units, documents, etc. are covered), but also changing environmental events, market conditions, etc. Therefore, the traceability of models helps to identify related objects very fastly, but does not cover all possible influences. For that reason, semantic and context-aware search and recommendation mechanisms, which have been developed in the European FP7 project m:Ciudad were transferred to the B-HiP scope.

3.1.3 Semantic Relationship Discovery

The semantic context model as proposed in section 3.1.1 also implies a severe problem: According to that, any information in the enterprise needs to be modeled and all relationships across various modeling languages or ontologies need to be specified explicitly. On the one hand, intelligent mechanisms are needed to extract the information intelligently from enterprise data repositories. This could be achieved by standard approaches of database mining. On the other hand, even in contemporary enterprise data definitions important links between artifacts are missing. One reason for that is an old problem of data modeling: As only knowledge can be modeled, which is explicit, implicit knowledge cannot be covered by traditional modeling approaches.

Therefore an approach has been developed, which monitors each transaction occurring. A transaction is considered in this context as a process step, which requires a certain input, creates a certain output, is executed by certain organizational units or systems, etc. Every aspect which characterizes the transaction, should consequently be mapped to respective ontology concepts from the semantic context model (s. section 3.1.1. Using machine learning techniques over these transcaction logs allows to discover model relationships.

3.1.4 Cross-Application Traceability

One of the major problems of enterprise IT environments is the heterogeneity of applications, implying media breaks in the related business processes. E.g. if a certain process should be handled within a document management system, it ceases to be traceable, once someone in the process forwards the information via email.

To tackle this problem, B-HiP has developed an architecture approach, which takes advantage of the semantic context model and the associated traceability features. Figure 3 depicts the approach.

Plugins for each application in the enterprise IT infrastructure allow to exchange interaction logs as described in the previous section. By that means, the necessary ingredients are provided to analyze model-based traceability and probabilistic semantic search as described in section 3.2. Thus, all impacts of business events can be determined in real-time and applied in the respective business processes.

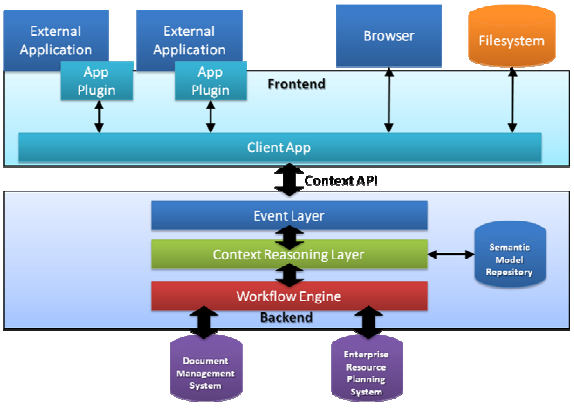


Figure 3: Cross-Application Traceability

3.2 Protypical Evaluation

The prototype loads the semantic data into a repository either in memory or into a disk database. Incoming changes and actions for example the use of a resource during a process or the execution of activities are analyzed. Related artifacts which are affected by the incoming event are highlighted and could be used to provide further information about the process trail. Despite the gain in the information level, forecasts could be simulated. What impact has a change in a resource on other processes and resources? It could be said which processes are affected by a loss of a certain resource for example in the case of illness of a warehouseman.

The software locates the resource in the repository and tries to find connected artifacts through logical reasoning. In a further step events could be executed automatically, such as the notification of a substitute worker or the corresponding manager.

In scenarios where millions of transactions arise during the day, the system has to react in (almost) real-time. Sesame [1] was used to create the repository. It

is possible to use file based storage layer as well as in-memory databases and traditional relational database management systems.

3.3 Performance Evaluation

In this phase we used three different ontology datasets of different size. In a first run the set is loaded into the repository which resides either in an In-Memory Database or on the hard drive. After the creation the repository gets queried with several requests. The aim is to find an artifact in the ontology and getting all of its related neighbors. In our business scenario this would be the analogy of finding related resources and processes of an observed object like an invoice or a material resource.

The first Ontology is the OWL Version [3] of the MIT Process Handbook [2] with a size of 16Mbytes and ~8000 artifacts like resources and processes which are interlinked with each other. The second one is the openCyc [4] Ontology with a size of ~160Mbyte. The Cyc Ontology contains general knowledge and is therefore not limited to business related topics. The third used Ontology is the YAGO [4] Ontology with a size of 20Gbytes and nearly 2 million entities.

An overall goal of this performance evaluation was to find the right technology and storage method of the ontology to get query results which are (almost) realtime. Therefore large datasets must be queried in a decent time.

In a first step we wanted to benchmark the difference between disk and In-Memory persistence.

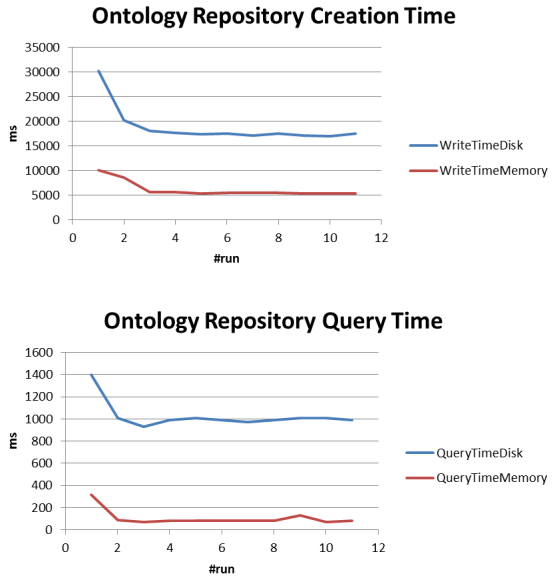


Figure 4: Performance Data Workstation

With the use of an In-Memory Database we had a gain in speed of the factor 3 in write access and factor 9 in terms of read access. For this test we used the smallest dataset. We did no further evaluation as this result fixed our opinion about using In-Memory technology instead of traditional disk based databases.

In the following step we used the big datasets with the In-Memory Database to get statistics about the read/write time. First we ran the benchmark on an ordinary workstation with a dual core processor and 4 Gbyte of main memory. Then we used the same data and setup on the HPC environment. To build the repository in the main memory the heapsize was increased to 3Gbytes on the workstation and 32Gbytes on the HPC environment.

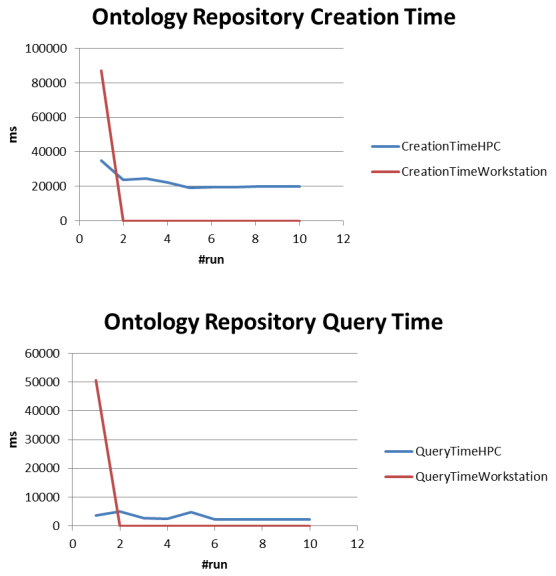


Figure 5: Performance Data HPC Cluster

We performed 10 runs on both systems but the workstation was only able to create a repository one time. After the first run the java virtual machine crashed during limited capacity. To query the whole dataset it needed in average 80 seconds which is far away from realtime. New technologies and/or clustering has to be considered in the next steps.

4 Next Steps

In the next project phase of B-HiP the depicted conceptual threads will be transferred in executable scenarios and will use the Future SOC lab infrastructure for performance evaluations and with real operational data provided throughout the SOC lab.

4.1 Mined Interaction and Web Data

The usage mining of interactions as well as the mining of web resources should help to provide probabilistic means to determine content which is not explicitly covered by the enterprise model. The mining of this data is not considered within the Future SOC lab

tests, as the only time-consuming activity is the crawling of web resources which is mainly limited by bandwidth constraints.

In the context of Future SOC Lab it will be evaluated, how these mined data can be continuously analyzed in order to discover structures and acquire knowledge from these data. The timeliness of such analysis is crucial to provide accurate support in unstructured scenarios.

4.2 Complex, Multi-View Model Reasoning

The behavior of processes is not only determined through process artifacts alone, but also documents, resources, employees have a crucial impact on the performance of a process. In consequence, mere process mining approaches cannot gather the whole complexity of a process, thus its behavior cannot be predicted in the best possible way.

In terms of an evaluation, all the semantic reasoning which is needed for such complex relationships of processes and artifacts, should terminate in less than one second. Moreover, tests with user groups should show, that the proactive system which is provided, is actually useful in structured, semi-structured and unstructured scenarios.

4.3 Real-time Semantic Search (Indexes)

Modern search engines deliver a considerably fast response time for search queries, but nevertheless, they seldomly leverage semantics. Only sporadic query processing is applied, which e.g. consider sword stemming, the location of the user, etc. Nevertheless, no operable high-performance search engine exists, which can meet the needs of huge enterprise applications.

The developed solutions and algorithms for semantic search and indexing of information should be applied to business data. Therefore, simulations for interaction data and search interactions will be performed within the Future SOC Lab. The time for a search query thus should be reduced from currently about 80ms to less than 10 ms.

References

[1] Broekstra, J., Kampman, A., van Harmelen, F. (2002): Sesame: An Architecture for Storing and Querying RDF and RDF Schema. In: Proceedings of the First International Semantic Web Conference (ISWC 2002). (LNCS), Springer (2002)

[2] Malone T, Crowston K, Herman G (2003) Organizing Business Knowledge - The MIT Process Handbook. The MIT Press

[3] OWL MIT Process Handbook (2010), http://www.ifi.uzh.ch/ddis/ph-owl.html (01.10.10)

[4] Lenat, D. B. (1995) “Cyc: A Large-Scale Investment in Knowledge Infrastructure”, CACM 38(11), 1995, pp. 33-38.

[5] Suchanek, F.M.; Kasneci, G.; Weikum, G.:. (2007) Yago: A Core of Semantic Knowledge. In 16th international World Wide Web conference (WWW 2007), ACM Press, New York, NY, USA

Business Process Model Intelligence

Mathias Weske, Sergey Smirnov, Matthias Weidlich, Artem Polyvyanyy
Hasso Plattner Institute
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam, Germany
{Mathias.Weske, Sergey.Smirnov, Matthias.Weidlich, Artem.Polyvyanyy}@hpi.uni-potsdam.de

Abstract

Management of large organizations is often driven by business processes. As a consequence, explicit representations of business processes are created in process modeling initiatives. The inherent complexity of large organizations results in repositories that contain up to thousands of process models. Apparently, such a model collection represents an enormous amount of information about the organization. This information might be leveraged to support harmonization of processes run by different organizational units. Given this state of the art, we would like to explore how large process model collections can be analyzed in an intelligent fashion. The focus of this project is on methods for discovering information hidden in large model collections and enabling business process improvement. Four research questions are investigated: model consistency analysis, activity patterns mining, process model refactoring, and process model execution. The project aims to deliver solutions for these four challenging questions, refining the existing algorithms and utilizing a high-performance infrastructure delivered by Future SOC Lab for experimentation. While there are process model collections available, the project would benefit from a collection of real-world process models and a complimentary set of event logs of complex enterprise systems provided by Future SOC Lab.

1 Project Idea

These days, management of large organizations as well as design and analysis of their supporting information systems is often driven by business processes. As a consequence, explicit representations of business processes are created in process modeling initiatives. The inherent complexity of large organizations along with the wide variety of drivers for business process modeling initiatives, reaching from business evolution and process optimization over compliance checking and process certification to process enactment, results in repositories of process models that contain up to thousands of process models. Apparently, such a model collection represents an enormous amount of information about the organization. This information, in turn, might be leveraged to support harmonization of processes that are run by different units of an organization. To this end, inconsistencies might be detected or future modeling efforts can be guided by the knowledge about the existing models. However, there are two major obstacles. On the one hand, the heterogeneous representation of process models created by different organizational units and for different purposes precludes any direct comparison of similar process models. On the other hand, the sheer amount of information that is materialized in the process models of a repository imposes serious challenges. Analysis of process instances is another promising direction of business process improvement. Thereby, it is necessary to go beyond process models and study running processes. This implies the research on design and architecture of process execution platforms in a highly scalable environment, the cloud.

We propose concrete approaches of leveraging the information kept in a large process model repository, ensuring high model quality, and opening the potential for process improvement.

1.1 Consistency Analysis

In order to detect inconsistencies in the operations of different business units, their process models can be compared in order to identify operational commonalities and differences. Such a comparison, as a first step, requires the identification of activities in one business process model that correspond to activities in the other model. A major challenge for this matching is that business process models often do not use the same level of detail and the same words to describe activities, i.e., there is a heterogeneous representation. Hence, the comparison of related process models typically requires an extensive amount of

manual preprocessing. Recently, techniques based on structural analysis and natural language processing have been proposed in order to identify correspondences between activities or sets thereof [1,2]. Although the techniques and the algorithms are available, the problem state space is normally huge: the analysis stems to a combinatorial problem, where a research object is a process model of a large size.

1.2 Patterns Mining

While large repositories of process models describe expected knowledge about the organization, they also contain lots of hidden information. Hence, a natural desire is to externalize this knowledge and make it available to the users. The newly obtained information can be used to improve the quality of existing models and facilitate creation of high quality models in the future. One way to represent the knowledge spread over numerous models of a repository is patterns – groups of objects, which often appear together in business processes. One can suggest different types of objects to be investigated for patterns identification. While [6] investigates activity patterns in process models, [5,7] explores action patterns. In [5,7] two types of action patterns are distinguished: co-occurrence action patterns and behavioral action patterns. While co-occurrence action patterns capture the fact that a group of actions reappear together in several models, behavioral action patterns also describe the recurrently observed behavioral relations between them. However, a study of labeling patterns, i.e., recurrent application of labels, or event patterns may reveal new strata of information about an organization. While there are efficient algorithms for derivation of action patterns, the task still remains computationally intensive, see [3-6]. Hence, the search for recurrent structures in large process model repositories demands sufficient computing power.

1.3 Process Model Refactoring

Assuring model quality always becomes crucial, once engineering methods are deployed in an organization. Assuring the quality of business process models in large repositories is not an exception. The process model quality includes several aspects, from the correctness of behavioral properties to understandability of models by humans. The latter aspect can be further refined to a bunch of questions. Among them is the influence of a model structure on the human comprehension. It has been argued that block-structured process models are easily interpreted by humans, while models with an arbitrary structure are confusing. There exist approaches addressing this problem by stemming an arbitrary process model to a block structured one [8]. However, this task belongs to the NP class and assumes availability of large computation powers. A further investigation of the problem requires thorough experiments with the existing methods and the analysis of their outcomes.

1.4 Process Execution Engine

Going beyond process modeling and aiming at process instances reveals new horizons for business process analysis. Hence, deployment, and monitoring of a process execution platform in a highly scalable environment, the cloud, is in the focus of this project. Traditional process execution engines are typically built as siloed applications with a transactional database at its base, running in a rather static infrastructure. Such architectures do not scale well in flexible environments. Cloud providers, such as Google, Microsoft, and Amazon, offer virtual server platforms that scale extremely flexible on demand. However, traditional process engines need to be adapted to benefit from this flexibility. Thereafter, a demand for highly scalable and elastic software architectures for process engines running in the cloud emerges.

2 Used Future SOC Resources

To achieve the goals outlined in this project, we expect technical support from Future SOC lab. In addition, access to real-world data that could be analyzed would strengthen project results even more. It would fit nicely into current developments in business process management, where real world data and application scenarios play an increasingly important role.

2.1 Infrastructure Support

As the BPT group already has in possession the application prototypes to be executed on the infrastructure proposed by the Future SOC Lab, we expect that we can receive support services from the Lab. Such services should include deployment and execution of the existing code on the high-performance servers. To this day we have been supported by the Future SOC Lab with respect to the required hardware and software infrastructure.

2.2 Process Model Data

The key aspect of the research project is an availability of a suitable research object, i.e., one or more process model collections and associated execution logs. Logs record activities that occur during process execution. We would welcome process logs or, if logs are not available, an infrastructure in which we could generate process logs. Logs need to be correlated with process models so that execution data can be associated and analyzed with respect to process model data. Notice that it is of primary importance to study real world process models originating from the industry, rather than artificially designed modeling

artifacts. This enables the research project to focus on the challenges relevant in practice and increase the research impact of the initiative. To this day we have got an opportunity to get acquainted with the data on business processes realized in SAP Business ByDesign Software Solution.

2.3 Process Data Contact Person

To maximize the utility of provided process model data we request to establish a reliable connection with a contact person who represents the company that delivers the process data. The Future SOC Lab introduced us to Mr. Thorsten Kugelberg, a contact person who will guide us through the SAP Business ByDesign system.

3 Findings

At the current stage the project is in the exploratory phase. As mentioned in Section 2, the declared project goals assume availability of the following resources:

1. a hardware and software infrastructure,
2. a process model collection.

With respect to the first resource, we have got acquainted with the infrastructure provided by the Future SOC Lab.

With respect to the second resource, process model collection, a number of alternatives has been investigated. As a backup solution we can use the collection of 604 process models dating back to year 2000. While this collection has been thoroughly studied in the literature on business process technologies, its process models are rather simplistic and do not exhibit all properties that are in focus of this research project. This motivates the demand for another collection of real world business process models.

Recently, Future SOC Lab has provided us the contact person, Mr. Thorsten Kugelberg, who can introduce us to the data about processes realized in the SAP Business ByDesign software solution. As SAP Business ByDesign delivers preconfigured process best practices in financing, customer relationship management, human resources management, to name a few, we perceive this data as a preferable research object. However, the process data in SAP Business ByDesign still requires a thorough evaluation with respect to the project usage.

4 Next Steps

The direct next step is the study of information about processes in the SAP Business ByDesign software solution. Two types of information are of interest: 1) the business process models explicitly captured by the system in the form of best practices and 2) the logs generated through business process execution. If the SAP Business ByDesign software solution provides sufficient data, the respective processes can be selected as the research object. In the opposite case the backup process model collection can be used.

Once the system is in place, we start the experiment. This means that the selected process model collection is analyzed by means of the software developed by the BPT group members. If the process model collection of SAP Business ByDesign solution is selected, a preparatory step might be needed: development of software components for accessing process data in Business ByDesign system.

References

[1] M. Weidlich, R. Dijkman, and J. Mendling. *The ICoP Framework: Identification of Correspondences between Process Models.* In Proceedings of the 22nd International Conference on Advanced Information Systems Engineering, Hammamet, Tunesia, 2010, Springer.

[2] M. Weidlich, J. Mendling, and M. Weske. *Computation of Behavioural Profiles of Process Models.* Technical report, Hasso-Plattner-Institute, June 2009. http://bpt.hpi.unipotsdam.de/pub/Public/MatthiasWeidlich/behavioural_profiles_report.pdf

[3] R. Agrawal, T. Imielinski, and A. N. Swami. *Mining Association Rules between Sets of Items in Large Databases.* In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pages 207-216, Washington, D.C., 1993, ACM

[4] R. Agrawal and R. Srikant. *Fast Algorithms for Mining Association Rules in Large Databases.* In VLDB, pages 487-499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[5] S. Smirnov, M. Weidlich; J. Mendling, and M. Weske. *Action Patterns in Business Process Models.* In Proceedings of the 7th International Joint Conference on Service Oriented Computing, pages 115-129, Stockholm, Sweden, 2009, Springer

[6] L. H. Thom, M. Reichert, C. M. Chiao, C. Iochpe, and G.N. Hess. *Inventing Less, Reusing More, and Adding Intelligence to Business Process Modeling.* In Proceedings of the 19th international conference on Database and Expert Systems Applications, pages 837-850, Turin, Italy, 2008, Springer

[7] S. Smirnov, M. Weidlich, J. Mendling, and M. Weske. *Object-Sensitive Action Patterns in Process Model Repositories*. Proceedings of the 1st International Workshop on Reuse in Business Process Management (rBPM), Hoboken, NJ, USA, 2010

[8] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. *Structuring Acyclic Process Models*. Proceedings of the 8th International Conference on Business Process Management (BPM). Hoboken, NJ, US, 2010

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
41	978-3-86956-108-0	**The effect of tangible media on individuals in business process modeling: A controlled experiment**	Alexander Lübbe
40	978-3-86956-106-6	**Selected Papers of the International Workshop on Smalltalk Technologies (IWST'10)**	Hrsg. von Michael Haupt, Robert Hirschfeld
39	978-3-86956-092-2	**Dritter Deutscher IPv6 Gipfel 2010**	Hrsg. von Christoph Meinel und Harald Sack
38	978-3-86956-081-6	**Extracting Structured Information from Wikipedia Articles to Populate Infoboxes**	Dustin Lange, Christoph Böhm, Felix Naumann
37	978-3-86956-078-6	**Toward Bridging the Gap Between Formal Semantics and Implementation of Triple Graph Grammars**	Holger Giese, Stephan Hildebrandt, Leen Lambers
36	978-3-86956-065-6	**Pattern Matching for an Object-oriented and Dynamically Typed Programming Language**	Felix Geller, Robert Hirschfeld, Gilad Bracha
35	978-3-86956-054-0	**Business Process Model Abstraction : Theory and Practice**	Sergey Smirnov, Hajo A. Reijers, Thijs Nugteren, Mathias Weske
34	978-3-86956-048-9	**Efficient and exact computation of inclusion dependencies for data integration**	Jana Bauckmann, Ulf Leser, Felix Naumann
33	978-3-86956-043-4	**Proceedings of the 9th Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS '10)**	Hrsg. von Bram Adams, Michael Haupt, Daniel Lohmann
32	978-3-86956-037-3	**STG Decomposition: Internal Communication for SI Implementability**	Dominic Wist, Mark Schaefer, Walter Vogler, Ralf Wollowski
31	978-3-86956-036-6	**Proceedings of the 4th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI
30	978-3-86956-009-0	**Action Patterns in Business Process Models**	Sergey Smirnov, Matthias Weidlich, Jan Mendling, Mathias Weske
29	978-3-940793-91-1	**Correct Dynamic Service-Oriented Architectures: Modeling and Compositional Verification with Dynamic Collaborations**	Basil Becker, Holger Giese, Stefan Neumann
28	978-3-940793-84-3	**Efficient Model Synchronization of Large-Scale Models**	Holger Giese, Stephan Hildebrandt
27	978-3-940793-81-2	**Proceedings of the 3rd Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI
26	978-3-940793-65-2	**The Triconnected Abstraction of Process Models**	Artem Polyvyanyy, Sergey Smirnov, Mathias Weske
25	978-3-940793-46-1	**Space and Time Scalability of Duplicate Detection in Graph Data**	Melanie Herschel, Felix Naumann

ISBN 978-3-86956-114-1
ISSN 1613-5652