HPI Future SOC Lab - Proceedings 2011

Christoph Meinel, Andreas Polze, Gerhard Oswald,
Rolf Strotmann, Ulrich Seibold, Doc D'Errico (Hrsg.)

Technische Berichte Nr. 70

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 | 70

Christoph Meinel | Andreas Polze | Gerhard Oswald | Rolf Strotmann |
Ulrich Seibold | Doc D'Errico (Hrsg.)

HPI Future SOC Lab – Proceedings 2011

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.de/ abrufbar.

Universitätsverlag Potsdam 2013
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
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/2013/6400/
URN urn:nbn:de:kobv:517-opus-64004
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-64004

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-230-8

Contents

A Benchmark Suite for Evaluating Parallel Programming Models

Michael Andersch, Ben Juurlink, and Chi Ching Chi
Technische Universität Berlin
Einsteinufer 17
10587 Berlin
{andersch,juurlink,cchi}@cs.tu-berlin.de

Abstract

The transition to multi-core processors enforces software developers to explicitly exploit thread-level parallelism to increase performance. The associated programmability problem has led to the introduction of a plethora of parallel programming models that aim at simplifying software development by raising the abstraction level. Since industry has not settled for a single model, however, multiple significantly different approaches exist. This work presents a benchmark suite which can be used to classify and compare such parallel programming models and, therefore, aids in selecting the appropriate programming model for a given task. After a detailed explanation of the suites design, preliminary results for two programming models, Pthreads and OmpSs/SMPSs, are presented and analyzed, leading to an outline of further extensions of the suite.

1. Introduction

The move towards multi-core architectures changes the programmers view of the architecture and introduces yet unresolved programmability issues. Increasing performance now requires the explicit exploitation of thread-level parallelism. The development of parallel programs is generally not a trivial task since an appropriate parallel decomposition of the algorithms needs to be found. Additionally, the programmer usually has to perform architecture-specific optimizations such as thread-to-core mapping and page placement, which could lead to different optimal parallelization strategies for different platforms. Furthermore, the verification and debug processes of such programs introduce additional difficulties caused by the complexity associated with sophisticated threads running in parallel.

All this has led to the introduction of several *programming models* in an attempt to relieve developers partly or completely from such parallel programming issues. These models, however, differ significantly in the provided underlying parallel principles, abstraction levels, semantics, and syntax.

This work aims at providing some means of comparison by introducing an *evaluation suite* consisting of several applications to examine and classify the features and qualities of shared memory parallel programming models. These applications will not only be used as benchmarks to measure performance levels of programs developed in a particular model, but also to evaluate the usability and features of that model. Additionally, a first version of this suite is presented along with preliminary results for two currently relevant models. The contributions of this work can be summarized as follows:

- We propose a benchmark suite specifically targeted at the evaluation of performance and usability of parallel programming models rather than parallel machines.
- We focus on modularity and portability for the benchmarks incorporated into the suite.
- We perform a case study, evaluating the novel OmpSs programming model [10], using POSIX threads as a reference for comparison.

This paper is structured as follows. Section 2 encompasses the top-level design decisions made in creating the evaluation suite. In Section 2.1, we define general requirements the suite must fulfill. In Section 2.2, these requirements are utilized to create a preliminary selection of benchmarks which are then presented in a more detailed fashion. A case study using the benchmark suite is presented and analyzed in Section 3. Section 4 discusses related work. Finally, in Section 5, conclusions are drawn and future perspectives are given.

2. Suite Design

On a high level, the benchmark suite must fulfill several critical requirements which can be derived from its objective as a suite to evaluate the programmability and performance of parallel programming models.

These criteria will then function as guidelines for the selection of benchmarks and benchmarking practice. In the following section, we identify six such criteria. They assure comparability, fairness and ease of use as well as enabling developers using the suite to gather hands-on experience with the programming model in use.Please read the following carefully.

2.1. General Requirements

1. A broad range of application domains must be covered.
2. Various parallel patterns and characteristics must be covered.
3. Various application sizes must be covered.
4. The suite must include input data sets of varying size.
5. Simplicity, modularity and portability must be ensured.
6. The parallelization approach must be fixed and well documented.

2.2. Benchmark Suite

The current benchmark suite is presented in Table 1. The **K**, **W**, and **A** identifiers in the second column are a realization of the third requirement, grouping benchmarks into one of three different categories, **K**ernels, **W**orkloads, and **A**pplications. A kernel consists of (a part of) the extracted core of a real-world application. Kernels are, therefore, comparably small ($<$ 1000 LOC) and exhibit only a single, isolated parallelization pattern.
Workloads are either derived by combining several kernels, thereby introducing additional data dependencies and covering more parallelization patterns, or it is a program considered too large to fit in the kernel class, but still only an extracted part of a real application.
Applications are full-grown software products which are widely used in industry or science and therefore feature the highest number of subsystems, dependencies and combinations of parallelization patterns.
Following is, in the order depicted in Table 1, a short description of each benchmark. It should be kept in mind this is a preliminary selection which shows the current and not the final state of the suite and is subject to change and extension.

1) c-ray: *c-ray* is a simple, brute-force ray tracer [17]. It is small (ca. 500 LOC for the serial version) and renders an image in PPM binary format from a simple scene description file. Despite its simplicity, c-ray is a very compute-intensive benchmark, featuring a high computation-to-communication ratio. The parallelization approach is depicted in Figure 1.

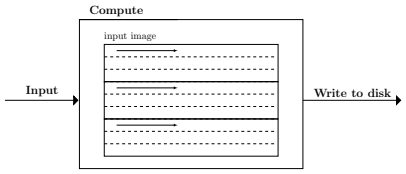


Figure 1. Parallel pattern for c-ray, rotate and rgbcmy kernels

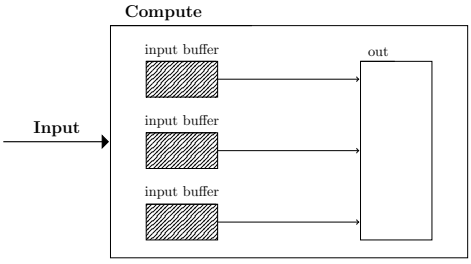


Figure 2. Parallel pattern for md5 kernel

2) md5: *md5* is a benchmark utilizing a standard implementation of the MD5 hash algorithm [15] to produce hash values. Since there is no exploitable thread-level parallelism in the block cipher construction used in MD5, the benchmark uses multiple input buffers consisting of predefined raw binary data which it processes in parallel. This structure is shown in Figure 2; hatched parts illustrate the exploitable parallelism.

3) rgbcmy: The *rgbcmy* kernel converts an input RGB PPM image to the CMYK color space used for image printing. Parallelism is found in the different pixels (which can be converted independently), visualized in Figure 1.

4) rotate: *rotate* is a benchmark which rotates an RGB image in binary representation by 0, 90, 180 or 270 degrees. The parallelization approach can also be visualised by Figure 1.

5) kmeans: The *kmeans* kernel executes the k-Means clustering algorithm [8] used in the domains artificial intelligence and data mining. It is derived from the correspondent benchmark in the NU-MineBench benchmark suite [9]. The algorithm is visualized in Figure 3.

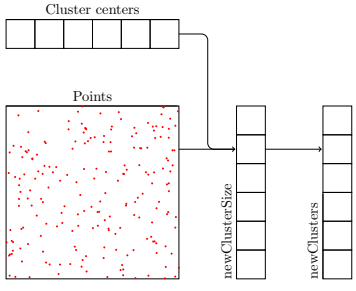


Figure 3. Algorithm for kmeans kernel

Name	Type	Application	Domain	Problem Sizes	Code Size
c-ray	K	Offline Raytracing	Computer Graphics	18 / 192 objects	500 LOC
md5	K	MD5 Calculation	Cryptography	various	1000 LOC
rgbcmy	K	Color Conversion	Image Processing	3.8 / 30.5 MP	700 LOC
rotate	K	Image Rotation	Image Processing	3.8 / 30.5 MP	1000 LOC
kmeans	K	k-Means Clustering	Artificial Intelligence	various	600 LOC
rot-cc	W	rotate + rgbcmy	Combined Workload	3.8 MP / 30.5 MP	1400 LOC
ray-rot	W	c-ray + rotate	Combined Workload	18 / 192 objects	1300 LOC
h264dec	A	H.264 Decoding	Video Processing	Full HD / QHD video	20000 LOC

Table 1. Benchmarks

6) rot-cc: As mentioned before, workloads consist of combinations of kernels. The first workload combines the *rotate* and *rgbcmy* kernels and is therefore called *rot-cc* (for rotation + color conversion). The parallelization structure is illustrated in Figure 4. Interesting cases are those where the rotation changes the image orientation (90 and 270 degrees) since this leads to a strided memory access pattern for the color conversion kernel.

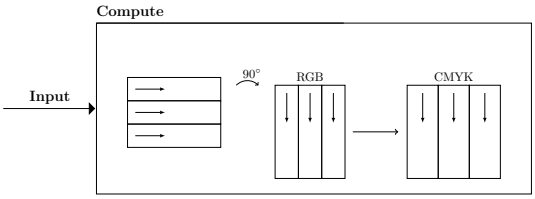


Figure 4. Parallel patterns for rot-cc workload

7) ray-rot: By chaining the *c-ray* and rotate kernels, we obtain the *ray-rot* workload. It exhibits additional functionlevel parallelism and is especially interesting because the two phases are highly different in characteristics and must, therefore, be load balanced to achieve high performance. *ray-rot* can be visualized as two chained stages of the pattern in Figure 1.

8) h264dec: h264dec is an H.264 decoder [5], derived from FFmpeg, a free, universal video transcoder [6].

For the H.264 decoder benchmark, parallelism is exploited at two levels: function-level and data-level parallelism (DLP). First, in the decoder stages, each stage can be performed in parallel in a pipeline fashion on different frames as shown in Figure 5. Additionally, DLP is exploited within the entropy (ED) and macroblock decoding (MBD) stages. This is illustrated in Figure 6. Here, hatched blocks denote data that can be processed in parallel.

3. Case Study: Pthreads vs. OmpSs

We now present performance results and the documentation of usage experiences comparing two programming models, Pthreads and OmpSs/SMPSs [10].

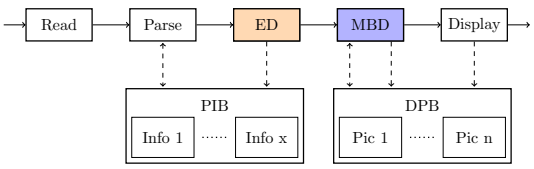


Figure 5. Pipeline parallelism in h264dec

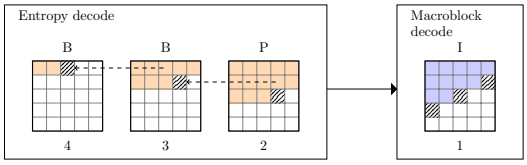


Figure 6. DLP parallelism in ED and MD stages

In Section 3.1, we describe the features of the programming models in comparison. After the experimental setup is presented in Section 3.2, Section 3.3 compares the speedup characteristics of the different benchmarks. From this comparison, we then derive information about the benchmark behavior and gain a first impression on how the two models compare to each other.

3.1. Evaluated Programming Models

The POSIX thread library [7] provides basic threading support for the C programming language. Synchronization is achieved using mutexes to protect critical sections and condition variables to achieve thread synchronization. The threads themselves have to be created, managed (i.e., set to a certain priority or in a detached state) and terminated explicitly. Pthreads thus fully leaves the management of the parallel algorithm to the programmer, enforcing him or her to consider dependencies, synchronization points, and possible race conditions in a direct, exposed way.

OmpSs/**SMPSs** [10] is the SMP instance of the *OpenMP SuperScalar model* (OmpSs). It is a novel task-based programming model which consists of

a runtime library and a source-to-source compiler. SMPSs requires the programmer to annotate functions as tasks using *#pragma css task* directives and label every task argument as an *input, output, inout,* or *reduction* parameter. These keywords declare an argument either read-only, write-only, read-write or as part of a reduction operation. Once such a task is created, it will be added to a runtime data structure, called the *task dependency graph*. The task graph is maintained and populated by the underlying runtime system which performs the dependency resolution and the scheduling of tasks on worker threads. This is similar to the way a superscalar processor dispatches instructions to available execution units. The only additional synchronization constructs SMPSs provides to the programmer are a *barrier* directive, which requires all previously created tasks to finish, a *wait on* directive, used to wait for a certain task to complete, and a mutex, which currently is required for reduction operations. An advantage of SMPSs is that the serial base code is maintained, allowing profiling and debugging of the sequential code with established tools. Its functionality can easily be regained by compiling an SMPSs program with a compiler not recognizing the preprocessor pragmas.

3.2. Experimental Setup

All available results have been obtained during the development of this benchmark suite and are therefore neither specifically optimized nor have been analyzed in detail. Their main objective at this stage is to classify and analyze the early benchmark behavior. Due to this early state of the described benchmarks, results for kmeans and h264dec are not yet included. Our evaluation platform is a 64-core cc-NUMA system with the following features:

- 8x Xeon X7560 (Nehalem EX architecture),
- 2.26 GHz clock frequency,
- HyperThreading disabled,
- 2 TB RAM,
- 204.8 GB/s aggregate memory bandwidth.

Each reported result is the average of three runs. Timing is done using timestamps inside the benchmarks and always excludes the I/O-phases (i.e., loading the input from disk into memory and cleaning up). Additionally, the execution time of all programs has been measured using both a small and a large input data set (see Table 1 for details).

3.3. Preliminary Scaling Results

In this section, the preliminary results for the Pthreads and SMPSs versions of the benchmarks are discussed. The speedup has been obtained by dividing the execution time on one processor by the execution time on n processors *for the same program*, thus normalizing the speedup factor for a single core to one.

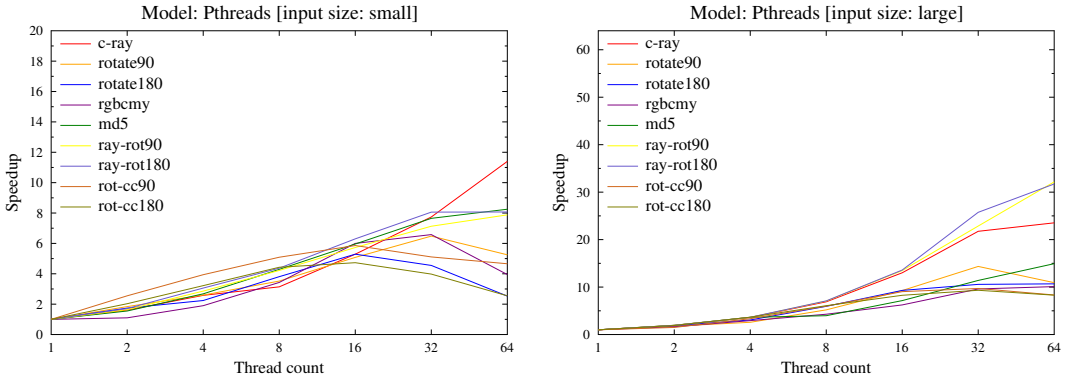
The speedup results for up to 64 cores for the Pthreads programming model using small input data sets are shown in Figure 7(a), the ones for large input data in Figure 7(b). The corresponding ones for SMPSs can be found in Figures 8(a) and 8(b).

The figures show that the behavior varies widely across the applications. For the highest thread count of 64, the Pthreads benchmarks achieve speedup factors ranging from 2.53x to 11.4x for the small and from 10.1x to 31.7x for the large input data sets. For the same number of threads, the SMPSs benchmarks achieve speedups between 1.7x and 33.4x for the small and 3.0x and 52.5x for the large input data sets. The differences observed between small and large input sizes are caused by the naturally higher amounts of DLP, leading to a coarsened granularity of the work units and thus diminishing the impact of the threading overhead. The results show that regarding only speedup and not real execution time, for these benchmarks, SMPSs performs on average two times better than Pthreads. For SMPSs, the runtime must be initialized before and shut down after any calls to it are made. This is excluded from the timing, while for Pthreads, thread creation is mostly tightly coupled with the actual execution and is therefore generally included. This is one of the reasons for the higher speedups of SMPSs.

The highest speedups are achieved for benchmarks which include ray tracing. This is expected since c-ray has a high computation-to-communication ratio. The performance of the Pthreads version of the c-ray kernel for large inputs saturates, however. In this case, performance increases by only 8% when going from 32 to 64 threads, compared to an average 70% for the other test cases (Figures 7(a), 8(a), 8(b)). This is fully reproducable and will be investigated further.

The lowest speedups are achieved by benchmarks which include the rotate kernel (and do not also include ray tracing). The reason for this supposedly is the cc-NUMA evaluation platform. Because there is only a small amount of computation in the rotate kernel, increasing the thread count does not improve performance but instead leads to memory contention, causing a high amount of (coherence) traffic. This is especially the case for the transition from 32 to 64 threads where four additional processor sockets are used for 64 threads, resulting in a lower speedup than for 32 threads.

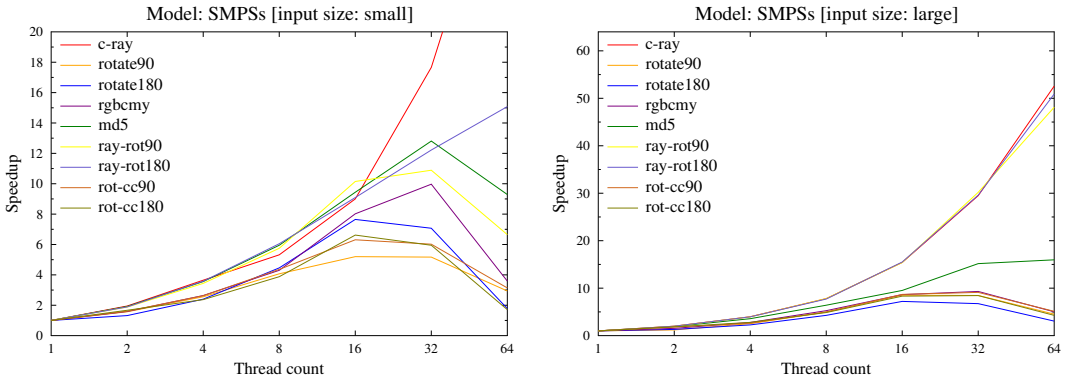
A deeper analysis of these observations and further machine-specific optimizations are future work.



(a) Small input size

(b) Large input size

Figure 7. Baseline performance for Pthreads



(a) Small input size

(b) Large input size

Figure 8. Baseline performance for OmpSs/SMPSs

4. Related Work

Benchmark suites have been developed previously, including several proprietary, domain-specific products [2, 16]. They are, however, mainly non-compliant with the portability concept our work chooses to follow.

PARSEC [3, 4] is a recent benchmark suite consisting of 12 programs. The target platforms of PARSEC originally are chip multiprocessors, however, the programs included in the suite are not inherently limited to this usage scenario. The stated goal of PARSEC is to discover new trends in the research and development of parallel machines, algorithms and applications. Featured for all benchmarks are variants for Pthreads, OpenMP and Intel Thread Building Blocks. However, PARSECs set goal is also to provide a fix framework for benchmark execution, input data size control and installation. This complicates the processes of extending the suite quickly or extracting an application out of it for further, isolated use.

Apart from benchmark suites, previous work also includes attempts particularly targeted at the evaluation of parallel programming models.

Podobas et al. [11] performed an evaluation of three taskbased parallel programming models, OpenMP, Cilk++ and Wool. They focus on leveraging the performance characteristics of these parallel programming models, studying in detail the cost of creating, spawning and joining tasks as well as overall performance. The results are limited to only three programming models, only kernel-type programs and only performance characteristics. Moreover, the work excludes the extension to new programming models and therefore is, in contrast to our work, not portable.

Ravela [14] presents an evaluation of Intel TBB, Pthreads, OpenMP and Cilk++, containing results for both achieved performance and the time required to develop the respective versions of the benchmarks. All used benchmarks are, however, taken from the domain of high performance computing, resulting in limited relevance for different application domains.

5. Conclusions and Future Work

In this paper, we presented a benchmark suite to evaluate the programmability and performance of emerging parallel programming models. To achieve this, we focused on a structured, portability-focused, fixed-

parallelism approach. We analyzed the intended usage of the suite, thereby compiling a set of requirements which must be met by a benchmark suite aimed at evaluating parallel programming models. We presented and described an early collection of such benchmarks, covering a wide range of application domains, and used them in case study, comparing an established with an emerging programming model.

The preliminary experimental results obtained in this process have shown a wide range of characteristics for the chosen benchmark set, especially giving insights about the behavioral properties of those benchmarks and producing valuable information on the scaling and speedup characteristics of the two analyzed programming models. This study must also be extended to additional types of parallel machines, for example heterogeneous architectures or large chip multiprocessors. Such an investigation could also include a detailed analysis of the statistical features of each benchmark, resulting in concrete measures for properties such as bandwidth usage, arithmetic complexity or memory size requirements.

Naturally, our benchmark suite will be subject to extension. As mentioned in Section IV, porting suitable, existing opensource benchmarks from other collections to this suite is ongoing work. Furthermore, we seek to extend the suite with additional, industry-relevant applications in order to gain key insights on how modern programming models fare when used in large, real-world applications. Benchmarks currently being considered are POV-Ray [1] or a game engine. Aside from adding more benchmarks to the suite, another goal is to evaluate more programming models to gather more experiences in using the suite. Evaluating a larger number of programming models is naturally an advantage because it will provide more references to compare to when evaluating new programming models.

Acknowledgment

This research has been supported in part by the European Communitys Seventh Framework Programme $[FP7/2007 - 2013]$ under the ENCORE Project (www.encore-project.eu), grant agreement n^{o} 248647 [13], and the Future SOC Lab of Hasso-Plattner-Institute Potsdam [12].

References

[1] Persistence of vision ray tracer. http://www.povray.org/, 2011.

[2] M. Berry. Public international benchmarks for parallel computers: Parkbench committee: Report-1. Technical report, Scientific Program, 1994.

[3] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, 2011.

[4] C. Bienia and K. Li. PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.

[5] C. C. Chi and B. Juurlink. A QHD-Capable Parallel H.264 Decoder. In *Proceedings of the 25th International Conference on Supercomputing*, 2011.

[6] FFmpeg group. http://www.ffmpeg.org/ffmpeg.html, 2011.

[7] IEEE Opengroup. Portable Operating System Interface. 2004.

[8] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[9] R. Narayanan, B. Özisikyilmaz, J. Zambreno, G. Memik, and A. Choudhar. Minebench: A benchmark suite for data mining workloads. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2006.

[10] J. Perez, R. Badia, and J. Labarta. A Dependency-Aware Task-Based Programming Environment for Multi-Core Architectures. In *IEEE International Conference on Cluster Computing*, 2008.

[11] A. Podobas, M. Brorsson, and K.-F. Faxén. A comparison of some recent task-based parallel programming models. In *Third Workshop on Programmability Issues for Multi-Core Computers*, 2010.

[12] H.-P.-I. Potsdam. Future soc lab. http://www.hpi.uni-potsdam.de/forschung/future soc lab.htm, 2011.

[13] E. Project. Enabling technologies for a future many-core., 2011.

[14] S. C. Ravela. Comparison of shared memory based parallel programming models. Master's thesis, Biekinge Institute of Technology, 2010.

[15] R. Rivest. The MD5 Message-Digest Algorithm. 1992.

[16] Standard Performance Evaluation Corporation. SPEC Benchmark Suite. http://www.spec.org/index.html, 2011.

[17] J. Tsiombikas. http://www.futuretech.blinkenlights.nl/c-ray.html, 2010.

Towards Multi-Core and In-Memory for IDS Alert Correlation: Approaches and Capabilities

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 usually addressed by correlation and clustering of alerts. As real-time analysis is crucial for security operators, 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. We identify the data storage and processing algorithms to be the most important factors influencing the performance of clustering and correlation. We implement memory-optimized algorithms and column-oriented or In-memory databases for correlation and clustering in an extensible IDS correlation platform, which leads to significant improvements of the performance. The platform supports multi-core frameworks, such as OpenCL and MapReduce. The efficiency of the proposed platform is tested by practical experiments with several alert storage approaches and different simple algorithms.

1 Alert Correlation and its Performance

The alert correlation framework usually consists of several components [4]: *Normalization*, *Aggregation (Clustering)*, *Correlation*, *False Alert Reduction*, *Attack Strategy Analysis*, and *Prioritization*. Over the last years, alert correlation research focused on new methods and technologies for these components. IDMEF [5] and CVE [6] are important efforts in the field of *Normalization*. Approaches of aggregation are mostly based on similarity of alerts or generalization hierarchies. The correlation algorithms [4] can be classified as: *Scenario-based correlation*, *Rule-based correlation*, *Statistical correlation*, and *Temporal correlation*. Most of the efforts do not consider the aspect of performance, which is needed in case of huge amounts of alerts, as well as the scenarios requiring real-time.

The efficiency of the correlation depends on the quality and performance of the algorithm as well as the storage and organization of original alerts. The quality is a measure of the correctness of the algorithm and depicts how many of the recognized correlations are correct, i.e., how many of the correlations found represent existing relations between alerts. Furthermore, it depicts how many of the existing relations between alerts are found by the algorithm. The performance of the correlation describes the amount of time needed to correlate a number of alerts. Due to the complexity of large scale networks, the amount of alerts increases significantly. Therefore, the performance of correlation algorithms is a major aspect of the efficiency of correlation.

The work described in [27] considers the performance of alert correlation by using a memory-based index for hyper alerts. A hyper alert is a cluster of alerts with the same properties, e.g., the same source address or target address. The approach using index tables is introduced in [11]. To perform correlation in real-time, the approach filters and clusters alerts to hyper alerts, which reduces the number of processed alerts significantly. However, this technique may lead to inaccurate results of the correlation, as multiple alerts are generalized to a single hyper alert. The approach reaches a correlation rate on the order of $100,000$ alerts per second based on the massive reduction of alerts by clustering in hyper alerts. In [2] the *data storage* and the *processing algorithms* have been identified to be the

most important factors influencing the performance of clustering and correlation. The platform introduced in [2] considers different storage mechanisms and can handle up to $100,000,000$ for specific algorithms that make heavy use of the caching mechanisms of the platform. For storage, a column-based database, an In-Memory alert storage, and memory-based index tables lead to significant improvements of the performance. Although this work considers data and task distribution in general, the platform is not mature enough to distribute one correlation algorithm over multiple computing cores. Furthermore, using a hybrid memory architecture and GPU based computing is not considered.
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. The existing IDS correlation platform was extended for multi-core processing. We conducted preliminary practical experiments using the existing platform with known algorithms. Apart from the practical achievements, we have been able to publish papers on the correlation platform [2] and started research on a complex correlation algorithm using attack graph data and environmental information for IDS correlation [1]. Some results are summarized in the following subsections.
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. Furthermore, we conducted some tests and experiments using the NVIDIA FluiDyna System. The following feature set has been realized:

- Snort alert generator that generates IDS alerts using a network description
- 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
- Multi-core support for OpenCL and MapReduce
- Visualization of correlation results

2.1 Parallel Processing in IDS Correlation

Parallel processing is useful in all steps of the correlation. In most of the steps, the possibility to parallelize the computing depends on the algorithm used to perform this step. The preferred technology for implementing the steps of the correlation highly depends on the chosen algorithm. While algorithms that can be expressed in the MapReduce model can use a data parallel or task-parallel approach and are easier to implement, the OpenCL implemented algorithms are data-parallel and support GPU processing which might be more efficient for specific tasks. A correlation platform needs to support multiple ways for multi-core processing to be useful for most of the algorithms in the correlation framework.
The normalization is data-parallel, as each event can be processed independently. Let $\mathcal{A}$ be the set of IDS alerts in IDMEF format. Let $\mathcal{C}$ be the set of events in CEE [7] format. The function $n : \mathcal{A} \rightarrow \mathcal{C}$ is a mapping from $\mathcal{A}$ to $\mathcal{C}$ and needs only one parameter as input.

$$n(a_i) = c_i | a_i \in \mathcal{A} \wedge c_i \in \mathcal{C} \tag{1}$$

Let $C \subset \mathcal{C}$ be the set of events that is supposed to be aggregated. Let $\Upsilon_k(x, y)$ be an expression with $x \in \mathcal{C}, y \in \mathcal{C}$ that represents the aggregation condition. The relation R_C is defined as:

$$R_C = \{(x, y) \in C^2 : \Upsilon_k(x, y)\} \tag{2}$$

R_C^* defines an equivalence relation on the transitive closure of R_C. The alert aggregation combines alerts that are similar and being created together in a short time, i.e., the difference of the timestamps is below a certain threshold. It defines a set of equivalence classes $C_{/R_C^*}$ over the equivalence relation R_C^*. As the aggregation needs to be expressed as an relation between two events, the processing needs to be on specific sets of events. Thus, parallelizing the process needs additional efforts and might not be possible for each $\omega_k(x, y)$. For instance, if the aggregation condition is true for events with the same message, a classical data-parallel approach could be used. If the aggregation condition is true for events with the same message and with a small difference in timestamps, a data-parallel based approach might not be easy to implement. A fragmentation of the original data set in

chunks with events that happened in the same time interval might be a better solution. A task-parallel approach might be used for aggregating events based on different conditions.
The correlation of events is basically a creation of relations between events and thus is similar to the aggregation in terms of parallelization. For instance, a statistical correlation requires counting of similar events and thus might be handled with a data-parallel approach and merging of the counted results. Correlation algorithms are mostly based on specific machine learning algorithms where each computation result depends on input data and on former computation results, a data-parallel approach might not be feasible or easy to implement. The same holds for False Alert Reduction, Attack Strategy Analysis, as well as Prioritization.
As the different steps of the correlation process might be parallelized in different ways, a correlation platform is required to support parallelization approaches natively. Furthermore, programming paradigms, such as MapReduce, can simplify the implementation of the algorithm. Hardware independent implementations of multi-core support, such as OpenCL, provides a high flexibility in deploying the system in different environments.

2.2 A Correlation Platform for Parallel Processing

It is expected that the proposed IDS correlation platform can support both, batch mode and stream mode, as described in [9]. The batch mode is useful for forensic analysis and operates on a fixed set of alerts. The stream mode is useful for realtime analysis for security operations and monitoring of the current state of the network. Obviously, the multi-core functionality helps for the batch mode, as the fixed data set can be parted in chunks and processed in a data-parallel way. Furthermore, the multi-core support can be useful for the stream mode when task-parallel processing is possible. Different independent correlation algorithms can be executed simultaneously on the same set of incoming alerts. The platform should provide an easy-to-use interface for programming and uploading modules. Moreover, it should be capable of connecting different kinds of sensors as well as other management systems by using a unified format for alerts and messages, i.e., IDMEF [5] or CEE[7].
As shown in Figure 1, the platform runs correlation modules as plugins, which improves the extensibility and flexibility of the platform. Each correlation module has the possibility to make use of OpenCL by defining own kernels that are loaded with the module. As soon as the module makes use of OpenCL kernels, it is automatically handled by the OpenCL implementation, either NVIDIA CUDA or ATI Stream. As the ATI Stream API supports both, graphical processing units (GPU) as well as central processing units (CPU), it is used as the default processing API for OpenCL kernels. Each correlation module runs autonomously as a thread on the Java Virtual Machine (JVM), which is capable of distributing tasks to existing CPUs. The system uses an internal event storage to process events in batch mode. Additionally, the EventStreamReader receives events from IDS sensors and log gatherers at runtime and enables processing in stream mode. The Mutli-Core Interface is responsible for compiling as well as triggering the execution of the OpenCL kernels for each correlation module.
The event storage is an in-memory database that holds all existing events in memory. The number of possible events in the memory depends on the size of the random access memory (RAM) on the hardware the correlation platform is running on. As RAM is getting cheaper and there are systems with up to 2TB of RAM, this approach is now possible. On the running prototype, a basic alert needs about 152 Bytes of allocated memory. The hardware bases of our prototype is a *Hewlett Packard DL980 G7* with 2 TB of main memory, i.e., $2,199,023,255,552$ Bytes. In the ideal situation where we are able to use the full amount of memory for alerts, it is possible to store $1.44672583 * 10^{10}$ alerts.

3 AG-based Correlation Algorithm using HMM

A more complex correlation algorithm uses attack graphs to correlate alerts. Attack Graphs have been proposed as a formal way to simplify the modeling of complex attacking scenarios. Based on the interconnection of single attack steps, they describe multi-step attacks. Attack Graphs not only describe one possible attack, but many potential ways for an attacker to reach a goal. In an attack graph, each node represents a single attack step in a sequence of steps. Each step may require a number of previous attack steps before it can be executed, denoted by incoming edges, and on the other hand may lead to several possible next steps, denoted by outgoing edges. With the help of attack graphs most of possible ways for an attacker to reach a goal can be computed. This takes the burden from security experts to evaluate hundreds and thousands of possible options. Thus, a program can identify weak spots much faster than a human. At the same time, representing attack graphs visually allows security personal a faster understanding of the problematic pieces of a network [24, 25].
In this paper, we adopted the AG based correlation algorithm as described in [3]. The algorithm consists of three simple steps: 1) the preparation, 2) the mapping, and 3) the building of an alert dependency graph. In the preparation phase, all necessary information is loaded, i.e., the system and network information is gathered, the database with alert classifications is imported, and the attack graph for the network is loaded.

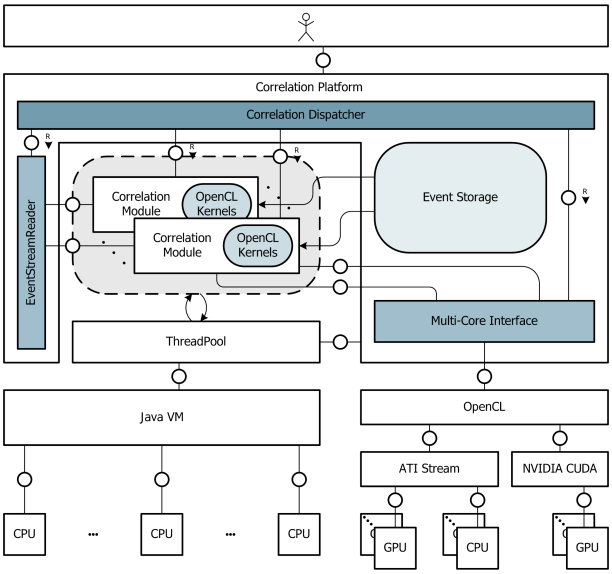


Figure 1. Mutli-Core Platform Architecture

We use a similar mapping function as it is described in [26], i.e., alerts are mapped to match with specific nodes in the attack graph. An alert matches if the alert type matches, e.g., when the alert refers to an existing vulnerability in the network, and the source and target address of the alert fit to the node in the attack graph. To build an alert dependency graph, we are using a Breadth First Search (BFS) on the attack graph and create a dependency between alert a_i and a_j, if $t(a_i) \leq t(a_j)$ and a_j is mapped to a node that can be reached from the node that a_i is mapped to ($t(a)$ returns the creation time of the alert a). Each path in the alert dependencies graph identifies a subset of alerts that might be part of an attack scenario. The alert dependency graph can be used for further analysis, e.g., ranking of attack scenarios or manual evaluations of attack scenarios.
To identify the most probable path in the attack graph based on the observed events, a Hidden Markov Model (HMM) is used and the Viterbi path is calculated. By defining an attack graph with its notes as Markov chain of events, the mapping function defines a set of possible observations. The log events are a set of actual observations that can be the input for the Viterbi algorithm that is supposed to find the most probable set of nodes in the HMM that lead to the observations made.

4. Future Work

Within the next few months, we want to prepare 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
- Improve multi-core support
- Implement more algorithms with multi-core support
- Research on correlation algorithms that are using environment information and attack graphs
- Research on visualization techniques for correlation results

References

[1] 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).

[2] Roschke, S., Cheng, F., Meinel, Ch.: An Alert Correlation Platform for Memory-Supported Techniques. In: Concurrency and Computation, Wiley Blackwell, 2011 (to appear).

[3] Roschke, S., Cheng, F., Meinel, Ch.: A New Correlation Algorithm based on Attack Graph. In: Proceedings of the 4th Conference on Computational Intelligence in Security for Information Systems (CISIS'11), Springer LNCS 6694, Torremolinos, Spain, pp. 58-67 (2011).

[4] 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).

[5] 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).

[6] Mitre Corporation: *Common vulnerabilities and exposures (CVE)*, WEBSITE: http://cve.mitre.org/ (accessed Apr 2011).

[7] Mitre Corporation: *Common Event Expression (CEE)*, WEBSITE: http://cee.mitre.org/ (accessed Apr 2011).

[8] 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).

[9] 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).

[10] 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).

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

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

[13] Breshears, C.: *The Art of Concurrency*, O'Reilly Media, Sebastopol, CA, USA (2009).

[14] Khronos Group OpenCL, WEBSITE: http://www.khronos.org/opencl/ (accessed Apr 2011).

[15] Java OpenCL Library (JavaCL), WEBSITE: http://code.google.com/javacl/ (accessed Apr 2011).

[16] NVIDIA CUDA, http://www.nvidia.com/object/cuda_home_new.html (accessed Apr 2011).

[17] ATI Stream Technology, WEBSITE: http://www.amd.com/us/products/technologies/stream-technology/Pages/stream-technology.aspx (accessed Apr 2011).

[18] Dean, J., Ghemawat, S.: *MapReduce: Simplified Data Processing on Large Clusters*, In: Communications of the ACM, ACM Press, 51(1), pp. 107-113 (January 2008).

[19] Yoo, R.M., Romano, A., Kozyrakis, Ch.: *Phoenix Rebirth: Scalable MapReduce on a Large-Scale Shared-Memory System*, In: Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'09), IEEE Press, Austin, TX, pp. 198-207 (September 2009).

[20] Mao, Y., Morris, R., Kaashoek, F.: *Optimizing MapReduce for Multicore Architectures*, Technical Report MIT-CSAIL-TR-2010-020, MIT, 2010.

[21] Apache Hadoop Project, WEBSITE: http://hadoop.apache.org/ (accessed Apr 2011).

[22] Arnes, A., Valeur, F., Vigna, G., Kemmerer, R.: Using Hidden Markov Models to Evaluate the Risks of Intrusions: System Architecture and Model Validation. In: *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID'06)*, Springer LNCS 4219, Hamburg, Germany, pp. 145-164 (2006).

[23] MacQueen, J.B.: Some Methods for classification and Analysis of Multivariate Observations. In: *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, University of California Press, pp. 281-297 (1967).

[24] Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated Generation and Analysis of Attack Graphs. *Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P'2002)*, IEEE Press, Los Alamitos, CA, pp. 273-284 (2002).

[25] Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. *Proceedings of Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC 2004)*, ACM Press, New York, NY, USA, pp. 109-118 (2004).

[26] Wang L, Liu A, Jajodia S. Using attack graphs for correlation, hypothesizing, and predicting intrusion alerts. *Computer Communications*, 29(15), pp. 2917-2933 (April 2006).

[27] 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).

Elastic VM for Dynamic Virtualized Resources Provisioning and Optimization

Christoph Meinel, Wesam Dawoud, and Ibrahim Takouna
Hasso Plattner Institute (HPI)
University of Potsdam
Potsdam, Germany
{christoph.meinel, wesam.dawoud, ibrahim.takouna}@hpi.uni-potsdam.de

Abstract

Rapid growth of E-Business and frequent changes in websites contents as well as customers interest make it difficult to predict workload surge. To maintain a good quality of service (QoS), system administrators must provision enough resources to cope with workload fluctuations considering that resources overprovisioning reduces business profits while underprovisioning degrades performance. In this project, we present elastic system architecture for dynamic resources management and applications optimization in virtualized environment. In our architecture, we have implemented three controllers: CPU, Memory, and Application. These controllers run in parallel to guarantee efficient resources allocation and optimize application performance of co-hosted Virtual Machines (VMs) dynamically. We evaluated our architecture with extensive experiments and several setups; the results show that considering online optimization of application, with dynamic CPU and Memory allocation, can reduce service level objectives (SLOs) violation and maintain application performance.

1. Introduction

Later advance in virtualization technology software, e.g. Xen [1] and VMware [6], enabled cloud computing environment to deliver agile, scalable, elastic, and low cost infrastructures. However, current implementation of elasticity in "Infrastructure as a Service" cloud model considers VM as a scalability unit. In this project, we developed an automated dynamic resources provisioning architecture to optimized resources provisioning in consolidated virtualized environments (e.g., Cloud computing). Unlike current implementation of elasticity in cloud infrastructure, we replaced the VM (i.e., coarse-grain scalability unit) with fine-grain resources units (i.e. CPU%, Memory in MB). Our Elastic VM is scaled dynamically in-place to cope with workload fluctuations. Furthermore, the hosted application is also tuned after each scaling to maintain predetermined (SLOs). As a use case, we implemented our approach into Xen environment and used Apache web server as an application. Our SLO in this project is to keep the response time of the web requests less than a specified threshold.
The key contributions of this work are as follow: First, we have studied Apache application performance under different configuration and different CPU and Memory allocation values. Second, we have developed a dynamic application optimization controller for Apache application to maintain the desired performance. Third, we built CPU and Memory controllers based on [2]. Fourth, we built elastic system architecture that join CPU, Memory, and application optimization controllers for managing consolidated virtualized environments. Finally, the elastic system architecture has been evaluated with extensive experiments on several synthetic workload and experimental setups. Our results show that elastic system architecture can guarantee the best performance for application in terms of throughput and response time.

2. Elastic VM architecture

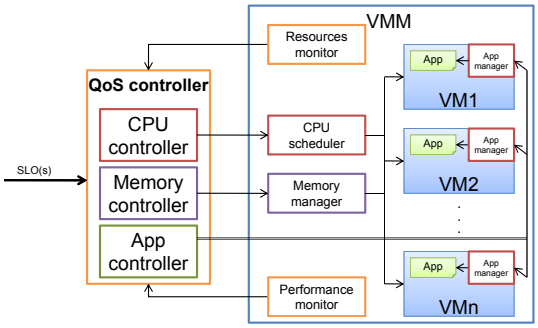


Figure 1. Elastic VM architecture

Our architecture has main component "QoS controller" which communicates with many other mod-

ules implemented into the Virtual Machine Manager (VMM) and VMs levels as the following:

- Resources monitor module dynamically measures the resources consumption and updates the QoS controller with new measurements.
- CPU scheduler is implemented to dynamically change the CPU allocation of the VMs according to determined values by QoS controller, this module depends on Xen credit scheduler as an actuator for setting the CPU shares for VMs.
- Memory manger is implemented with the help of balloon driver in Xen to allows online changing of VMs Memory.
- Performance monitor also keeps the controller up to date with performance metrics, i.e. the average response time and the throughput.
- Application manager (App manager) is implemented into VM level, its job is to get new MaxClients value from the Application controller (App controller), to update the Apache configuration file, and then to reload Apache gracefully.

On the left side of figure 1 is the QoS controller; the controller has (SLOs) as inputs and the proposed CPU capacity, proposed Memory allocation, and proposed MaxClients as outputs. In our approach the main SLO is to keep average response time of Apache web server into specific value regardless of the workload fluctuations. For this purpose, we implemented three controllers to run in parallel, these controllers are as the following:

CPU controller: Which is a nested loop controller developed in [8]. The inner controller (CPU utilization controller) is an adaptive-gain integral (I) controller was designed in [7]:

$$a_{cpu}(k+1) = a_{cpu}(k) - K_1(k)(u_{cpu}^{ref} - u_{cpu}(k)), \quad (1)$$

Where

$$K_1(k) = \alpha . c_{cpu}(k)/r_{cpu}^{ref} \quad (2)$$

The controller is designed to predict the next CPU allocation $a_{cpu}(k+1)$ depending on last CPU allocation $a_{cpu}(k)$ and consumption $c_{cpu}(k)$, where the last CPU utilization $u_{cpu}(k) = c_{cpu}(k) = a_{cpu}(k)$. The parameter α is the constant gain that determines the aggressiveness of the controller. In our experiments, we set β=1.5 to allow the controller to aggressively allocate more CPU when the system is overloaded, and slowly decrease CPU allocation at low workload. The disadvantage of this controller is that, it implies determining the reference utilization u_{cpu}^{ref} manually. However, this is not practical while as seen in figure 2, the response time does not only depend on CPU utilization, but also on the request rate, which changes frequently. So, it is more realistic to have u_{cpu}^{ref} value automatically driven by the application's QoS goals rather than being chosen manually for each application. For this goal, another outer loop controller (RT controller) is designed [8] to adjust the u_{cpu}^{ref} value dynamically. The outer loop controller can be interpreted as the following:

$$u_{cpu}^{ref}(i+1) = u_{cpu}^{ref}(i) + \beta(RT_{cpu}^{ref} - RT_{cpu}(i))/RT_{cpu}^{ref} \quad (3)$$

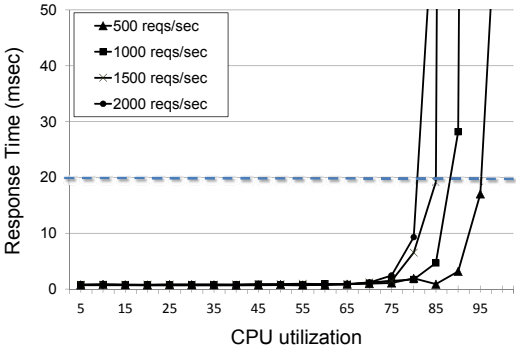


Figure 2. Mean response time vs. CPU utilization under

Where $u_{cpu}^{ref}(i+1)$ is the desired CPU utilization, $RT_{cpu}(i)$ is the measured response time, and RT_{cpu}^{ref} is the desired response time determined by SLO. The outer controller (RT controller) ensures that the value fed to the CPU controller is always within an acceptable CPU utilization interval $[U_{min}; U_{max}]$.
In our experiments, we set β=1.5. The CPU allocation is limited to the interval [10, 80], and the CPU utilization is also limited to the interval [10, 80]. The desired response time (RT) in all our experiments is 20 milliseconds.

Memory controller: In our experiments we noticed that increasing the number of Apache processes can increase the throughput. However, at some level, the performance is degraded drastically when the Apache processes consumed the whole available Memory. At that point, system starts swapping Memory contents into the hard-disk. This behavior adds more workload to the CPU, which is already overloaded by the big number of the processes. To keep the system away from bottlenecks, we implemented the Memory controller, designed in[2], to keep the CPU controller runs in an operating region away from the CPU contention:

$$a_{mem}(i+1) = a_{mem}(i) + K_2(i)(u_{mem}^{ref} - u_{mem}(i)) \quad (4)$$

Where

$$K_2(i) = \lambda u_{mem}(i)/u_{mem}^{ref} \quad (5)$$

The controller aggressively allocates more Memory when the previously allocated Memory is close to saturation (i.e. more than 90%), and slowly decreases

Memory allocation in the low workload region. Along our experiments, we set u_{mem}^{ref}=90%, λ=1, and the limits of the controller are [64, 512], where the 64 is the minimum allowed Memory allocated size, and the 512 is the maximum allowed allocated Memory size.

Application controller: after extensive experiments and monitoring of Apache behavior, we found that there was a specific value of *MaxClients* that gives the best throughput and the minimum response time as seen in figure 3.

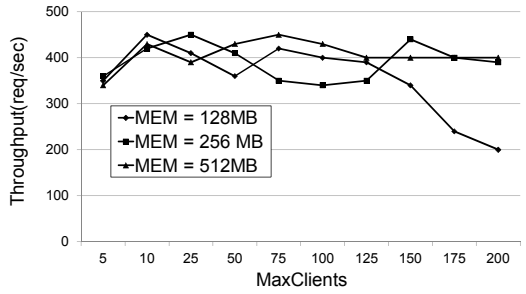


Figure 3. Throughput vs. MaxClients under different hardware settings

Finding the optimum value of *MaxClients* was examined by former research e.g. [3]. Unfortunately, these optimization methods are not applicable to our case for many reasons: First, we have a dynamic resources. So, it will be difficult to dynamically determine the new optimum *MaxClients* value for each new resources allocation. Second, we dont have the chance to run an active optimization using our generated traffic, because this could influence the real service performance. Third, the optimum value is affected by traffic type and CPU utilization.

On the light of the mentioned problems, we designed our heuristic Apache controller to find the best *MaxClients* value passively (i.e., depending on the real traffic). The Apache controller monitors four measured values to determine the best *MaxClients*: response time, throughput, CPU utilization, and number of running Apache processes. The controller saves the best record of these values. The best record is calculated by finding the record that satisfies the QoS response time metric and gives the highest throughput with less CPU utilization. With each new measurement of monitored values, Apache compares the current record with the best record, if it is better; the current record will be saved as the best record. While it is running, if the Apache noticed a violation of QoS metrics (response time in our case) it tries to predict the problem by the following rules:

Rule1: Apache processes starving problem: Apache processes starving problem occurs when Apache server runs big number of processes. As a result, CPU spends most of the time switching between these processes while giving small slot of the time to each process. Such behavior causes requests to spend longer time in application queue, which end up with high response time and many timed-out requests. To eliminate this problem, the Apache controller reloads the Apache server with the last best record.

Rule2: Resources competition problem: The competition on resources is predicted by Apache controller as the following: response time increases, number of running apache processes reaches MaxClients value, and at the same time CPU utilization decreases (i.e. less than 90%).

As seen above, with both rules, the proposed Apache controller will not only look for the optimum MaxClients value, but also will eliminate performance bottlenecks by keeping history of the last best running configurations.

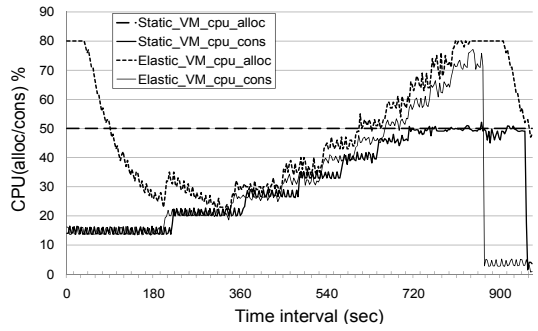
3. Experimental Setup in Future SOC Lab

Our experiment conducted on a testbed of two physical machines (Client and Server). Server machine (Future SOC Lab machine) is Fujitsu PRIMERGY RX300 S5 server with 4-cores, it is equipped with 12GB physical memory. Server runs Xen 3.3 with kernel 2.6.26-2-xen-686 as hypervisor. On the hypervisor are hosted VMs with Linux Ubuntu 2.6.24-19. These VMs run Apache 2.0 as a web server in prefork mode. For workload generation, httperf tool [4] is installed on client machine. In the following experiments we deal with three VMs setup: First, Static VM, which is a virtual machine initialized with 512MB of RAM and limited to 50% of the CPU capacity. Second, Elastic VM with CPU/Memory controllers, it is a VM controlled with the CPU and Memory controllers seen in equations 1 to 5, the CPU limits of this machine is 80of CPU capacity, and the Memory is 512MB of RAM. Third, Elastic VM with Apache, it has the same setup of first VM except that it is equipped with our Apache controller in addition to CPU and Memory controllers. In all our experiments, SLO is to keep response time threshold (RT threshold) less than 20 milliseconds.

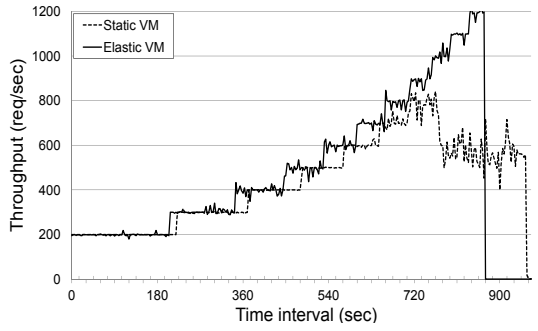
3.1. Static VM vs. Elastic VM response to step traffic

In this experiment, we would like to study our Elastic VM ability to cope with traffic change to maintain the specified SLO. As a basis for our experiments; we used dynamic web pages requests, in each request, the web server executes a public key encryption operation to consume a certain amount of CPU time. The step traffic initiated with the help of autobensh tool [4], it started with 20 sessions, each session contains 10 connections. The number of sessions increases by 10 with each load step. The total number of connections for each step is 5000, and the timeout for the request is 5 seconds. Throughput result from the generated web

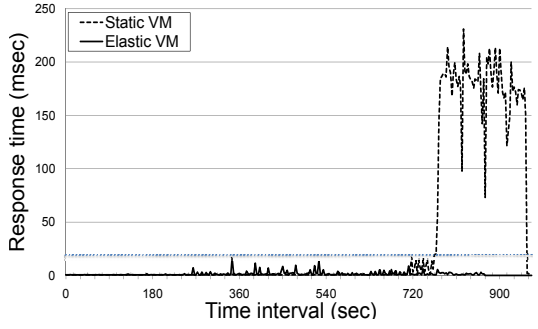
traffic is seen in figure 4(b).
Each step of the graphs in figure 4(b) represents the throughput of a specific traffic rate. For instance, along the period 0 to 780 seconds; both VMs respond to requests successfully without any lost or time-out. On the other hand, after 780 seconds, the Static VMs CPU is saturated, which caused requests to wait longer in the TCP accept queue, and consequently increased response time. The long queuing results in a continues period of SLO violation as seen in figure 4(c).



(a) CPU consumption



(b) Throughput



(c) Response time

Figure 4. Static VM vs. Elastic VM response to step traffic

The percentage of timed-out requests with the corresponding traffic rate is illustrated in table 1. The table started at 900 req/sec because there was no significant timed-out traffic before this rate. If compared to the Elastic VM for the same high traffic rate (i.e. 800 to 1200 req/sec), figures 4(a) to 4(c) show how the Elastic

Requests rate(req/sec)	Static VM (timeout %)
900	7.232
1000	15.328
1100	18.258
1200	27.772

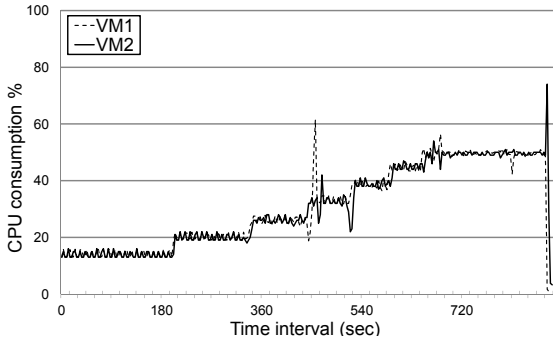
Table 1. The timeout started after the Static VM received 900 req/sec.

VM was able to borrow more resources dynamically, serve more requests, maintain a low response time, and prevent SLO violation.

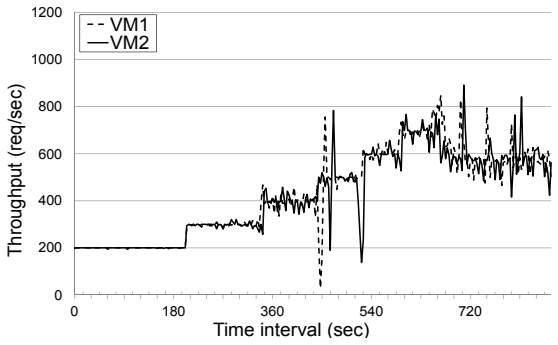
3.2. Two Elastic VMs compete on resources

In the previous experiment, we studied the ideal case where the host was able to satisfy the Elastic VMs need for more resources to cope with the increase of incoming requests. In this experiment, we study the competition on the CPU between two Elastic VMs. To raise this competition, we pinned the virtual CPUs of two Elastic VMs into same physical core. To clarify the benefits of Apache controller usage, the step-traffic has been run two times simultaneously to both Elastic VMs, one time without Apache controller and another time with Apache controller.
The first part of the experiment is illustrated in figures 5(a) to 5(c). Figure 5(b) shows that Elastic VMs were not able to cope with the traffic rate higher than 800 req/sec while the host committed only 50% of the CPU power for each VM starting from second #660, as seen in figure 5(a). The reason behind this fair sharing is Xen credit scheduler. During this experiment, we setup the scheduler with the same share for running VMs. According to competition on CPU, many requests are queued for a long time causing high response time and continues violation of SLO, as seen in figure 5(c). Moreover, many other requests are timed-out before being served as seen in second and third columns of table 2. From the above experiments, we can conclude that Elastic VM can improve the performance if the host has more resource to redistribute, but in case of competition on resources, under the fair scheduling, Elastic VM (without) Apache controller merely behaves as a Static VM.
The previous experiment is repeated on two Elastic VMs (with) Apache controller. Figure 6(a) shows that in spite of the limited CPU capacity (50%) available to each VM, starting from second #660, apache controller does two improvements: First, the moment of the Apache reload is a good chance for the other Apache server to have more processing power and serve more requests. Second, after the reload, the Apache servers are tuned with a new MaxClients value, if this value achieved better performance, the Apache controller will keep it, otherwise it will con-

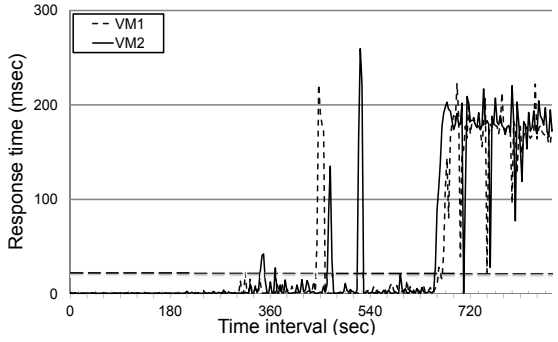
tinue looking for more optimum value. Third and fourth columns of table 2 show the improvemrent in terms of reductions of timed-out traffic and SLO violation.



(a) CPU consumption

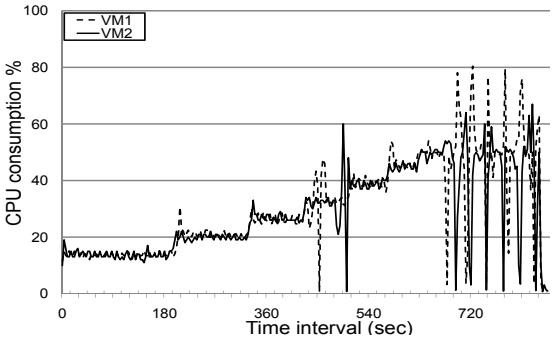


(b) Throughput

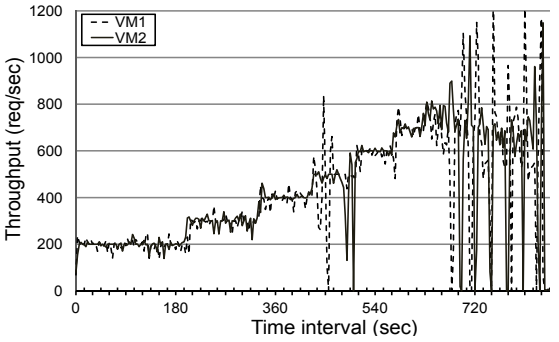


(c) Response time

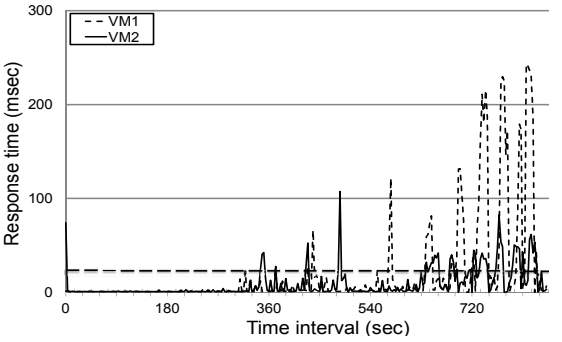
Figure 5. Two Elastic VMs (without) Apache controller responding to step traffic



(a) CPU consumption



(b) Throughput



(c) Response time

Figure 6. Two Elastic VMs (with) Apache controller responding to step web traffic

	VM1	VM2	VM1	VM2
(req/sec)	Timeout(without)	Timeout(with)		
800	4.0%	0%	0%	0.2%
900	13.3%	23.8%	8.8%	8.2%
1000	20.5%	23.2%	16.52%	17.0%
1100	25.0%	35.0%	21.0%	22.0%
1200	31.0%	37.0%	26.2%	27.8%
	Violation(without)		Violation(with)	
	23.9%	26.4%	14.7%	16.8%

Table 2. Two Elastic VMs (without) Apache controller vs. two Elastic VMs (with) Apache controller responding to step traffic

4. Conclusions & Future work

In this part of the project, we have presented an implementation for elastic system architecture for optimizing resources consumption in consolidated environments. Our system includes three controllers CPU, Memory, and Application run in parallel to preserve SLO. We have evaluated our system in a real Xen based virtualized environment; the experiments show that using Application controller maintains the performance and mitigates SLO violation and requests time-

out. Our immediate future work will include analyzing more applications such as database and their optimization feasibility in such dynamic resources allocation environment. The analysis will consider analytical models such as queuing analysis. We will also extend our work to be integrated with other resource management schemes like "running multiple instances" and "VM migration". Therefore, we will expand our experimental setup to more than one host in Future SOC Lab.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, Oct. 2003. ACM Press.

[2] J. Heo, X. Zhu, P. Padala, and Z. Wang. Memory Overbooking and Dynamic Control of Xen Virtual Machines in Consolidated Environments. In *Proceedings of IFIPIEEE Symposium on Integrated Management IM09 miniconference*, pages 630–637. IEEE, 2009.

[3] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. Parekh. Online Response Time Optimization of Apache Web Server. 2003.

[4] J. T. J. Midgley. Autobench, 2008.

[5] D. Mosberger and T. Jin. httperf - A Tool for Measuring Web Server Performance. In *In First Workshop on Internet Server Performance*, pages 59–67, 1998.

[6] VMWare. Vmware. http://www.vmware.com, 2010.

[7] Z. Wang, X. Zhu, S. Singhal, and H. Packard. Utilization and slo-based control for dynamic sizing of resource partitions. 2005.

[8] X. Zhu, Z. Wang, and S. Singhal. Utility-Driven Workload Management using Nested Control Design. *American Control Conference*, pages 6033–6038, 2006.

VMs Core-allocation scheduling Policy for Energy and Performance Management

Christoph Meinel, Ibrahim Takouna, and Wesam Dawoud
Hasso Plattner Institute (HPI)
University of Potsdam
Potsdam, Germany
{christoph.meinel, ibrahim.takouna, wesam.dawoud}@hpi.uni-potsdam.de

Abstract

In this phase of the project, we investigate the sensitivity of a VM performance running scientific multithreading applications to changes in clock frequency and VM performance dependency on Domain-0 for IO-intensive applications. Then, using sensitivity analysis to schedule VM to suitable core with suitable frequency settings. Currently, our work is built on a static heterogenous system, next we are going to investigate building a dynamic heterogenous system to realize power consumption proportional to a service requirements. However, our test environment showed that we can gain power savings up to 17%.

1. Project Idea

Merging between multi-core processors and virtualization technologies has prompted us to investigate the possibility of achieving power saving for such combination for scientific multithreading applications. In this project, we investigated the advantages of virtualizing heterogeneous multicore systems where they could provide better performance per watt compared to homogeneous processors [1, 10, 7]. A single processor will contain hundreds of cores that vary in some micro-architecture features such as clock frequency, cache size, power consumption, and others [3], but these cores exploit the same instruction-set architecture. A single chip might have several complex cores and many simple cores. The simple cores are characterized as low-speed clock frequency, cache size, and low power consumption while fast cores are equipped with high-performance features such as high-speed clock frequency, cache size, and high power consumption. Consequently, their potential to achieve different levels of performance that meet applications heterogeneity has prompted researchers in the operating systems domain to implement heterogeneous aware schedulers [9, 8, 2].

With heterogeneity of applications characteristics, a Hypervisors scheduler is efficient if it assigns a virtual CPU (vCPU) to run on the appropriate cores based on the application characteristics in terms of CPU-intensive, Memoryintensive, or IO-intensive. Further, it should have knowledge of the physical processors architecture and their characteristics such as cores clock frequency. By this knowledge, VMs with CPU-intensive applications should be assigned to complex fast cores to be executed faster. Generally, scientific applications are CPU-intensive, multithreaded, and fewer CPU stalls due to infrequent memory accesses or I/O operations. On the other hand, I/O-intensive could be assigned to simple slow cores without losing significant performance and achieving the power savings. However, Hypervisors scheduling-policy is based on the round-robin algorithm to ensure fairness among VMs. Emerging heterogeneous system and virtualization bring more power savings and better resources utilization. This combination needs a new scheduler, which schedules each VM to an appropriate core based on its characteristics.

We used NAS Parallel Benchmarks [5] as CPU-intensive application and netperf benchmark [6] as I/O-intensive application. We denoted performance sensitivity to CPU clock frequency as performance-frequency sensitivity and performance dependency on Domain-0 as performance- Domain-0 dependency. Our scheduling-policy based on these two categories: performance-frequency sensitivity and performance-Domain-0 dependency to assign a vCPU to the appropriate core. Consequently, the results showed good performance improvements for VMs with CPUintensive applications and for VMs with IO-intensive applications as well. Finally, our heterogeneous experimental environment achieves promising power savings reach to 17% which theoretically could reach to 45% . The power savings are gained from this architecture, which runs on two cores with high frequency and other two cores with low frequency.

2. Used Future SOC Lab Resources

The evaluation tests were performed on Fujitsu PRIMERGY RX300 S5 server. It has a processor of Intel(R) Xeon(R) CPU E5540 with 4-cores and the frequency range is 2.53GHz to 1.59GHz. the server is equipped with 12GB physical memory. Additionally, We used ServerView Remote Management to monitor power consumption for CPU.

3. Findinds: VMs Sensitivity Analysis

In this section, we analyzed sensitivity of VMs performance to changes in CPU clock frequency for VMs that run CPU-intensive and IO-intensive applications. Then, we illustrated dependency of VMs on Domian-0 for VMs with IO-intensive applications.

3.1. VMs with NBP Sensitivity Analysis

To analyze VMs performance-frequency sensitivity, we used NBP-SER and NPB-OMP benchmarks as CPUintensive programs. In this experiment, we pinned vCPUs of Domain-0 to cores (0,1) and vCPUs of VMs were pinned to the another two cores (2,3) to avoid Domain-0s influence on the VMs; in other words, to prevent Domain-0 from being queued with the VMs in the same queue. First, the experiment was run while the cores (2,3) were set to run with high frequency F_F =2.53GHz as fast cores. Then, it was run again after changing frequency settings of the cores (2,3) to low frequency F_S=1.59GHz as slow cores. Finally, we used the price elasticity of demand economics formula to determine programs completion time and throughput sensitivity of clock frequency. We considered T the completion time and Th the throughput as the demand, and F clock frequency as the price. $E_{T,F}$ is the completion time sensitivity of clock frequency, and $E_{Th,F}$ is throughput sensitivity of clock frequency.

$$E_{T,F} = \frac{T_F - T_S}{F_F - F_S} * \frac{F_F + F_S}{T_F + T_S} \quad (1)$$

$$E_{Th,F} = \frac{Th_F - Th_S}{F_F - F_S} * \frac{F_F + F_S}{Th_F + Th_S} \quad (2)$$

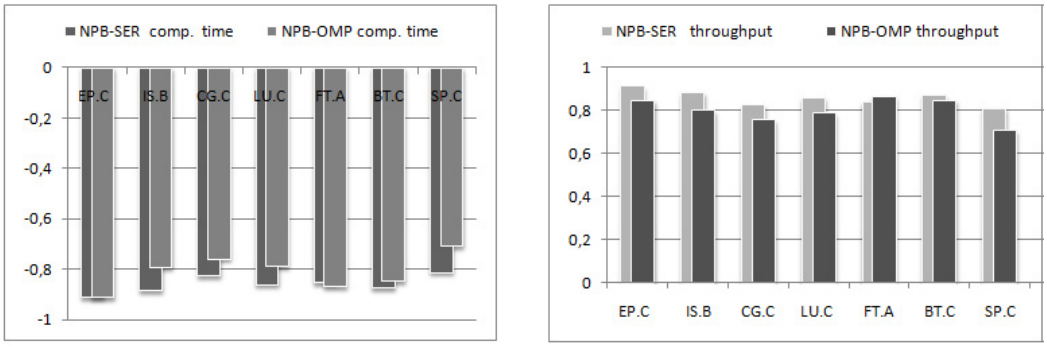
Due to the inverse relationship between CPU frequency and completion time, $E_{T,F}$ values are negative, so completion time increases as CPU frequency decreases and vice versa. On the other hand, $E_{Th,F}$ values are positive because of the direct relationship between CPU frequency and throughput. Program speedup depends on program characteristics, so it does not have a liner relationship with CPU frequency. However, CPU-intensive programs might have a semiliner relation with frequency because of either infrequent memory accesses or I/O operations. Figure 1 shows NPB-OMP and NPB-SER benchmarks performancefrequency sensitivity (i.e., completion time and throughput). NPB benchmark each program has different memory access behavior and various interprocess communication patterns. These characteristics determine sensitivity of a program to frequency changes. For example, the completion time of EPOMP and EP-SER programs had the same sensitivity and they gained the highest sensitivity. The similarity between these two programs is that EP-OMP a multithreaded program but has negligible inter-process communication and EP-SER is a single thread program without inter-process communication. Further, EP-OMP is seldom memory access compared with CG-OMP and LU-OMP. Generally, NPB-SER programs sensitivity to frequency changes was higher than NBP-OMP due to the sequential execution of instructions in NPB-SER and inter-process communication patterns or IO operations in some of NBP-OMP programs such as CG and BT respectively. On the other hand, NBPOMP programs with intensive inter-process communication were less sensitive to frequency such as CG-OMP and LUOMP. FT, a mixed type program, almost had the same sensitivity in NPB-SER and NPB-OMP. Unlike LU-OMP, BT-OMP includes a number of I/O operations that do not need synchronization among its threads.

3.2. VMs with I/O Sensitivity Analysis

We analyzed sensitivity of VMs performance with I/Ointensive to CPU frequency. Then, as I/O operations depend on Domain-0, we tested VMs performance-Domain-0 dependency.

1) CPU Frequency Sensitivity: In this experiment, we ran netperf with TCP-STREAM and UDP-STRAEM options to test I/O performance-frequency sensitivity using formula (2). The setting of this experiment was the same setting when we tested VM with NBP sensitivity. As shown in figure 2-(a), TCP test is more sensitive to core frequency than UDP due to the nature of TCP-packet; UDP does neither message fragmentation nor reassembly. Further, the aggregate costs of non-data touching overheads consume majority of the total software processing time. The nondata touching overheads come from as network buffer manipulation, protocol-specific processing, operating system functions, data structure manipulations (other than network buffers), and error checking[16]. To validate our test, we used SCP application TCP-based to transfer a 500MB file between two VMs and we found the same results obtained using netperf-TCP.

2) VMs with I/O Domain-0 Dependency: In this experiment, we ran netperf benchmark with TCP-STREAM and UDP-STRAEM options to test I/O performance-Domain- 0 dependency. For this end, we reversed the scenario of VM performance-frequency sensitivity, so the cores (2,3) settings were not changed and were



(a) (b)

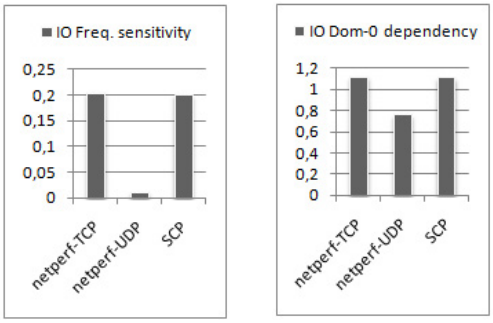
Figure 1. Performance-frequency sensitivity for NPB-OMP and NPB-SER versions run on a VM with two vCPUs. (a)sensitivity of completion time to frequency, and (b)sensitivity of throughput to frequency.

set to high frequency F_F=2.53GHz where VMs were pinned in cores (2,3). On the other hand, The cores (0,1) were set to high frequency F_F=2.53 GHz where Domain-0 was pinned. Then, we ran it again while the frequency of cores (0,1) is low F_S=1.59GHz. Finally, we computed the performance-Domain-0 dependency using formula (2). The result of this experiment is shown in figure 2(b). It illustrates that both netperf-TCP and netperf-UDP depend on Domain-0 for commutation between to VMs, but netperf-TCP depends on Domain-0 more than netperf-UDP.
The conclusion is that applications based on TCP protocol are frequency sensitive and they are Domain-0 dependant as depicted in figure 2(a) and figure 2(b) respectively.

4. Performance Evaluations

In this section, we evaluated our improved scheduling-policy with the following rules:

- The weight of VM is proportional to the number of vCPUs.
- CPU-intensive vCPU should not being queued with I/O-intensive vCPU. Further CPU-intensive vCPU should be placed in the fast pCPUs queue and I/Ointensive vCPU in the slow pCPUs queue.
- A virtual machine with CPU-intensive application and a single vCPU should be placed in fast pCPUs queue to accelerate the sequential execution.
- The time-slice for the fast pCPUs queue is 30ms and time-slice for slow cores is 10ms as show in figure 3. We chose the value 10ms for the short slice as one tick to avoid high context switching and to keep consistent credit accounting.



(a) (b)

Figure 2. Performance-frequency sensitivity and Domain-0 dependency for NPB-OMP and NPB-SER versions run on a VM with two vCPUs. (a) performance-frequency sensitivity, and (b) performance-Domain-0 dependency.

- The settings of cores in the experiments are the fast cores (0,1) with F_F =2.53GHz and the slow cores (2,3) with F_S=1.59GHz.

3) I/O and CPU-intensive Isolation: In this experiment, we created three VMs one with two vCPUs and the other two each has one vCPUs. We ran netperf on the two VMs with one vCPU for testing TCP and UDP bandwidth channels between them. The VM with two vCPUs used to run NPB-SER, then NBP-OMP. We ran the three VMs with our new scheduling-policy. First, we used EP and CG programs in NPB-SER with netperf, then EP and CG of NPB-OMP were used. We pinned the VMs with I/O to the slow cores (2,3) and the VM with CPU-intensive was pinned to the fast cores (0,1). Performance improvement for

both I/O and CPU-intensive VMs compared to the default scheduler is illustrated in figure 4. Figure 4(c) shows that the performance gain of CG.C is better than EP.C. Indeed, EP.C has negligible inter-process communication compared to CG.C which has also memory accesses. On the other hand, netperf-TCP throughput when co-hosted with VM that ran NBP-SER is better than when co-hosted with VM that ran NBP-OMP. As seen in figure 3(b), netperf depends on Domain-0 and NPB-SER is a single thread test that gave Domain-0 chance to be scheduled in fast cores and improve I/O operations for netperf-TCP. The aggregate average gain is depicted in figure 4(c). Obviously, isolating CPU-intensive vCPUs from IO-intensive vCPUs was the main reason for performance improvement. Using isolation eliminated the sources of delay that effect on CPU-intensive vCPUs performance.

4.1. VMs with sensitive Inter-process Comm.

In this experiment, we tested the performance gain for inter-process communication intensive such as CG and LU of NPB-OMP version. The performance of NPB-OMP benchmark in VM is near to the performance in physical server as long as the vCPUs are less than pCPUs, and LUOMP is the most sensitive program to communication delay [12]. For testing inter-process communication intensive program performance improvement, we created one VM with one vCPU and another VM with four vCPU. Nevertheless, we had five vCPUs in addition to four vCPUs for Domain-0. The performance gain is illustrated in figure 5 where figure 5(a) shows Throughput gain and completion time speedup for NPB-OMP while figure 5(b)illustrates Throughput gain and completion time speedup for NPB-SER. Figure 5(c) shows the average aggregated performance gain for NPB programs with two versions. Nevertheless, LU-OMP gained about 70% performance improvement. This improvement due to changing the time-slice of the slow pCPUs to 10ms which increases scheduling frequency. Increasing scheduling frequency gave chance for inter-process communication and synchronization. Further, decreasing time-slice decreases holding time when vCPU status busy blocking holds pCPU [4]. A lot of busy blocking wastes pCPU cycles and degrades the overall system performance.

5. Next Steps

The next steps of our project will be as follows:

- Analyzing energy characteristics of scientific mutlithreading applications executed on VM with multi- Virtual-CPU to provide a dynamic mechanism for saving energy while satisfying performance requirements. As we studied the NPB benchmarks performance sensitivity of changing in CPU frequency, Our method is to determine number of virtual-CPUs for a virtual machine and cores frequency settings in order to minimize energy consumption.
- We would like to use SOC LAB resource that has 64-cores which enable many configuring combinations of Frequency, Voltage, and number of cores where the used machine only has 4 cores. Using 64-cores machine also could be useful to apply our idea of cores clustering based on CPU clock frequency and other features.

References

[1] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, and P. Husbands. The Landscape of Parallel Computing Research: A View From Berkeley. Technical Report UCB/EECS-2006-183, UC Berkeley, 2006.

[2] M. Becchi and P. Crowley. Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures. In *Proceedings of the Conference on Computing Frontiers*, 2006.

[3] S. Borkar. Thousand Core Chips-A Technology Perspective. In *Proceedings of the DAC*, 2007.

[4] H. Chen, H. Jin, K. Hu, and J. Huang. Dynamic Switching-Frequency Scaling: Scheduling pinned Domains in Xen VMM. In *Proceedings of 39th International Conference on Parallel Processing*, pages 287–296, 2010.

[5] R. V. der Wijngaart. NAS Parallel Benchmarks v. 2.4. Technical Report NAS-02-007, NAS, Oct. 2002.

[6] R. Jones. NetPerf: a Network performance benchmark. http://www.netperf.org, 2011.

[7] R. Kumar, K. I. Farkas, and N. J. et al. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In *Proceedings of MICRO 36*, 2003.

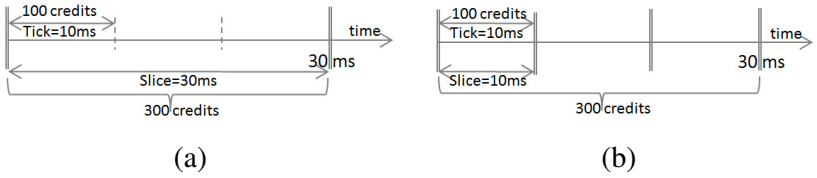
[8] R. Kumar, D. M. Tullsen, and P. R. et al. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In *Proceedings of ISCA*, 2004.

[9] R. Kumar, D. M. Tullsen, P. Ranganathan, N. Jouppi, and K. Farkas. Single-ISA Heterogeneous Multicore Architectures for Multithreaded Workload Performance. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, 2004.

[10] T. Y. Morad, U. C. Weiser, A. Kolodny, M. Valero, and E. Ayguade. Performance, Power Efficiency and Scalability of Asymmetric Cluster Chip Multiprocessors. *IEEE Computer Architecture Letters*, 5(1):4, 2006.

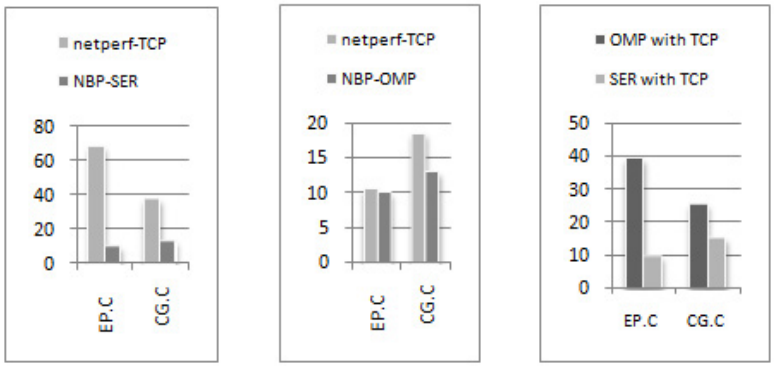
[11] D. Shelepov and A. Fedorova. Scheduling on Heterogeneous Multicore Processors Using Architectural Signatures. In *Proceedings of the Workshop on the Interaction between Operating Systems and Computer Architecture, in conjunction with the 35th International Symposium on Computer Architecture*, Beijing, China, June 2008. WIOSCA '08.

[12] C. Xu, Y. Bai, and C. Luo. Performance Evaluation of Parallel Programming in Virtual Machine Environment. In *Proceedings of Sixth IFIP International Conference on Network and Parallel Computing*, pages 140–147, 2009.



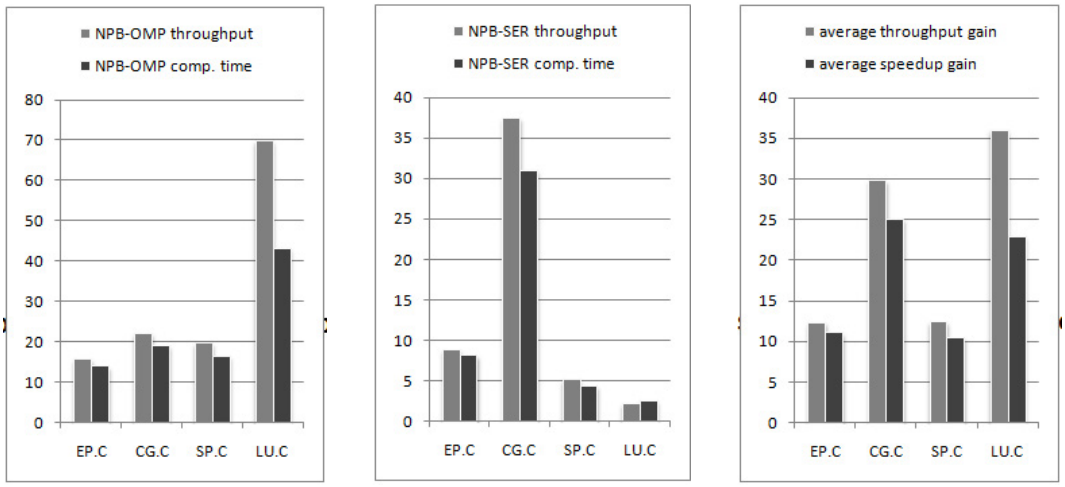
(a) (b)

Figure 3. Scheduling time-slice modifications.(a) time-slice = 30ms for fast cores, and (b) time-slice = 10ms for slow cores. The accounting period of vCPU is 30ms for both fast and slow cores.



(a) (b) (c)

Figure 4. I/O and CPU-intensive Isolation Performance improvements; netperf-TCP run on a VM with one vCPU,and NPB-OMP run on a VM with two vCPUs. (a) Throughput gain for NPB-OMP and netperf- TCP benchmark, (b) throughput gain for NPB-OMP and netperf-TCP benchmark, and (c) the average improvement of the overall system.



(a) (b) (c)

Figure 5. CPU-intensive with inter-process communication intensive performance improvements;NPB-OMP run on a VM with four vCPUs and NPB-SER run on a VM with one vCPU. (a) Throughput gain and completion time speedup for NPB-OMP,(b) throughput gain and completion time for NPB-SER, and (c) the average improvement of the overall system.

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

Sebastian Wätzoldt, Stephan Hildebrandt, Holger Giese and Axel Uhl
System Analysis and Modeling at Hasso Plattner Institute for IT Systems Engineering
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
{sebastian.waetzoldt|stephan.hildebrand|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. 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

Today, Moores law 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 for *multi-core computing* [2], which integrates multiple cores on one CPU rather than to speed up 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, todays 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 resources 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 supporting a domain-specific language (DSL) characterized as follows: (a) The higher-level abstraction of the software description enables deriving several optimized platform-specific solutions for different as well as changing hardware settings. (b) In order to ensure that the system always operates 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

*Special thanks to the students Moritz Burkhardt, Ralf Diestelkämper, Michael Kusber and Richard Meißner of the bachelor project "Model-Driven Software Development for Multicore and Cloud Systems" for the contribution and results of this project.

self-optimization of its execution strategies to ensure scalability.

To achieve this goal, we require in the long run (i) a high-level modeling approach for the software and its deployment, (ii) a suitable runtime environment to support the monitoring as well as adaptation, and (iii) alternative execution strategies as well as selection strategies for them.

After some first more general experiments with a given higher-level modeling language in the initial 6 months project (cf. [4]), we now cooperate with SAP and address the three questions from an industrial perspective supported also by a joint bachelor project "Model-Driven Software Development for Multicore and Cloud Systems". Together with the bachelor project, the focus of providing a domain specific language, which enables an easy usage for business queries and exploits the underlying parallel hardware infrastructure as good as possible is key.

At first, we want to the support *explicit parallelism* given explicitly in the specification by the user of the DSL. In addition, we also exploit *implicit parallelism* when the specified behavior is executed on large data sets or an optimization analysis finds potential parallel execution queries. Here, highly abstract specification of the data queries and data manipulation in the DSL 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 modeldriven 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 first point out the ideas using parallelism in a domain specific language in Section 2. A conclusion and an outlook on planned additional work close the report.

2. Parallel Concepts

With respect to our industry partner, the domain of our specific language is about typical business queries. We identify four perspectives for the DSL development. In the following, we briefly discuss some of the ideas which influence the DSL development.

2.1. Explicit Parallelism

At first, we want to deal with explicit parallelism. So, we take the actor concept from Scala programming language (cf. [3]). Actors are independent units which receive signals asynchronously and perform some piece of work in parallel according to the received message. The following pseudo code Listing 1 depicts the usage of actors within the DSL.

```
1  actor optimizer :
2    calculateOptimization() :
3    {
4      //heavy operation
5    }
6
7  actor monitor :
8    update() :
9    {
10     //update all data
11   }
12
13 //now use the defined actors
14 optimizer.calculateOptimization()
15 monitor.update()
16 //do some other operation
```

Listing 1. DSL Actor Example

In Line 1 and 7 a new actor is defined. If the *optimizer* receives an asynchronous *calculationOptimization* signal, it performs an according operation. After definition, the user can trigger these operations like it is shown in the lines 14 and 15. Each actor queues the signals to avoid the loss of messages and perform them according to an internal scheduling strategy. Figure 1 shows one possible execution scenario for the example in Listing 1 (ll. 14-16).

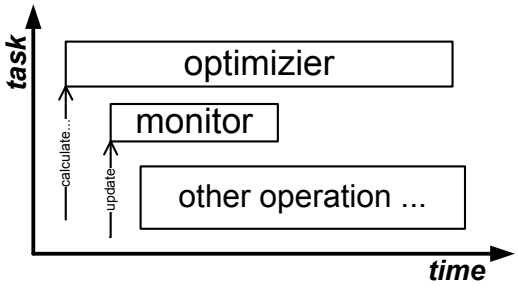


Figure 1. Actor Execution Scenario

After receiving the signal, the actor runs in parallel and stops if all messages are performed. So, the normal execution (*other operation*) is able to proceed immediately.

In addtion to the actors, we investigate a parallel block construct. Similar to a fork in programming languages, all requests and queries in the parallel block are processed in parallel. Listing 2 shows an example for a parallel block. After the key word *parallel* in Line 1, three statements are marked as concurrent for execution (ll. 3-5).

```
1 parallel :
2 {
3   a = function1()
4   b = function2()
5   c = a + b
6 }
```

Listing 2. DSL Parallel Block Example

Figure 2 depicts one possible execution plan for this example. The execution environment is able to detect

dependencies. Therefore, the statement $c = a + b$ in Line 5 is processed after the other two statements. Additionally, a warning or a hint could be given to the user of the DSL to remove that statement from the parallel block.

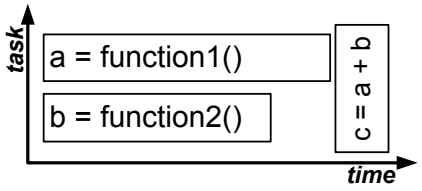


Figure 2. Parallel Block Execution

Defining explicit parallelism within the DSL should be easier for the user because of the appropriate level of abstraction. Additionally, the specialized constructs according to the domain should minimize modeling errors. These two assumptions must be verified in further research. However, defining explicit parallelism could only be a first step to reach the goal of scalability in the software and might be still hard for the user. If the user did not consider potential dependencies between queries, the execution environment can handle and avoid race conditions between data. In the worst case, the data dependencies lead to a serial execution.

2.2. Implicit Parallelism

To avoid challenging explicit parallelism modeling and to reach the goal of a scalable software, the execution environment should be able to optimize data queries and therefore using implicit parallelism without any user interaction. This optimization depends on the modeled use cases and is the second perspective of using (data) parallelism with our DSL approach.
One example for using implicit parallelism is shown in Listing 3. For some scenarios, the runtime engine is able to parallelize code blocks. In this example a for each loop could be executed in parallel. Here, no explicit user interaction is needed. The degree of parallelization depends on the available hardware infrastructure and other key performance indicators (description is following later).

```
1 table = //operation which returns a big table
2
3 for element in table
4 {
5   check(element);
6 }
```

Listing 3. DSL Implicit Parallelism

In the example, the *check* operation in Line 5 is side effect free and can be done in parallel. If enough resources are available or according to a specialized strategy the task execution could look like it is depicted in Figure 3.

At this point, offering only a few DSL constructs with a clear semantics can help the execution environment to exploit the full parallelization potential.

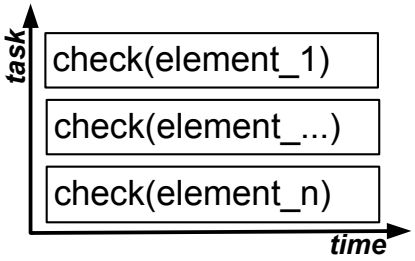


Figure 3. Using Implicit Parallelism

2.3. Heuristics and Adaptation

Additionally, the adaptation of the software behavior takes an important role and is the third perspective in our research. Here, we investigate two directions. The first one deals with static analysis and heuristics which leads to optimized execution strategies. These strategies can be computed offline and can differ for varying use case scenarios. Furthermore, this offline analysis enables the usage of implicit parallelism and detects data dependencies.
The second dimension is about self-adaptation at runtime. An analyzer component within the execution environment defines and monitors key performance indicators (KPIs). A changing workload, network connection speed or different resource limitations (like main memory or available CPU) lead to a changing software behavior. At this time we identify and investigate different KPIs for hardware resources as well as data and query constellations, which indicate a good parallelization potential.

2.4. Hardware Infrastructure

The fourth research perspective in our DSL considers the lowest layer, which is a variable hardware infrastructure and a different software landscape. For example, we run different use case scenarios on varying machines, from a standard PC with two physical cores up to a Future SOC machine with 48 cores. Due to the fact that our domain deals with typical business queries, we work with two different data bases for the query processing. The main question from this fourth research perspective is: How we can find a good mapping from the high level DSL specification to the current hardware infrastructure and the available data base type. Additionally, we offer different micro optimizations for a given execution environment which can be used in the adaptation process (static or dynamic). At this point, we must decide how the architecture of our execution model should look like (for

example splitting up the execution in an application server at the front side and a main memory data base in the back end).
For flexibility reasons, we decide to interpret the DSL. This allows us to switch between different architecture to investigate various use case scenarios and there parallel potential.

3. Conclusions and Future Work

We only recently started with the evaluation of the outlined parallelization strategies. Currently, we can only show that there is high potential for major performance improvements by means of parallelizing the execution of domain specific constructs. At this time, we are measuring different combinations of the proposed optimizations and use case scenarios. So, further statistical evaluation results are not yet available. 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 specialized DSL queries 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 speed up the execution of the queries 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. Additionally, 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, Feb. 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] P. Haller and M. Odersky. Actors that unify threads and events. In *Proceedings of the 9th international conference on Coordination models and languages, COORDINATION07*, page 171190, Berlin, Heidelberg, 2007. Springer-Verlag.

[4] S. Wätzoldt, S. Hildebrandt, A. Seibel, G. Gabrysiak, and H. Giese. Towards Scalable and Self-Optimizing Software for Multi-Core and Cloud Computing. Technical Report 42, Proceedings of the Fall 2010 Future SOC Lab Day, Universitätsverlag Potsdam, 2011.

Buildbot Project Progress Report

Martin v. Löwis, Bernhard Rabe, Peter Tröger, Alexander Schmidt
Hasso Plattner Institut, University of Potsdam
P.O.Box 90 04 60, D-14440 Potsdam, Germany
{martin.vonloewis, bernhard.rabe, peter.troeger, alexander schmidt}@hpi.uni-potsdam.de

Abstract

In this phase of the project, we have been focusing on achieving the objective of building all PyPI packages for a given Debian/Ubuntu release, performing each build in a separate virtual machine. We focused on the management aspect for virtual machines, determining how the FutureSOC infrastructure can be used appropriately for the given task.

1. Introduction

Our work for an automated build system is intended for the Python Package Index[1] (PyPI, formerly known as the Cheeseshop) a central repository for software libraries and applications. Currently, it hosts ca. 14,000 packages, maintained by ca. 5,000 registered developers. While the packages are written primarily in Python, other languages (in particular C/C++ and JavaScript) are also used. One of the authors of this paper is currently in charge of maintaining the PyPI repository server application.

In the Python ecosystem, the installation procedure for a package is typically written in Python itself. The installation steps range from copying files around on disk, over generating files on-the-fly during installation, to invoking the C compiler to build machine code for the target system. Part of the installation procedure is also to make sure that all dependencies of the package are satisfied, i.e. that appropriate versions of prerequisite packages are installed. The list of specific dependencies required by a package may itself be computed only at installation time, e.g. when a dependency is conditional on the kind of operating system or the specific version of the Python interpreter.

Linux distributions solve the installation issues with a slightly different approach: instead of installing source code, users install pre-compiled binary packages. In these binary packages, most build steps have already been performed, resulting in a package that is specific for the Linux distribution. As a consequence, it becomes possible to declare dependencies in a strictly static manner. For example, the Debian family of operating systems uses the .deb file format [1] for packaging. Debian users often prefer to install "native" Debian packages, instead of having to run custom installation procedures. Specific packages need to be built for every Debian release, as well as for every distribution derived from Debian, such as the various Ubuntu releases.

The objective of the intended build system is to provide all PyPI packages as Debian packages, enabling users to use standard installation procedures. As a consequence, the project needs to build the whole set of PyPI packages for every new Debian and Ubuntu release. In addition, each package needs to be built for all target systems whenever a new release of the package is made. The build infrastructure should be fully automated, so that Debian packages become available quickly after a new release. This may result in packages that have a lower quality than the manually-maintained Debian packages. For example, the manually-created package might provide the documentation for the package as a separate binary package, whereas the automatically created one fails to package the documentation at all, as the standard installation procedure does not cope with documentation building. In exchange, users will have to wait (often for many months) for a new release of the Debian package.

1.1 Issues with automated PyPI installations

In order to run a fully-automated installation procedure, a number of issues need to be considered. First, the Python packages often do not contain enough information to build a "correct" Debian package. The *stdeb* package copes for this lack of information, by extracting as much meta-data as feasible from the Python package, and filling the rest with dummy data.

Second, the build procedure may run untrusted code. PyPI is open for anybody to submit code; the manual review process to establish trust that the Debian

[1] `http://pypi.python.org/`

maintainers perform would not be used when creating packages automatically. Therefore, a malicious user may introduce malware in the build process. Even if not assuming malicious users, installation procedures often leave the system in a modified state that cannot be easily reverted: files may be placed in central library folders, and libraries be overwritten; services may be started; user accounts created; etc.
Third, packages may fail to properly declare their dependencies. This should result in failing tests, but will so only if the build machine doesn't have the dependencies installed. In order to detect this issue, each build should start with a minimum set of installed packages.
To resolve the second and third issue, a clean file system image is needed for every new build, on a machine that does not have sensitive data on it, and the system performing the build needs to be decoupled from the network in a way to prevent malicious code to perform an attack on the local network. The only data surviving each build should be the log files of the build, and, if the build was successful, the resulting binary package.
With the need for a file system reset comes the fourth issue of build times. Many of the build procedures complete within a few seconds, as shown in Figure 1. The time to reset the file system to a clean state should not be significantly larger than the actual build times.
More specifically, the packages released on a single day should become available in binary form on the same day (or else the build process will never catch up), and the complete rebuild necessary for a new operating system release should complete well before the next release of the operating system.

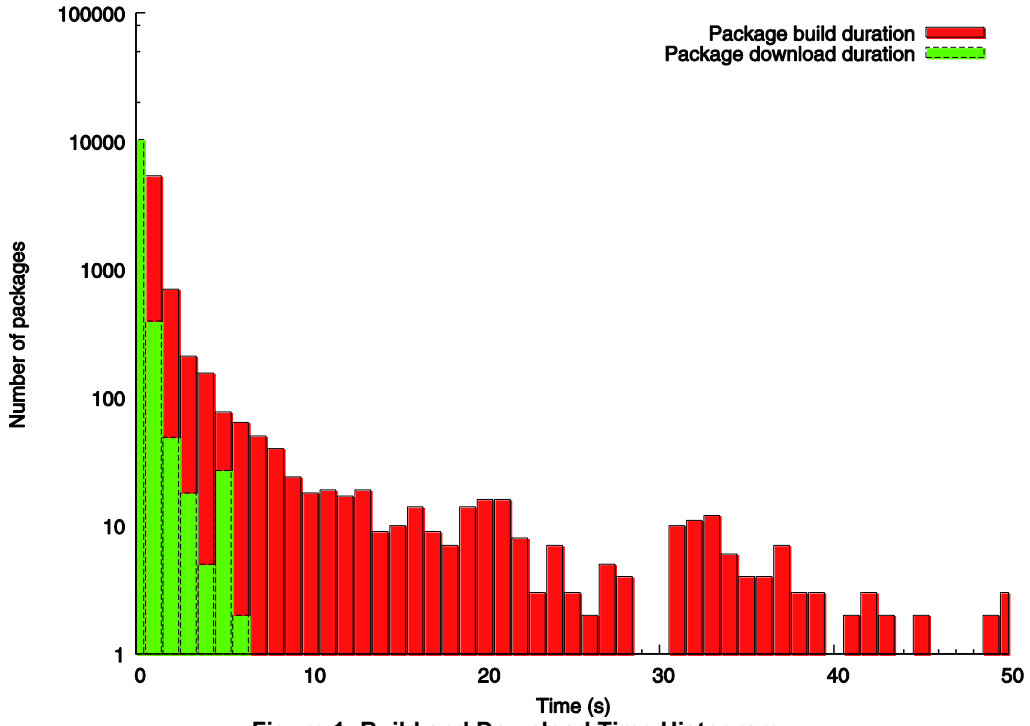


Figure 1: Build and Download Time Histogram

2. Virtualization As Solution

To implement this project, we have decided to use virtual machines as an approach. Virtual machines have been demonstrated to provide the necessary isolation for the execution of untrusted or malicious code [2].
With respect to the issues discussed above, we have considered a number of design alternatives that we would like to present.
The main challenge of the project is to start each build process from a clean, minimal system. Each declared pre-requisite package ought to be installed during the build process, followed by a download of the package to be built, followed by the actual build steps, followed by an upload of the build results.

In order to reset the file system, we identified three possible strategies:

- As Linux supports overlay mounts, a large-enough ram-disk can be laid over the boot disk. All installation and build steps will write to the ram disk, which is discarded after the build. This does not use special hypervisor functionality, but may run into semantic limitations of overlay mounts.
- Hard disk images of the guest system are often represented as files in the host system. Before booting the virtual machine, a copy of the hard disk could be made, and be restored after the build is complete. This is likely an expensive operation.
- Some hypervisor implementations support disk snapshot mechanisms, storing modifi-

cations to the base disk image in a separate file. Rolling back the disk state is as simple as deleting the delta information. The cost of this approach is in the overhead of virtually merging the base image with the delta modifications, which is necessary for each read access.

As a specific detail of our project, consider that we are using host systems with very large amounts of physical memory. Therefore, it becomes feasible to store all virtual hard disk information of the guest systems in physical RAM of the host machine. As we would also want to run multiple build processes in parallel, it would be best if the base image could be shared in a read-only manner in RAM across all guest systems. The modifications then can still go into RAM as well, subject to the limitation of the physical RAM. Our current hardware has 48 logical processors in system of 256 GiB of RAM, allowing for, e.g. 40 virtual machines running with 5 GiB of RAM each.

Another issue is the communication with the guest system: as systems start in a clean state, they will have no indication what the build process is that they need to run. We have identified two approaches:

- Upon start-up, each node contacts a service at a well-known IP address, to download configuration information, prerequisite packages, the source package, and to upload results to. As the network of the guest system needs to be isolated from the internet, setting up the necessary infrastructure may be challenging.
- Some hypervisors support mounting parts of the host file system into the guest system, often using special paravirtualized file system drivers. It would be possible to provide the guest system with a read-only volume containing all the files it may need (e.g. providing a complete mirror of the respective Debian release for use in `/etc/apt/sources.list`), as well as a separate volume where output files are stored (initially empty).

The guest system gets pre-configured (in its clean state) to perform a build script during start-up which actually triggers the build process. When the build process is complete, the build output gets uploaded, and the machine shuts down. As a further optimization, we are considering to have the clean image already represented a booted machine state (provided sufficient support in the hypervisor).

We are currently using Oracle VirtualBox[2] as the hypervisor technology, as it supports delta storage for disk snapshots, as well as memory snapshots, and is scriptable using a command line interface and a Python API.

3. Results

In order to evaluate the chosen approach, we have performed initial measurements of the execution time for the build step of setting up a virtual machine, booting it, performing an empty build action, and then shutting down and deleting it. Using the rx600s5-256g machine, with Oracle VirtualBox 4.0.8r71778, a complete cycle takes about 17s to complete. In this scenario, we have been using a minimal installation of Ubuntu Lucid in 32-bit mode, installed on an immutable disk (i.e. all changes are made to storage that is discarded at the end of the run).

In order to further reduce the execution time, we have placed the entire machine state into a RAM disk. In this setup, the time was reduced to about 15s, which is a smaller reduction than we expected. As a side note, development was done on an Apple MacBook laptop computer, where the build step took only about 11s. As an explanation for the relatively larger runtime on the FutureSOC system, consider that this system has 1.87 GHz processors, whereas the laptop has 2.53 GHz CPUs.

4. Outlook

The next phase of the project will integrate the pieces achieved so far, that is attempt to perform full builds of all packages on PyPI, preferably for all current Ubuntu and Debian releases. Assuming that no limitations arise from this approach, we would then like to test this infrastructure for a longer period of time.

References

[1] I. Jackson and C. Schwarz: *Debian Policy Manual v3.9.2.0,* 4 2011.

[2] G. Somani and S. Chaudhary: Application Performance Isolation in Virtualization, In *2009 IEEE International Conference on Cloud Computing,* pages 41–48, Washington, DC, USA, 2009. IEEE Computer Society.

[2] `http://www.virtualbox.org/`

Downtime Analysis for Pro-Active Virtual Machine Migration Report for the HPI Future SOC Lab

Felix Salfner
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany

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

Abstract

Live migration of virtual machines is a technique to move virtual machines from one physical host to another during runtime. The HPI Future SOC Lab project "Towards an Architectural Pattern for Pro-Active Virtual Machine Migration" investigates live migration as a mean to handle imminent faults even before they resulted in a failure. However, this approach can only be used successfully if live migration meets certain requirements regarding the duration of the migration process. In this report, we present experimental results investigating the factors determining duration of live migration.

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 tomorrows average server environments.
One commonly agreed problem resulting from smaller structural sizes, extreme memory increase (as in the Future SOC 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.

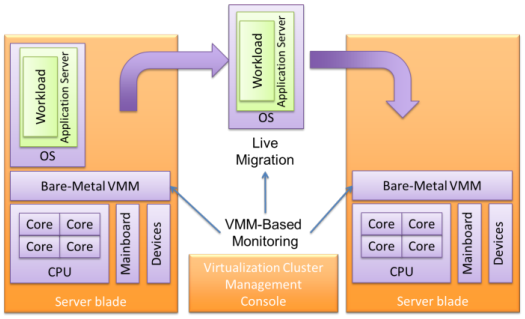


Figure 1. Reactive Live Migration of Virtual Machines

2. Approach

Live migration is a technique to move running virtual machines from one physical host to another without disrupting running applications or the virtualized operating system. So far, live migration is primarily used for load balancing, server consolidation, for planned maintenance and for recovery from a failure that has occurred either in the physical host or the hypervisor. However, reacting to failures that have already occurred leaves only 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. Our Future SOC Lab project investigates an approach where live migration is used in a proactive way to increase system dependability. Proactive in this context means to act on the first symptoms before a problem has actually evolved into a severe problem (a failure). More specifically, we seek to preventively move running virtual machines away from failure-prone hosts based on an underlying failure prediction. More specifically, 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 systems 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.

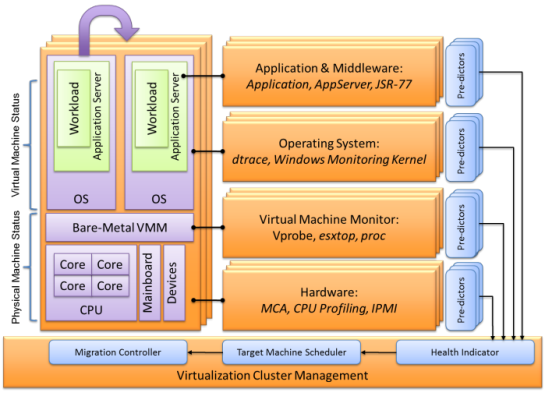


Figure 2. Pro-Active Virtual Machine Migration through Failure Prediction

Until today, such an approach would have had a serious performance impact due to the monitoring and prediction computation overhead. However, the identified problem domain 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 semiautomated 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.
As the technique of live migration is already available in todays virtualization products, in this report we focus on properties of live migration rather than its technical implementation. More specifically we investigated the factors that impact both the total duration of live migration as well as the downtime involved when switching from the source virtual machine to the destination. As we will show the relationships are non-trivial and can affect migration times significantly.
It should also be noted that our investigations focus on bare-metal virtualization only, since todays most capable live migration implementations are based on this model.

3. Experiment Setup

The goal of our experiments is to determine both total migration time as well as the downtime involved when migrating a virtual machine from one physical host to another under various load conditions. Load in this context refers to the software running in the virtual machine. Load in this context comprises CPU utilization, memory allocation as well as read and write access to the memory. In order to be able to control various aspects of load independently, we used a load generator running in the virtual machine (see Figure 3).

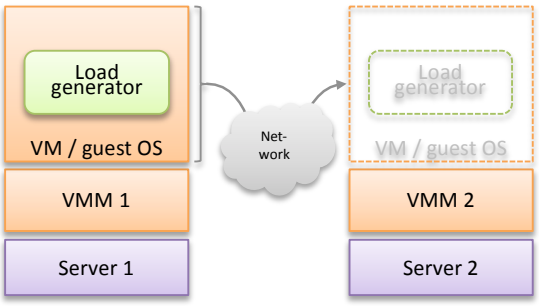
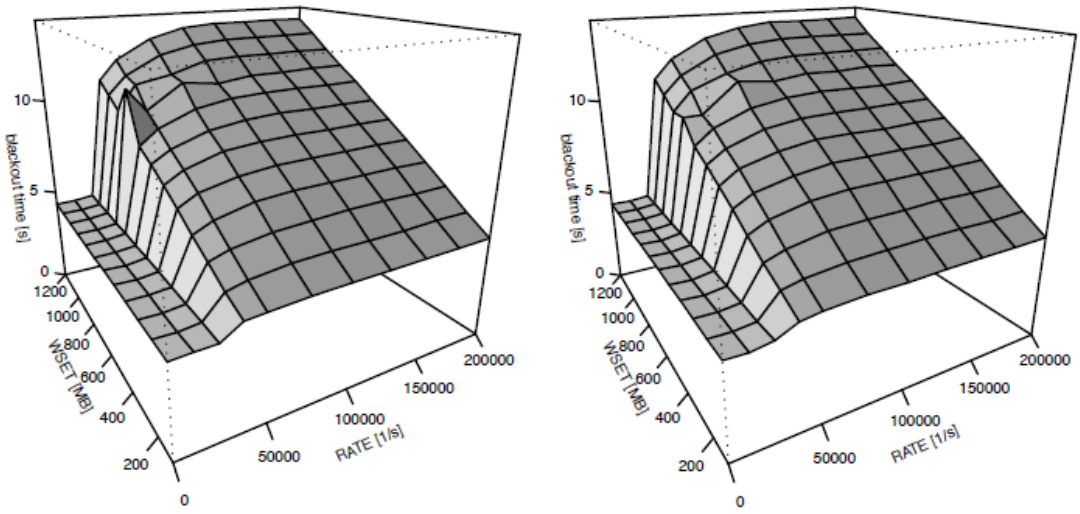


Figure 3. Experiment setup

More precisely, using the load generator we controled the following parameters:

- CPU utilization
- The amount of memory allocated (working set)
- The pattern how memory pages were written
- The frequency at which memory pages were written

In addition to the parameters controlled by the load generator we varied the hypervisor, i.e., we performed experiments with XenServer with a CentOS as guest operating system, with VMware with Linux as guest



(a) VMSIZE=4096 (b) VMSIZE=8192

Figure 4. Mean downtime for Xen plotted over RATE and WSET

operating system and with ProxmoxVE with Linux as guest operating system. Each combination was investigated with four sizes of virtual memory configured: 2GB,4GB, 6GB, and 8GB.

Parameter	Description
Hypervisor	Hypervisor product and guestOS used
VMSIZE	Amount of main memory statically configured for the VM
LOAD CPU	utilization of the virtualized operating system in percent
WSET	Working set, the sum of utilized memory
PERIOD	The period for one memory modification cycle in microseconds
BPC	Blocks per cycle. Number of modified blocks per cycle
FILL	Filling degree. The average percentage of a block being actively modified

Table 1. Parameters used in experiments

Investigating all combinations of parameters (called factors) listed in Table 1 at all their levels would result in a prohibitively large number of experiments. However, from the experiments shown in the previous report we were able to conclude that the factors LOAD and FILL can be omitted from further analysis.
A second reduction in the number of factors can be achieved by leveraging on the fact that BPC (blocks per cycle) and PERIOD (duration of one cycle) can be combined into one factor

$$RATE = \frac{BPC}{PERIOD}$$

which denotes the number of blocks that is modified per millisecond.
We have hence reduced the number of factors to the following three parameters: VMSIZE, WSET, and RATE.
We performed experiments according to a full factorial design, meaning that all possible combinations of parameter levels have been measured in the experiment. More specifically, for Xen we investigated a total number of 528 combinations (treatments), each with 20 measurements resulting in an overall number of 10560 migrations. In each experiment we measured migration time and downtime as response variables. In case of the VMware hypervisor, we performed experiments for 352 combinations resulting in 7040 migrations and for ProxmoxVE for 384 combinations resulting in 7680 migrations.
In the following we will discuss results for each virtualization framework separately.

3.1. Analysis of XenServer

As we have three factors (plus downtime/ migration time response variables) we cannot present the entire results in one plot. Since VMSIZE has significantly less levels, we decided to plot the mean response, i.e. mean migration time or downtime, over WSET and RATE for a fixed value of VMSIZE. Comparing Figure 4 to Figure 5, we can see that downtime shows a very different behavior in comparison to migration time, although the first is part of the latter.
Downtime (Figure 4) in general increases with increasing WSET and increasing RATE. This is not surprising as an increased usage of memory (more pages written at an increasing rate) requires more memory to be transferred in the stop-andcopy phase. We can also conclude from the figure that WSET

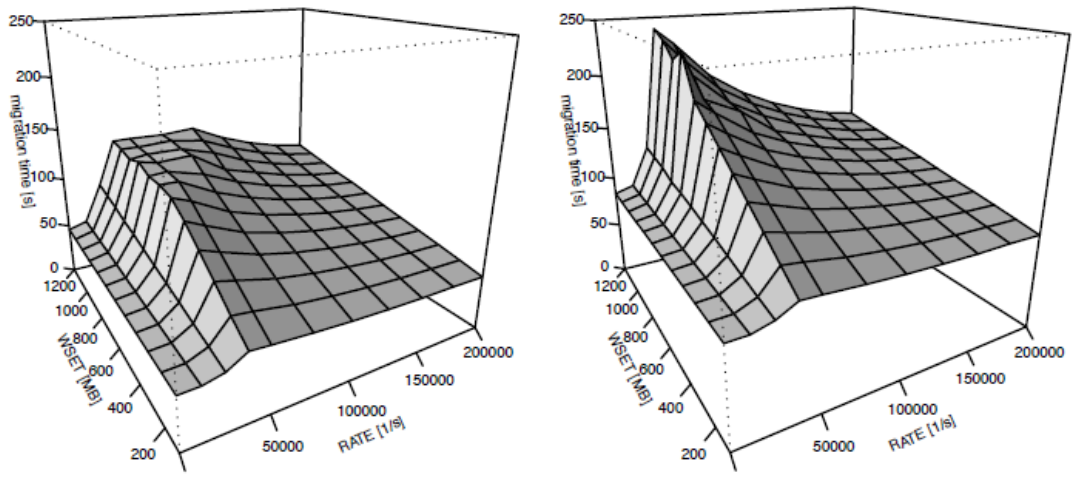


Figure 5. Mean migration time plotted for Xen plotted over RATE and WSET

seems to have a linear effect on downtime, regardless of the values of VMSIZE and RATE.

One peculiarity in Figure 4 is the abrupt change at a RATE level around $30,000\frac{1}{s}$. In order to analyze this further, we conducted additional experiments that investigated a sub-range of values for RATE at greater level of detail. Results showed that the change is not as abrupt as might have been concluded from Figure 4.

Turning to total migration time (Figure 5) we also observe a sudden change at the same level of RATE as we have observed for downtime. Again, an additional analysis showed that the change is smooth although rather steep. However, in general the mean migration time is more irregular. It came as a little surprise to us that for RATE levels "above the jump" total migration time decreases with increasing RATE. In order to check that this behavior really occurs we have carried out separate experiments specifically targeted to this question with the same consistent result. Although we cannot give a precise explanation, yet, we presume that it is caused by the rate-adaptive algorithm employed by the hypervisor. This supports our assumption that a load model is essential in order to assess duration of live VM migration.

The effect of VMSIZE can be observed by comparing the two subfigures (a) and (b) in Figure 5. It can be seen that VMSIZE has a non-trivial effect on migration time: since the shapes look very different at different levels of VMSIZE, the effect does not appear to be linear, except for the case where RATE equals zero.

There is no effect of WSET if RATE is zero, which is consistent with the single variable experiments described in the previous report.

The plots in Figure 4 and Figure 5 show migration times averaged over all 20 measurements. In order to assess the variability in the data, we report the ratio of maximum to minimum values as well as standard deviations for the data in Table 2. Specifically, two ratios and two standard deviations are reported: the ratio of the maximum treatment mean to the minimum treatment mean and the ratio of the maximum to the minimum values across all measurements. Regarding standard deviations we have reported the largest standard deviation computed within each treatment (parameter combination) as well as the standard deviation for the overall data set. In addition, the table reports the mean time averaged across all measurements.

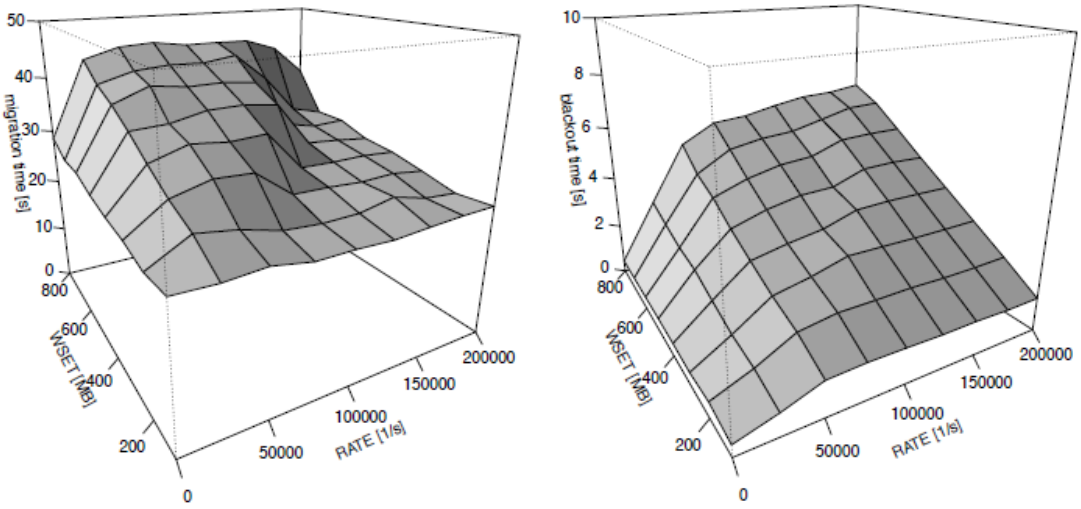
The table quantifies what has also been observable from the plots: Both migration time as well as downtime vary tremendously depending on VMSIZE and the memory load.

3.2. Analysis of VMware

Due to space limitations we report results for VMWare only for VMSIZE equal to 4GB (see Figure 6). This is no severe limitation as the behavior is very similar for other values of VMSIZE.

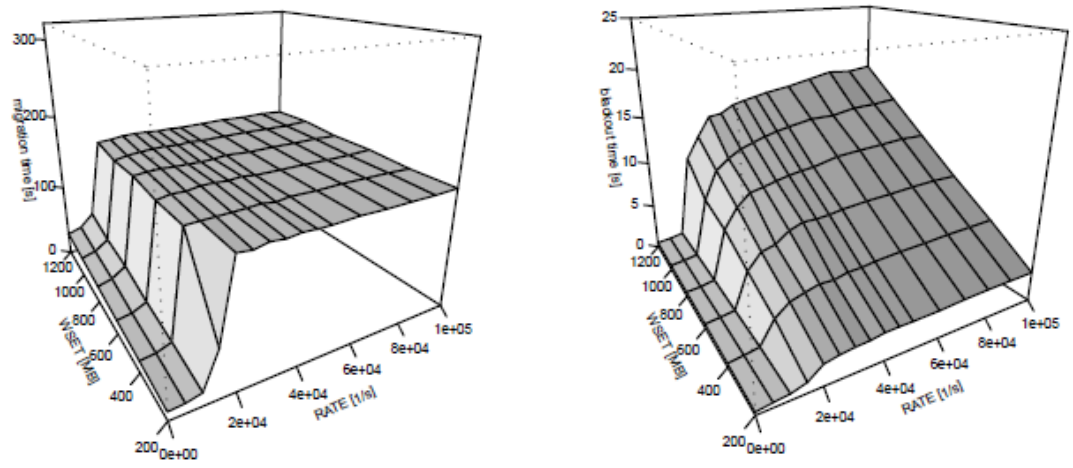
As can easily be observed the behavior differs significantly from the one of Xen, which emphasizes that the choice of the hypervisor product can have significant impact on availability. The main reason for the different behavior seems to be the different rate-adaptive algorithms employed in the two virtualization products.

Variability of the data for VMware is also listed in Table 2. Regarding the max to min ratio of downtime computed from treatment means we have observed a ratio of 16.27. This shows that due to different memory load the maximum mean downtime can be 16.27 times as large as the minimum mean downtime. If we do not consider mean downtimes but the maximum and minimum value observed across all experiments, the factor even goes up to 23.83!



a) Migration time b) Downtime

Figure 6. Mean times for VMware plotted over RATE and WSET for VMSIZE=4096



a) Migration time b) Downtime

Figure 7. Mean migration and downtime for ProxmoxVE for VMSIZE=4GB

3.3. Analysis of ProxmoxVE

Similar to VMware we only report results for VMSIZE equal to 4GB (see Figure 7).
Again the behavior is significantly different from the Xen and VMware. The most prominent difference is that for RATE greater than $20,000\frac{1}{s}$ total migration time is independent of both RATE and WSET. However, migration time is strongly dependent on VMSIZE as it grows with increasing VMSIZE. More precisely, the plateau is at about 100s for VMSIZE=2GB, about 220s for VMSIZE=6GB and about 300s for VMSIZE=8GB. Downtime does not show such dependence on VMSIZE. This behavior could be explained that ProxmoxVE copies the entire virtual machine regardless of the size of the working set (WSET). Results on data variability are reported in Table 2. As can be seen from the table, overall migration performance is significantly worse than for the other two virtualization products. This holds both for absolute values (column Mean time) as well as for the variability in the data. Regarding the max to min ratio of downtime computed from treatment means we have observed a ratio of 74.07, indicating that due to memory load the maximum mean downtime can be 74.07 times as large as the minimum mean downtime. Considering all values (without mean operator) the value goes up to 98.00, which is almost two orders of magnitude! The conclusion from this observation is that if service downtime is critical for meeting reliability goals, a realistic assessment of reliability can only be achieved if the maximum downtime for the application-specific memory load is used.

Standard deviations are also significantly higher for ProxmoxVE in comparison to the other two hypervisors. An analysis has shown that this is due to measurements in the region of abrupt change, where, for example, migration time can vary from 37s up to 292s for the same memory load (see Figure 8). It can also be observed from the plots that the abrupt change is much steeper than it was the case for the other hypervisors. In fact, in this area the data is not normally distributed and providing a standard deviation does not make sense, here.

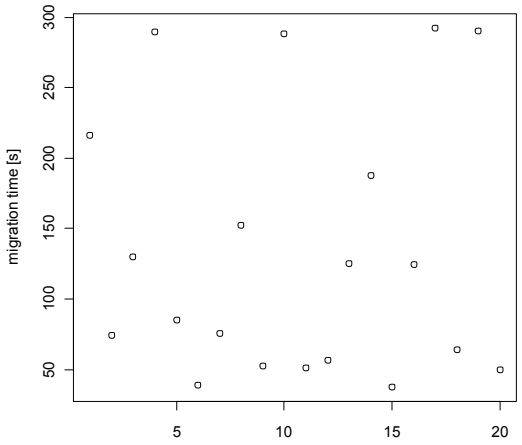


Figure 8. Migration times for ProxmoxVE for VMSIZE = 8GB, WSET = 400MB, RATE=15000/s

4. Conclusions and Next Steps

Two trends can be observed in large scale computing: an increasing number of cores and amount of memory, which leads to increased failure rates. gives rise to additional and frequently unused computing power. A second trend is the widespread use of virtualization technology. It is the goal of this project to use virtualization to better cope with increasing failure rates. In contrast to existing approaches we want to use pro-active live migration of virtual machines to move virtual machines away from failure-prone hosts even before a failure occurs. In order to have the ability to do so, upcoming failures have to be anticipated. We propose a failure prediction approach that operates on various levels of the system architecture. However even the most accurate failure prediction is useless if the action to be taken upon predicting an upcoming failure takes longer to execute in comparison to the time left until failure occurrence. Since we plan to use live migration as such an action we need to know how long it can take to migrate a virtual machine to a fault-free host. The experiments described in this report have identified and investigated decisive factors determining the duration and downtime involved when a virtual machine is migrated.
Our measurements are based on three representative virtualization products, namely VMware ESX 4.0.0, Citrix XenServer 5.6 and ProxmoxVE1.7 (KVM version 2.6.32-4-pve). Our analysis shows that performance of live migration varies heavily. Based on these measurements, we are able to support the hypothesis that memory access patterns of the guest system are the determining factor for live migration performance.
Although the use of artificial load generators was helpful to identify the factors determining the duration of live migration, it remains yet to show to what extent our results apply to industry-relevant workloads. Next steps will hence include to map workloads obtained from benchmark applications such as the ones from the SPEC suite to our measurements results. A second field of research will focus on failure prediction algorithms on data obtained from the hypervisor level.

Hypervisor / Guest	Time	Mean time [s]	Max:Min Ratio		Standard deviation	
			Mean	Overall	Treatment Max [s]	Overall [s]
XenServer / CentOS	Migration	89.73	9.01	9.10	6.32	39.08
	Downtime	7.69	3.17	3.46	0.62	2.94
VMware / Linux	Migration	30.93	2.24	2.96	7.72	7.51
	Downtime	3.10	16.27	23.83	0.50	1.80
ProxmoxVE / Linux	Migration	156.69	36.42	42.73	93.80	94.65
	Downtime	7.25	74.07	98.00	4.41	5.31

Table 2. Data variability

References

[1] A. Polze, P. Tröger, and F. Salfner. Timely Virtual Machine Migration for Pro-Active Fault Tolerance. In *2nd International Workshop on Object/component/service-oriented Realtime Networked Ultra-dependable Systems (WORNUS), at 14th International Symposium on Object/Component/Service-oriented Realtime Distributed Computing (ISORC)*, 2011.

[2] F. Salfner, P. Tröger, and A. Polze. Downtime Analysis of Virtual Machine Live Migration. In *The Fourth International Conference on Dependability (DEPEND)*, to appear.

[3] P. Tröger, A. Polze, and F. Salfner. On the Applicability of Virtual Machine Migration for Proactive Failover. In *SDPS International Conference, Special Track on Virtualization.*, to appear.

5. Teaching Activities

Project seminar “Zuverlssigkeit und Virtualisierung”, Hasso-Plattner-Institut, summer term 2011

Forward Business Recommendations – Realtime Management Support based on In-Memory Technology

Prof. Dr. Rainer Thome
Dipl.-Kff. Patricia Kraft
Chair in Business administration
and business computing
Joseph-Stangl Platz 2
97070 Wuerzburg
{thome|pkraft}@wiinf.uni-wuerzburg.de

Dr. Andreas Hufgard
Dipl.-Kff. Stefanie Krüger
IBIS Labs
Mergentheimer Str. 76a
97072 Wuerzburg
Hufgard@ibis-thome.de
skrueger@wiinf.uni-wuerzburg.de

Abstract

Rule based Business Matrix Processing is an innovative 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 [1].

Proving this idea with In-Memory Technology a prototype in Forward Business Recommendation is born. Given the complete information included in different business areas, and the cockpit issuing a practical statement managers can come to much better decisions.
Forward Business Recommendations (FBR) cockpit calls for relevant actual data and is checking **rules**. Based on these rules a **technical structure** can be established. Explaining the idea of **checked processes** the concept of Forward Business Recommendations begins to show. Finally the prototype offers **various possibilities** to support the process of reaching a decision.

1. Rules and technical structure

To use the cockpit for Forward Business Recommendations properly it is necessary to understand how data sources are combined. The prototype is based on the data source of SAP Business ByDesign (ByD) FP 2.6 using In-Memory Technology.
Valuable data is exported via Excel Add-in to create a widespread but very specific data source. On one hand information about opportunities, liquidity, orders, employees, pricing and available stock are accumulated. On the other hand the data is highly specified to keep the quantity of data exchange as small as possible.

To make sure information mirrors the processes employed, operational data has to be kept up to date. The prototype extracts only information recorded in SAP ByD.
On one hand, there is the linking of information from each employees part of the process to other sections in the chain, on the other hand, there is the overview and analytical solving of decision-making issues and organizational difficulties at **managerial level**. Management tasks have 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 procedures. 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 score-cards). The action to be taken may not be clear, but the constellation and the need for information is [1].
To place management decisions on a solid informational foundation only valid data is used. For example opportunities have to achieve a probability higher than 80 % to be factored into the calculation. To ensure proper results timing is summarized in weeks. Trying to plan future activities like stock receipt, receipt of a payment or ordering on a daily basis is as effective as reading tea leaves. Instead of prophesying we are forecasting future activities by using valid data.

2. Checked processes

Every process step has to be available to facilitate strategic planning information. Captured in this information overload, decisions are made without taking account of their influence on other divisions. Receiving orders of great value is a success for sales department. Low liquidity in order to high purchase orders is a catastrophe in financial department. Time delay in delivery upsets customers.

Figure 1. End to End process with implications on management decision

Top 10 offene Opportunities

Name der Opportunities	ID	Realisierungsfähigkeit: Realisier-barkeit	Realisierungsfähigkeit: Empfehlung	Finanzielle Auswirkungen: gewichteter Wert	Finanzielle Auswirkungen: Deckungs-beitrag	Finanzielle Auswirkungen: Bestell-kosten	Finanzielle Auswirkungen: Liquiditäts-wirkung	Logistische Planu: KW	Logistische Planu: Lagerbestan-deckung
Großauftrag für Kooperationspartner 123 Heizungen	255	25%	Wahrscheinlich nicht kombinierbar mit anderen Opportunities.	510.000 €	48%	265.788 €	-126.074 €	22	0%
Interesse an Solaranalagen	257	90%	Opportunity mit Priorität 1 weiterverfolgen	142.500 €	81%	12.250 €	123.105 €	26	80%
Interessiert an Holzofen	196	58%	Umgehend Beschaffung von Produkten mit hoher Wiederbeschaffungszeit einleiten (Handlungsspielräume erweitern).	123.500 €	40%	62.240 €	119.661 €	20	16%
Heizkessel	265	70%	Opportunity problemlos durchführbar. Mit niedriger Priorität weiterverfolgen.	96.000 €	47%	9.415 €	130.299 €	22	40%
Interesse an Solaranlagen für mehrere Gebäude	263	50%	Umgehend Beschaffung von Produkten mit hoher Wiederbeschaffungszeit einleiten (Handlungsspielräume erweitern).	90.000 €	33%	60.000 €	121.901 €	20	0%
Neugestaltung Leighton Areal	268	75%	Opportunity problemlos durchführbar. Mit niedriger Priorität weiterverfolgen.	76.500 €	50%	30.967 €	95.138 €	23	50%
Interesse an Kesseln	252	70%	Opportunity problemlos durchführbar. Mit niedriger Priorität weiterverfolgen.	38.400 €	43%	10.333 €	129.381 €	22	40%
Interessiert an Briketts	195	64%	Umgehend Beschaffung von Produkten mit hoher Wiederbeschaffungszeit einleiten (Handlungsspielräume erweitern).	36.000 €	33%	17.280 €	164.621 €	20	29%
Energiecheck und Boiler	234	59%	Opportunity problemlos durchführbar. Mit niedriger Priorität weiterverfolgen.	32.960 €	40%	15.750 €	106.355 €	24	18%
Energiecheck für Schiller	273	100%	Schnellstmöglicher Verkauf, Rabatt als Anreizinstrument prüfen.	28.000 €	100%	0 €	3.355 €	28	100%

Figure 2. Forward Business Recommendations Cockpit

To decrease problems occurring because of the lack of information, the FBR cockpit uncovers imponderabilities as long as they can be avoided. In this way it is possible to forecast relevant results caused by the current tide of events.

2.1. Sales

Information gathered in sales department contains consolidated knowledge about customers, market pattern and expectation value of opportunities. However it is difficult to measure the effects before a customer order is placed.
In SAP ByD opportunities are recorded to extend preliminary lead time. FBR prototype is based upon opportunities with a probability higher than 80 %. As shown in Figure 2 the top ten opportunities are part of the cockpit summary. Opportunities of small value have no bearing on this general view because they rarely provoke extraordinary business situations.

2.2. Warehouse

The stocks available for production are a sensitive bottleneck. A supply shortfall interferes production process or even delays sales orders. Even if products are stored reserved stock and safety stock have to be observed.
FBR cockpit allows to oversee the stock level coverage of a specific week in case of realizing the opportunity.

2.3. Purchasing

It is also advantageous that ordering costs for all products that are out of stock are forecasted with their material costs recorded in ByD. That implies supplier contracts and trade discounts.
In addition the purchasing period can be stretched in case of long replenishment time. To reduce purchasing costs greater product bundles are possible in anticipation of potential demand.

2.4. Financials

High expenses on preliminary products and raw materials endanger the liquidity of midsized companies. It is absolutely essential for concerned enterprises to oversee situations that cause financial difficulties. The FBR cockpit provides an opportunity to predict the financial effects of future orders. As a result it is possible to check whether ordering costs scratch a credit limit or whether they are impossible to be combined with other orders at the same time.

2.5. Further Constraints

An effective management ratio is the amount of workload per employee. If some employees are permanently overwhelmed with work, they are less effective. Monitoring the total number of opportunities per employee (Figure 3) can support balancing activities. For example green flagged employees can support red flagged co-workers in some cases.
FBR prototype offers a very supportive bundle of information within its basic version. Of course there are further constraints to be considered and occupied. For example production parameters were excluded in our first version to reduce complexity. Furthermore products that have not yet been procured or are infrequently purchased are not included.

3. Various possibilities

From a business viewpoint, a sales order may be influenced by several things. Earning a positive profit margin may be a priority. And theres 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 flexi-ble and cost effective alternatives can be added to the logistics chain [1].
FBR prototype accumulates information from different parts of the enterprise and recommends a specific proceeding that improves the situation of the company. For example if sales orders for an interesting opportunity scratch the credit limit it informs the manager to apply for a short-term credit. That simpli-fies project process and prevents a delay in delivery to customer. Additionally it wont interfere with the good relations to the suppliers. Another example for a recommendation placed by FBR is products in stock, stimulation per discount possible. If all products included in the opportunity are in stock, a fast selling process reduces capital commitment.

4. Next steps

Version two of Forward Business Recommendations cockpit will be built within ByD. Using the possibilities of integrated software development an Add-On is able to use far more system information than exported data could. Although real-time actualization of exported data offers great possibilities integration is expected to be an even better choice. Next generation Recommendations can be employee specific instead of manager specific.

Figure 3. Workload per employee based on open opportunities

Another improvement beyond embedding further business logic into ByD will be the realization of a real-time engine.

References

[1] R. Thome, A. Hufgard, and S. Krüger. Rule based Business Matrix processing (RBM) Business - Business Rules for real-time Process Management based on In-Memory Technology. In *Proceedings of the Fall 2010 Future SOC Lab Day*, number 42 in Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik der Universität Potsdam. Universitätsverlag Potsdam, 2011.

Accurate Mutlicore Processor Power Models for Power-Aware Resource Management

Christoph Meinel, Ibrahim Takouna, and Wesam Dawoud
Hasso Plattner Institute (HPI)
University of Potsdam
Potsdam, Germany
{christoph.meinel, ibrahim.takouna, wesam.dawoud}@hpi.uni-potsdam.de

Abstract

Power management is one of the biggest challenges in current datacenters. As processors consume the dominant amount of power in computer systems, power management of multicore processors is extremely significant. Efficient power models that accurately predict the power consumption of a processor are required to develop efficient power management techniques. However, this challenge rises with using virtualization and increasing number of cores in the processors.

In this project phase, we analyze power consumption of a multicore processor; we develop three statistical CPU-Power models based on number of active cores and average running frequency using a multiple liner regression. Our models were built upon a virtualized server. The models are validated statistically and experimentally. Statistically, our models cover 97% of system variations. Furthermore, we test our models with different workloads and different benchmarks. The results show that our models achieve better performance compared to the recently proposed model for power management in virtualized environments. Our models provide highly accurate predictions for un-sampled combinations of frequency and cores; 95% of the predicted values have less than 7% error. Thus, we can integrate these models into power management mechanisms for a dynamic configuration of a virtual machine in terms of number of its virtual-CPUs and the frequency of physical cores to achieve both performance and power constrains.

1. Introduction and Project Idea

Datacenters power consumption has become a significant concern with the rapid emergence of cloud services such as Amazon EC2. For example, Hamilton[8] has reported that Amazons datacenters are facing a highly increased power demand where the servers consume 59% of the total power supply. Furthermore, the U.S. Environmental Protection Agency (EPA) reported that the energy consumption of the datacenters located on U.S. consumed 61 billion kilowatthours in 2006 which costs $4.5 billion [14]. Thus, there have been many proposed approaches for datacenters power management [10, 12].

Current datacenters consist of a number of servers leveraging multicore processors. The number of cores in a single processor could be doubled every 18 months to maintain Moores law. The processor is the component that consumes the most dynamic power of a computer system [2, 14]. Nevertheless, an ideal sever consumes over 50% of its peak power [4] which means that a server with low utilization is very power-inefficient. Hence, virtualization technology has been rapidly employed in datacenters to increase servers utilization by enabling applications consolidation onto a fewer number of physical servers and turning off unused servers to save power.

There are several proposed approaches for power management. Mostly, these approaches consider the CPU frequency and CPU utilization to build power models. For instance, Urgaonkar et al. [13] and Gandhi et al. [7] have adopted non-liner quadric models of power consumption for power management in virtualized environments. The relationship between power consumption of multicore processor and frequency cannot be covered with one fitting curve, as we will see in Section 2. For instance, the curve of power consumption of 1 core slightly increases with cores frequency compared to 4 cores and 8 cores. Moreover, Fan et al. [6] have included CPU utilization in their proposed power model. However, using utilization to build a power model for a multicore processor could be not accurate, because the power consumed by a multicore processor with one active core with 100% utilization is more than the power consumed by two active cores each of them 50% utilized for the same workload. We found this result by conducting an experiment using a virtual machine (VM) with a mutlithreaded application. This VM ran with 1 virtual CPU and had 100% utilization and only ran on one physical core. In this

scenario, the power consumed by the physical CPU was 26 watts. On the other hand, when the same VM ran with 2 virtual CPU and the VM had the same total CPU utilization 100%. In this scenario, the physical CPU just consumed 17 watts, and each core was 50% utilized. Importantly, both of the configuration gave the same performance. Indeed, the latter could be better due to exploiting the multithreading. Thus, we conclude that only using CPU frequency and CPU utilization only as inputs for power modeling could be inefficient in particular for power estimation of mutlicore processors.

Evolving virtualized environments enables consolidation of multithreaded web and High Performance Computing (HPC) applications; these applications could efficiently utilize multicore processors. However, to implement power-aware resource management techniques for such environments accurate power estimation models are required. Hence, the purpose of this work is to build CPU-Power consumption models that accurately estimate the power consumption of virtualized servers with multicore processor. These models could be employed into power-aware resource management to achieve better power savings. Our work is distinct from others as follows. This project phase presents CPU-Power consumption models taking into account number of the actual active cores N and average running clock frequency F at each sample. It analyzes and evaluates the performance of our proposed models statistically and experimentally. The statistical analysis using the regression R^2 indicates that our models could cover more than 97% of system variations. Experimentally, our proposed models achieve better performance compared to the model adopted by [13, 7]. We evaluate models using three different applications with different characteristics (i.e., CPUintensive, Memory-intensive, and IO-intensive). The results show that 95% of the predicted values have less than 7% error. Furthermore, the maximum prediction error is less than 4% error for Memory-intensive and IO-intensive applications. As future work, we will use these models to build a dynamic optimizer that optimizes number of cores and their frequency settings and dynamically configures a VM to cope with workload and meet power consumption constrains.

2. Used Lab Resources and Experimental Setup

The evaluation experiments were performed on Fujitsu PRIMERGY RX300 S5 server that has a CPU-Power measurement capability. It has a processor of Intel(R) Xeon(R) CPU E5540 with 4-cores. The frequency range is 2.53GHz to 1.59GHz. Each core enables 2-logical cores. The server is equipped with 12GB physical memory. The experiments were run on a virtualized server using Xen-4.1 hypervisor.

To build our models, we used a CPU-intensive benchmark EP Embarrassing Parallel which is one of NAS Parallel Benchmarks (NPB) [5]. It generates pairs of Gaussian random deviates according to a specific scheme. EP is a multithreaded benchmark which runs number of threads corresponding to number of virtual CPU of a virtual machine. To evaluate our models, we used two other benchmarks of NPB namely CG and BT. BT benchmark is IO-intensive. It is a simulated CFD application that uses an implicit algorithm to solve 3-dimensional (3-D) compressible Navier-Stokes equations. CG benchmark is Memory-intensive which uses a Conjugate Gradient method to compute an approximation to the smallest eigen value of a large matrix. However, more details about the characteristics of NPB benchmarks is found in [11].

Finally, we used xenpm tool [1] to measure average running frequency and number of active cores. We used the CPU-Power measurement capability of our server to measure the power consumption of the CPU. In our experiments, the percentile average was considered to get accurate power readings. Fig. 1 summarizes the system overview and the procedures of CPU-Power models development. Fig. 1 shows the output of xenpm tool; it illustrates the change of average frequency, performance states (P0-P8), and sleeping states (C0-C3). Furthermore, the output demonstrates the percentage of time for each core and for each state.

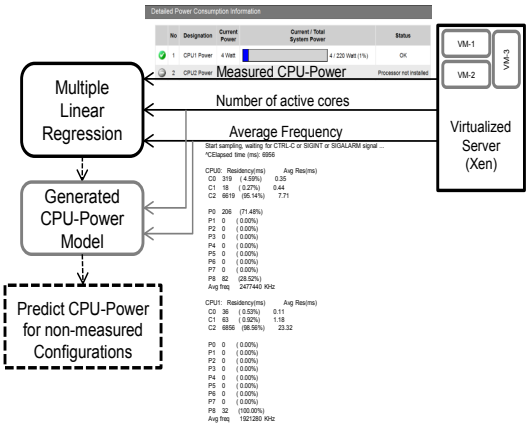


Figure 1. Overview of the system and CPU-Power models

3. Findings

Several works have used linear models to represent the power consumption of a system or just a processor. These models are based on CPU utilization or other concerned resources such as memory. As current processors have multicores which could operate at different frequency levels at runtime using DVFS, in this section we discuss the relationship between the CPU-Power consumption and CPU-frequency from one side, and the CPU-Power consumption and num-

ber of active cores from the other side.

3.1. CPU-Power and frequency relationship

CPU-Power consumption is composed of dynamic and static power. The dynamic power is the important factor for reducing power consumption using DVFS technique. The dynamic consumed power by a CPU with a capacitance C, frequency F, and supplied voltage V is computed by equation 1. However, Kim et al. [9] have considered power proportional to the cubic of frequency because the frequency is usually in proportion to the supplied voltage.

$$P = C.F.V_{dd}^2 \tag{1}$$

$$P_{(F)} = P_{min} + \theta(F - F_{min})^2 \tag{2}$$

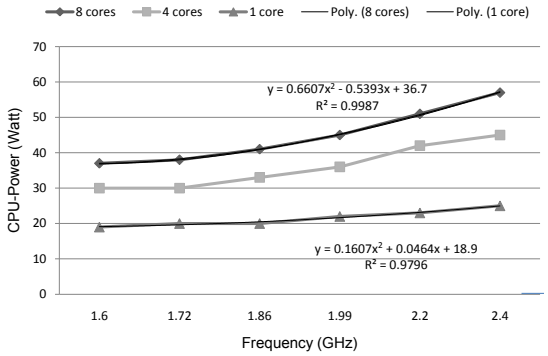


Figure 2. CPU-Power consumption relationship with frequency.

To obtain the relationship between CPU-Power consumption and frequency, we ran EP-NPB CPU-intensive benchmark on a virtual machine at different CPU frequencies. We measured the power consumption only for the CPU. Thus, we obtained the curves in fig. 2. Then, by applying the multiple linear regression, we found that the best relationship could be fit in polynomial linear with regression $R_2 = 0.99$. we could generalize it as a quadric model in equation 2 which resembles the proposed model by [13, 7] to estimate the power consumption of a server. Although fig. 2 shows perfect fitting for each curve of a number of cores, the total system variations cannot be covered with considering only frequency.

Hence, we need different values of θ and P_{min} at each number of active cores. For example, to estimate CPU-Power at frequency 1.72 GHz when 8 active cores using equation 2 the best values for θ and P_{min} are 37 watts and 37 $Watt/GHz^2$ respectively. The estimated power is 37.6 watts which is approximately equal to the measured value 38 watts. Nevertheless, The values of θ and P_{min} should be adapted

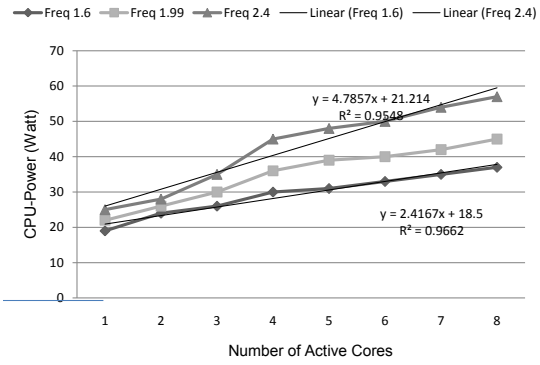


Figure 3. CPU-Power consumption relationship with number of

again to predicted the power when just 4 cores are active. Accordingly, we study the relationship between the power consumption and number of active cores in next section.

3.2. CPU-Power and number of active cores relationship

To estimate the power consumption of multicore processors, we found that it is important to study the relationship between CPU-Power and number of active cores. To this end, we obtained the curves in fig. 3. The curves have a linear trendline. The relationship is well approximated by a linear model with regression $R^2 0.95$, which means that the power consumption and number of active cores have a strong linear association and can be represented by equation 3. N is number of active cores, and P_{min} is the power consumed by one core running at frequency F. α_F is the slope of the power-to-active cores curve at frequency F. Importantly, each curve has two different slopes. The first one is when the number of active cores is less than 4 cores and the other one is when the number of active cores is more than 4 cores. Moreover, the first slope is greater than the second one. The main reason of this case was that we had a processor with 4 physical cores. Each physical core has two logic cores, and the power consumed by a logical core is less than the power consumed by a physical core.

$$P_{(N)} = P_{min} + \alpha.N \tag{3}$$

3.3. CPU-Power estimation models

From previous sections, we found a strong relationship between CPU-Power consumption and both frequency and number of active cores. In this section, we refer to the model adopted by [13, 7] as Model-0. Model-0 which is represented by equation 4 does not include number of active cores. Our first model is denoted by Model-1. Model-1 presented in equation 5 is

a multiple linear regression with the intercept constant C. Equation 6 represents Model-2. Model-2s intercept constant C is zero. Finally, we removed the first degree term of frequency of Model-2; we obtained Model- 3 represented by equation 7. However, we will study the predication accuracy of these models showing the worst and the best cases for each model.

$$P_{(F,N)} = \theta_2.F^2 + \theta_1.F + C \quad (4)$$

$$P_{(F,N)} = \theta_2.F^2 + \theta_1.F + \alpha.N + C \quad (5)$$

$$P_{(F,N)} = \theta_2.F^2 + \theta_1.F + \alpha.N \quad (6)$$

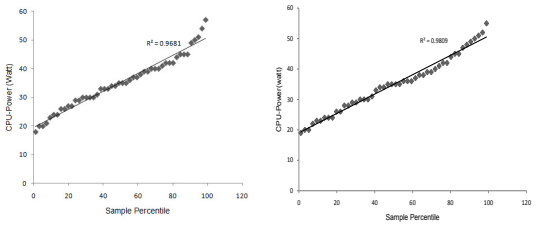
$$P_{(F,N)} = \theta_2.F^2 + \alpha.N \quad (7)$$

3.4. Statistical analysis

This section discusses some statistical analysis of our CPU-Power estimation models focusing on Model-1 to show its efficiency to predict the power consumption of a processor. We used plots to check models linearity and normality assumptions [3].

First, we tested the models linearity using a plot of residuals versus predicted values. Fig. 4-(a) represents residuals plot of Model-0. This plot shows a certain pattern indicating that the model makes systematic errors whenever the number of cores changes from 1 to 8. In fig. 4-(a), if we consider the first left vertical residuals points which represent residuals of frequency 1.6GHz, we find that the residuals values increase negatively when few of cores are active (e.g., the residual value is -11 when one core is active). On the other hand, the residuals values increase positively when the number of active cores is more than 4 cores. The reason was the over-estimation of CPU-Power with enabled logical cores. Moreover, the predicted power is limited by frequency. Thus, it gives a short range [29, 43]. Consequently, it will give serious errors when it will be used for predicting un-sampled data points. Furthermore, the residual rang [-15, 15] of Model-0 is wider than the residual rang [-3, 3] of Model-1. Fig. 4-(b) is residuals of Model-1. The points are symmetrically distributed around a horizontal line. This proves that our model satisfies the linearity assumption of the liner regression [3]. From this test, we conclude that our models could predict beyond the range of the sample data without significant errors. The predicted power is not limited by frequency and its range [17, 55] is wider than Model-0.

Second, we tested the models against normality using a normal probability plot of the residuals. Fig. 4 and 5 show similar plots of predicted and sample percentile points. These points are very close to the diagonal line which means that these models had normal distributed errors. Table 1 summarizes the determined values of CPU-Power models coefficients and statistics. This table shows the ranges of each coefficient of each model. Model-2 shows a small range for its coefficients. For instance, θ_2-range (i.e., [2.1, 5.10]) compared to Model-0's θ_2-range (i.e.,[-33.54, 42.9]) is very small. However, Model-2 and Model-3 are regressions with zero constant. Furthermore, the regression statistics in Table I show that the regression R^2 of our models is higher than R^2 of the Model-0. For instance, Model-2 and Model-3 have regression $R^2 0.99$ which means that these two models could explain 99% of the power variations. The power variations were determined by variations in the independent variables (i.e., frequency and active cores). In contrast, Model-0, which only considers frequency, has regression $R^2 0.259$. Model-0 explained only 25% of power variations using frequency.



(a) Model-0 (b) Model-1

Figure 5. Normal probability plot.

3.5. Performance evaluation

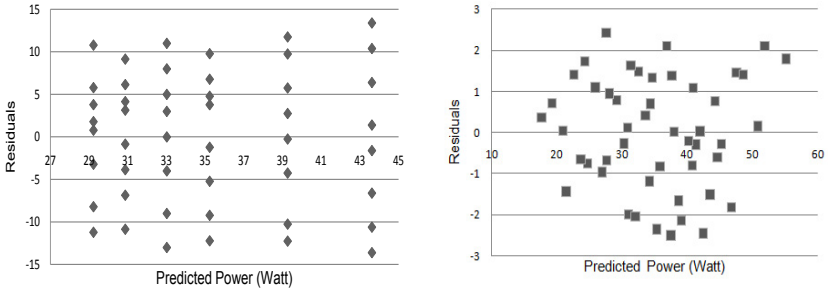
As we discussed models performance statistically, in this section we show and compare the performance of the models experimentally. To achieve this, we conducted three experiments using three different benchmarks of NPB benchmark namely EP, CG, and BT. These benchmarks represent CPU-Intensive, Memory-Intensive, and IO-Intensive applications respectively.

1) CPU-intensive applications: To evaluate our models against CPU-Intensive applications, we used EP benchmark to generate a workload which was changed with time. As shown in fig. 6, we started with a low workload which increased with time until it reached its maximum approximately at time 205 sec. Then, it started to decrease after time 250 sec. During the experiment, we measured the CPU-Power consumption every 5 seconds. Then, we computed the estimated power using the four different models. Obviously, the curve shows that Model-0 has a big difference between its estimation and the measured power when low-workload and few of cores are actives (i.e., 1-3 active cores). However, it shows a good performance in high-workload when all the cores are active. This case is similar to estimation power consumption of a processor as a unit regardless of active cores number.

As our models include the number of active cores and the average frequency, they accurately estimated the power in both areas of workload (i.e., low-workload and high-workload). Furthermore, although Model-2 and Model-3 statistically (i.e., regression R^2) are better than Model-1, the experiment demonstrated that

Model	θ_2-range	θ_2	θ_1-range	θ_1	α-range	α	C-range	C	Std. Err.	R^2
0. Equ.4	-29.98 - 42.6	6.31	-152.8 - 138.25	-7.27	0	0	-118.86 - 168.24	24.68	7.95	0.291
1. Equ.5	-0.14 - 12.77	6.31	-33.17 - 18.63	-7.27	3.12 - 3.48	3.3	-15.77 - 35.36	9.79	1.41	0.978
2. Equ.6	3.08 - 4.63	3.8	0.88 - 4.37	2.63	3.13 - 3.49	3.31	0	0	1.40	0.998
3. Equ.7	4.78 - 5.20	4.99	0	0	3.26 - 3.60	3.43	0	0	1.53	0.998

Table 1. The determined values of CPU-Power models coefficients and statistics.



(a) Model-0 (b) Model-1

Figure 4. Predicted power and residual plot.

Model-1 with constant C achieved better performance than the other models. Finally, slight percentage of error could be observed in our models due to the fact that we considered each logical core as physical core, but as we mentioned before in section 3.2 the power consumed by a logical core is less than the power consumed by a physical core.

Now, we discuss the prediction accuracy by computing the percentage of error using the following formula.

$PoE = |(Estimated - Measured)/Measured| * 100\%$

Furthermore, we obtained the Empirical Cumulative Distribution Function of Percentage of Error CDF(PoE). The plot of CDF(PoE) for each model is depicted in fig. 7. The x-axis represents the Percentage of Error (PoE), and y-axis shows the percentage of data points (i.e., predicted power values) that achieve error less than each value of x. For instance, 90% of the predicted values using Model-0 have less than 40%, and this error could increase to 50%. On the other hand, our models show that 90% of values were predicted with less than 9% error. Although R^2 value of Model-1 is less than R^2 value of both Model-2 and Model-3, Model-1 showed the best results where 95% of the predicted values had error less than 7% error. Generally, the prediction accuracy of Model-1 empirically outperforms the prediction accuracy of Model-2 and Model-3.

Significantly, our proposed models achieve high prediction accuracy due to considering both number of active cores and the average running frequency. Additionally, the accurate readings of CPU-Power that was realized using CPU-Power measurement capability of our server assisted us to build these accurate models. To test our models ability to predict those un-sampled combinations of frequency and number of cores, we conducted some experiments with different un-sampled combinations frequency 2.53GHz and number of cores. Table 2 presents the results of these experiments. The results proved that our models are capable to predict with high accuracy even un-sampled combinations of frequency and number of cores.

2) Memory-intensive applications: In this section, we evaluated performance of the models for applications that are considered memory-intensive using CG benchmark. Fig. 8 shows the estimated power versus the measured power. The diagonal line represents the perfect predication line which illustrates the deviation of the estimated values from the measured values. In other words, the predicted value is equal or close to the measured value if it is one of the perfect predication line points or close to this line. From fig. 8, the data points that represent our models are either on or close to perfect predication line. Furthermore, we computed the maximum prediction error of each Model. We found that Model-1 and Model-2 had less maximum prediction error compared to the other two models. However, with less than 6% maximum prediction error for Model-1 and Model-2, these two models are still accurate. The maximum prediction error of Model-0 was 14.4%.

Cores	Measured	Model-0	Model-1	Model-2	Model-3
3	42	47.09	42.27	41.23	42.06
4	45	47.09	45.57	44.53	45.46
8	58	47.09	58.77	57.73	59.06

Table 2. The predicted CPU-Power for un-sampled combination

3) IO-intensive applications: As we presented the performance of our models for CPU-intensive and Memory-intensive applications in the previous sections, this section presents performance of the mod-

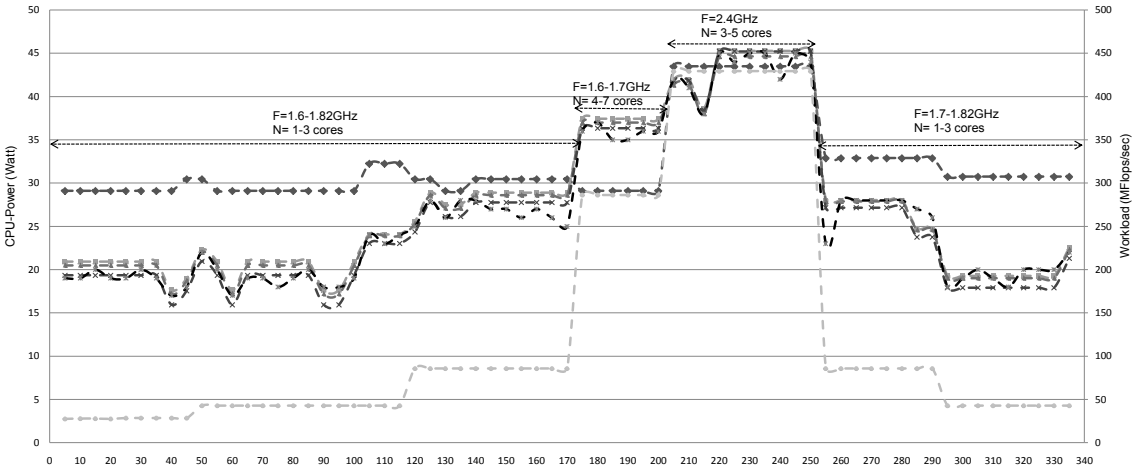


Figure 6. Trace of measured consumed CPU-Power and predicted CPU-Power for the four models.

els for IO-intensive applications. We repeated the experiments procedure of the previous section using BT benchmark. Fig. 9 also shows the estimated power versus the measured power. We can see that the predicted values of Model-1 are on perfect predication line or very close to it. In contrast, Model-0 and Model-3 show big deviation of the perfect line. Moreover, we found that Model-1 and Model- 2 had less maximum prediction error compared to the other two models. Model-1 and Model-2 achieved less than 5% maximum prediction error. Finally, the maximum prediction error of Model-0 became worse with 22.07% error.

3.6. Conclusions and next steps

In this project phase, we developed models to estimate the power consumption of multicore processors. Our work is distinguished from previous works in combining number of active cores with P-state of multicore processor. We develop our prediction models using an Intel(R) Xeon(R) CPU E5540 processor. Additionally, our models are based on a virtualized server that could host multiple heterogeneous applications. We validated our proposed models using statistical analysis and experimental approach using varied workloads. The results of the experiment showed that our model achieved high accuracy of CPU-Power estimation. Thus, our next steps are as follows:

- Evaluating our proposed using different types of applications and systems that consist of cores more than 4 cores.
- Applying our proposed models to dynamic power-aware configuration for a virtual machine in terms of number of cores and frequency. This enables new adaptive power management solution for virtualized servers. Furthermore, they could be used to realize fine-grained power provisioning proportional to workloads.
- Developing a mechanism to estimate the consumed power by each virtual machine.

References

[1] Xen wiki. http://wiki.xensource.com/xenwiki/xenpm, Accessed 30-6-2011.

[2] Y. Ben-Itzhak, I. Cidon, and A. Kolodny. Performance and power aware cmp thread allocation modeling. In *HiPEAC*, pages 232–246, 2010.

[3] S. Chatterjee and A. S. Hadi. *Regression analysis by example*. John Wiley and Sons, 2006.

[4] S. Dawson-Haggerty, A. Krioukov, and D. E. Culler. Power optimization - a reality check. Technical Report UCB/EECS-2009-140, EECS Department, University of California, Berkeley, Oct. 2009.

[5] R. V. der Wijngaart. Nas parallel benchmarks v. 2.4. Technical Report NAS-02-007, NAS, 2002.

[6] X. Fan, W.-D. Weber, and L. Barroso. Power provisioning for a warehouse-sized computer. In *34th annual international symposium on Computer architecture*, pages 13–23, San Diego, California, USA, 2007. ACM.

[7] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *SIGMETRICS*, June 2009.

[8] J. Hamilton. Cooperative expendable micro-slice servers (cems): Low cost, low power servers for internet-scale services. In *Innovative Data Systems Research CIDR09*, Jan. 2009.

[9] K. H. Kim, A. Beloglazov, and R. Buyya. *Power-aware provisioning of virtual machines for real-time Cloud services. Concurrency and Computation: Practice and Experience*. John Wiley and Sons, 2011.

[10] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: coordinated multi-level power management for the data center. In *ASPLOS*, Mar. 2008.

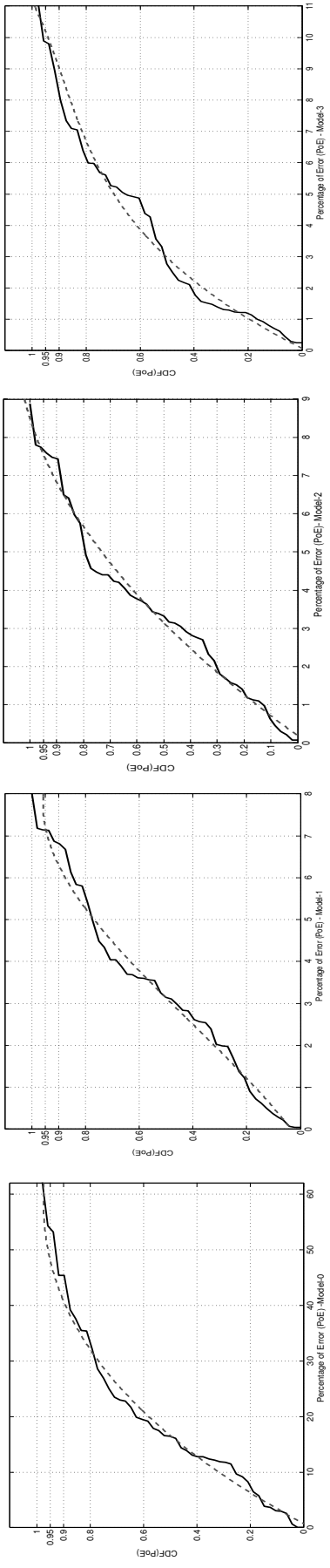


Figure 7. Prediction accuracy of the CPU-Power models.

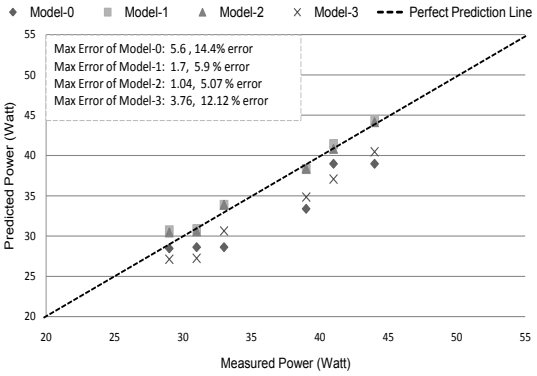


Figure 8. CPU-Power models fit for CG-Memory-Intensive:

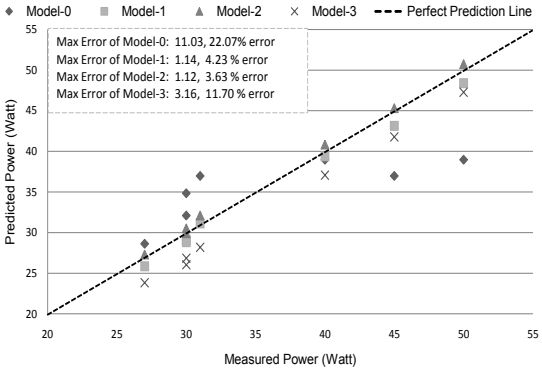


Figure 9. CPU-Power models fit for BT-IO-Intensive: measured

[11] J. Subhlok, S. Venkataramaiah, and A. Singh. Characterizing nas benchmark performance on shared heterogeneous networks. In *International Parallel and Distributed Processing Symposium: IPDPS 2002 Workshops*, volume 2, page 0086, 2002.

[12] R. Urgaonkar, U. Kozat, K. Igarashi, and M. Neely. Dynamic resource allocation and power management in virtualized data centers. *IEEE Network Operations and Management Symposium (NOMS)*, pages 479–486, 2010.

[13] R. Urgaonkar, U. Kozat, K. Igarashi, and M. Neely. Dynamic resource allocation and power management in virtualized data centers. *2010 IEEE Network Operations and Management Symposium (NOMS), IEEE*, pages 479–486, 2010.

[14] U.S. Environmental Protection Agency. Epa report on server and data center energy efficiency. ENERGY STAR Program, 2007.

Towards Multi-Core and In-Memory for IDS Alert Correlation: Approaches and Capabilities

Sebastian Roschke, Peter Ernicke, Martin Kreichgauer, Michael Frister, Florian Thomas,
Feng Cheng, Christoph Meinel
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam
{sebastian.roschke, feng.cheng, meinel}@hpi.uni-potsdam.de
{peter.ernicke, martin.kreichgauer, michael.frister, florian.thomas}@student.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 usually addressed by correlation and clustering of alerts. As real-time analysis is crucial for security operators, 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. We identify the data storage and processing algorithms to be the most important factors influencing the performance of clustering and correlation. The Security Analytics Lab (SAL) is developed to make use of multi-core and in-memory processing. Using the SAL, a multitude of algorithms is implemented, such as Attack-Graph based correlation using HMMs, QROCK categorical clustering, and rule-based correlation using a knowledge base. The SAL is using the Common Event Expression (CEE) and supports generic flat log data.

1 Alert Correlation and its Performance

The alert correlation framework usually consists of several components [4]: *Normalization*, *Aggregation (Clustering)*, *Correlation*, *False Alert Reduction*, *Attack Strategy Analysis*, and *Prioritization*. Over the last years, alert correlation research focused on new methods and technologies for these components. IDMEF [5] and CVE [17] are important efforts in the field of *Normalization*. Approaches of aggregation are mostly based on similarity of alerts or generalization hierarchies. The correlation algorithms [4] can be classified as: *Scenario-based correlation*, *Rule-based correlation*, *Statistical correlation*, and *Temporal correlation*. Most of the efforts do not consider the aspect of performance, which is needed in case of huge amounts of alerts, as well as the scenarios requiring real-time.

The efficiency of the correlation depends on the quality and performance of the algorithm as well as the storage and organization of original alerts. The quality is a measure of the correctness of the algorithm and depicts how many of the recognized correlations are correct, i.e., how many of the correlations found represent existing relations between alerts. Furthermore, it depicts how many of the existing relations between alerts are found by the algorithm. The performance of the correlation describes the amount of time needed to correlate a number of alerts. Due to the complexity of large scale networks, the amount of alerts increases significantly. Therefore, the performance of correlation algorithms is a major aspect of the efficiency of correlation.

The work described in [14] considers the performance of alert correlation by using a memory-based index for hyper alerts. A hyper alert is a cluster of alerts with the same properties, e.g., the same source address or target address. The approach using index tables is introduced in [11]. To perform correlation in real-time, the approach filters and clusters alerts to hyper alerts, which reduces the number of processed alerts significantly. However, this technique may lead to inaccurate results of the correlation, as multiple alerts are generalized to a single hyper alert. The approach reaches a correlation rate on the order of $100,000$ alerts per second based on the massive reduction of alerts by clustering in hyper alerts. In [2] the *data storage* and the *processing algorithms* have been identified to be the most important factors influencing the performance of clustering and correlation. The platform introduced in [2] considers different storage mechanisms and can handle up to $100,000,000$ for specific algorithms that make heavy use of the caching mechanisms of the platform. For storage, a column-based database, an In-Memory alert storage, and memory-based index tables lead to significant improvements of the performance. Although this work considers data and task distribu-

tion in general, the platform is not mature enough to distribute one correlation algorithm over multiple computing cores. Furthermore, using a hybrid memory architecture and GPU based computing is not considered.
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. The Security Analytics Lab (SAL) was used by master students to implement and test various algorithms: a rule-based correlation, an Attack Graph based algorithm that uses HMMs, and the ROCK/QROCK clustering algorithm. The platform could be extended to support generic flat log files for correlation using the Common Event Expression (CEE) standard [7].
Apart from the practical achievements, we have been able to publish papers on the correlation platform [2] and started research on a complex correlation algorithm using attack graph data and environmental information for IDS correlation [1]. The Attack Graph based algorithm is described in [3]. Additional results are summarized in the following subsections.
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. Furthermore, we conducted some tests and experiments using the NVIDIA FluiDyna System as well as the Fujitsu RX600 S5 1. The following feature set has been realized:

- Attack graph based correlation algorithm using HMMs
- Rule-based correlation using a knowledge base of predicates
- QROCK/ROCK clustering for IDS alerts
- Common Event Expression (CEE) support for processing of flat log files
- Snort alert generator that generates IDS alerts using a network description
- 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
- Multi-core support for OpenCL and MapReduce
- Visualization of correlation results

2.1 Normalization using CEE

The data format used in the platform should be standardized to improve flexibility and applicability, i.e., many different SIEM systems, sensors, and log gatherers need to be attached to the platform. The Intrusion Detection Message Exchange Format (IDMEF) [5] was the first approach for implementing the platform, but it yields drawbacks in terms of expressiveness of log events. Thus, the Common Event Expression (CEE) [7] was chosen for the current release of the platform.
To benefit from the new approaching standard and for being able to connect log gatherers to the platform, a preliminary implementation of CEE is developed and used in the platform. A fixed set of field identifiers is used that is extracted from the current set of white papers and suggestions on the mailing list. The implementation is generic to enable new field identifiers to be integrated easily. The implementation is done using Google Protobuf [18]. Protobuf provides a simple and efficient method to serialize structured data. The data types are defined using an abstract language and so called message definitions for each entity of the data format. The message descriptions are translated in a set of Java classes that can be used within the correlation platform. Each message is stored in a simple key-value manner, i.e., each possible field has an identifier (key) and a corresponding value. Apart from the predefined key set in the dictionary of the standard, a user can define individual keys.
As the standard is under development, a fixed set of field identifiers is chosen based on the mailing list discussions and existing requirements. Due to the open key-value structure of the standard, additional fields can be added easily.

- messageid - the unique identifier of an event
- prod_name - the name of the event producer/analyzer

- event_type - the type of the event, either IDS or log
- creation_time - the time of the creation of the event
- classification_text - the message of the event
- reference - existing references to a certain event
- source_name - the name of the source responsible for the action described by the event
- source_address - the address (IP) of the source responsible for the action described by the event
- source_port - the source port of the connection responsible for the action described by the event
- target_name - the name of the target of the action described by the event
- target_address - the address (IP) of the target of the action described by the event
- target_user - the target user of the action described by the event
- target_userid - the id of the target user of the action described by the event
- target_port - the target port of the connection responsible for the action described by the event
- target_process - the name of the target process of the action described by the event
- target_processid - the id of the target process of the action described by the event
- target_service - the name of the target service of the action described by the event

A mapping from IDMEF alerts to CEE events is possible without information loss. Most of the fields in the IDMEF message can directly be mapped to CEE, such as IDMEF's *analyzer_name* to CEE's *prod_name*. As CEE offers the possibility to add additional fields, any IDMEF field with no counterpart in CEE can simply be added as key-value pair. Special attention is needed for fields in IDMEF that support a list of values, such as *target*. IDMEF supports alerts that are describing an attack on multiple targets at once, e.g., a flooding attack on a subnet of hosts. These cases can be handled in two different ways. The alert covering multiple targets can be split up in multiple CEE events, each describing a single target. Alternatively, the CEE event can be extended by fields using a naming convention, such as *target_address_1*, *target_address_2*, *target_address_3*, and so on. In this way, events can even cover lists of values. The mapping from CEE to IDMEF is possible, but might lead to information loss, if certain fields in CEE have no counterpart in IDMEF.

2.2 Categorical Clustering with QROCK

In contrast to the k-means algorithm, the ROCK clustering algorithm enables the clustering of categorical data [15]. It is based on the concept of *Neighbors* and *Links*. Neighbors are defined by a similarity function $sim(x, y)$ and a threshold ρ. If the result of the similarity function is greater than the threshold, both are defined as neighbors.

$$sim(x, y) \geq \rho \tag{1}$$

with

$$0 \leq sim(x, y) \leq 1 \tag{2}$$

The similarity function for events is defined based on the Jaccard Coefficient [15]. Let $x, y \in (E)$ be a tuple $x = (ts, s, d, sp, dp, ref)$ with ts as the timestamp, s as the source address, d as the destination address, sp as the source port, dp as the destination port, and ref as the reference. $\mid x \mid$ defines the arity of the tuple, i.e., in our example $\mid x \mid = 6$. The distance $d(x, y)$ between two events $x, y \in (E)$ is the number of values in the tuple that are equal. Based on the distance, the similarity is defined as:

$$sim(x, y) = \frac{d(x, y)}{\mid x \mid} \tag{3}$$

Furthermore, the *Link* $link(x, y)$ is the number of common neighbors of the events x and y. This leads to the observation that if the number of common neighbors is high, there is also a high probability that both belong to the same cluster. The clustering algorithm works based on a goodness function $g(C_i, C_j)$ that is defined in [15]:

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{(n_i + n_j)^{1+2f(\rho)} - {n_i}^{1+2f(\rho)} - {n_j}^{1+2f(\rho)}} \tag{4}$$

with

$$link[C_i, C_j] = \sum_{p_q \in C_i, p_r \in C_j} link(p_q, p_r). \tag{5}$$

The ROCK clustering algorithm for IDS correlation is implemented similar to [15]. After the initial link/neighbor computation, two heaps are created (global heap and local heap) and created clusters are merged according to the goodness values. After each merge, the links are calculated again for the elements of the new cluster. This will be repeated until there are only k clusters left.
The QROCK clustering provides a method to improve efficiency of the ROCK algorithm [16]. The QROCK clustering algorithm can be applied to IDS correlation and is implemented on the platform. Both algorithms are parallelized by distributing the workload of the link computation, i.e, the first step of the algorithm.

3 Rule-based Correlation

The algorithm proposed in [19] is implemented on the platform. It works based on pre-defined prerequisites and consequences (pre- and post-conditions). The approach uses a definition of so called hyper alerts $T = (fact, pre-condition, post-condition)$. Ning et. al. provide a database of rules for the know set of snort alerts [1] that can be used for implementing the proposed algorithm. The parallel version of the algorithm is implemented as shown in Listing 1. The inner loop of the algorithm is executed by workers and thus parallelized.

```
attack_step_correlation(events,
    knowledge_base) {
  hyper_alert_types =
    read_hyper_alert_types();
  hyper_alerts =
    createHyperAlerts(events,
      hyper_alert_types);

  for (ha_1 in hyper_alerts) {
    for (ha_2 in hyper_alerts) {
      if (ha_2.prerequisite
        .subsetOf(ha_1.consequence)
        && ha_1.time < ha_2.time) {
        resultGraph
          .addConnection(ha_1,
            ha_2);
      }
    }
  }
  return resultGraph;
}
```

Listing 1. Simple Attack Step Correlation Algorithm

4 AG-based Correlation Algorithm using HMM

To create an HMM, a special form of an attack graph is used, which is designed for conversion to an HMM. The attack graphs nodes are either a host or if one or more vulnerabilities are known, a combination of host and vulnerability. The hosts and their vulnerabilities in the network structure as well as the information which networks and hosts are connected is known (can be collected by scanning). The host and vulnerability information is used to match alerts to a specific node in the attack graph.

The first two steps of the algorithm create two dictionaries: the networkMap contains lists of hosts in each network and the networkConnectionMap maintains for each network a list of connected networks. Two networks are defined as connected if there is one host connected to both networks. The idea is that an attacker can exploit a host and use it to forward packets into another network. Starting from one specified network, the algorithm walks through the network structure until all networks were visited. While walking through the networks, it creates a directed graph containing nodes as vertices and edges as possible connections between them. The direction of these edges is defined by the order in which the algorithm walks through the network. For example, when starting at network A, it creates edges originating from hosts in this network to hosts in the connected network B. The algorithm creates groups for hosts in the same network that are used as single node.

After walking the networks, the algorithm adds the transitive closure to the graph. The closure allows detecting attacks on a host if a previous attack on the way to this host was not detected by the NIDS. Additionally the algorithm adds reflexivity to the graph to manage multiple alerts in a row for one node.

The Hidden Markov Model is build using the attack graph created before. Each node in the graph is transformed to a state in the model. All edges, which are directed, are added to the model. Possible observations are created from known vulnerabilities, from a category of alerts which is assumed to result from attacks and are no false positives and from a category for all other alerts, which may or may not be relevant. The transition probabilities are calculated by counting the number of outgoing edges from one node and are assigned to the edges as the reciprocal of the count, so each transition from one node has the same probability. Observations for known vulnerabilities are created as a combination of host and vulnerability, such as attack graph nodes. Vulnerabilities are identified by their CVE identifier [17]. Alerts from NIDS also may have CVE Identifiers attached, thus, these can later be used for matching an alert to a specific observation. Observations for attacks in general and for all other alerts are each created per host. For a each state, each unrelated observation is assigned with a low priority. This is necessary since the attack graph allows only going forward in the network structure. Any alert triggering an observation with a probability of 0 would disturb the algorithm for finding the most probable path.

5 Future Work

Within the next few months, to prepare the correlation platform for further research and experiments as well as conduct more experiments on different algorithms. We would like to work towards our vision with the following steps:

- Test the platform with a dataset of 1 TB
- Implement more algorithms with multi-core support
- Implement database switching mechanism

[1] http://discovery.csc.ncsu.edu/software/correlator/ver1.0/index.html

- Research on correlation algorithms that are using environment information and attack graphs
- Research on statistical correlation algorithms
- Research on visualization techniques for correlation results

References

[1] 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).

[2] Roschke, S., Cheng, F., Meinel, Ch.: An Alert Correlation Platform for Memory-Supported Techniques. In: Concurrency and Computation, Wiley Blackwell, 2011 (to appear).

[3] Roschke, S., Cheng, F., Meinel, Ch.: A New Correlation Algorithm based on Attack Graph. In: Proceedings of the 4th Conference on Computational Intelligence in Security for Information Systems (CISIS'11), Springer LNCS 6694, Torremolinos, Spain, pp. 58-67 (2011).

[4] 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).

[5] 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).

[6] Mitre Corporation: *Common vulnerabilities and exposures (CVE)*, WEBSITE: http://cve.mitre.org/ (accessed Apr 2011).

[7] Mitre Corporation: *Common Event Expression (CEE)*, WEBSITE: http://cee.mitre.org/ (accessed Apr 2011).

[8] 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).

[9] 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).

[10] 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).

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

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

[13] Arnes, A., Valeur, F., Vigna, G., Kemmerer, R.: Using Hidden Markov Models to Evaluate the Risks of Intrusions: System Architecture and Model Validation. In: *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID'06)*, Springer LNCS 4219, Hamburg, Germany, pp. 145-164 (2006).

[14] 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).

[15] Guha, S., Rastogi, R., Kyuseok, S.: ROCK: A Robust Clustering Algorithm for Categorical Attributes, In: Proceedings of the *15th International Conference on Data Engineering (ICDE'99)*, IEEE Press, Washington, DC, USA, pp. 512-537 (1999).

[16] Dutta, M., Kakoti Mahanta, A., Pujari, A. K.: QROCK: A quick version of the ROCK algorithm for clustering of categorical data. In: Pattern Recognition Letters, Elsevier, vol. 26(15), pp. 2364-2373 (2005).

[17] Mitre Corporation: *Common vulnerabilities and exposures (CVE)*, WEBSITE: http://cve.mitre.org/ (accessed May 2011).

[18] Google Protobuf, WEBSITE: http://code.google.com/p/protobuf/ (accessed May 2011).

[19] Ning, P., Cui, Y., Reeves, D.: Constructing attack scenarios through correlation of intrusion alerts, In: Proceedings of the *9th ACM Conference on Computer and Communications Security (CCS'02)*, ACM, New York, NY, USA, pp. 245-254 (2002).

Duplicate Detection on GPUs

Benedikt Forchhammer[1], Thorsten Papenbrock[1], Thomas Stening[1], Sven Viehmeier[1],
Uwe Draisbach[2], Felix Naumann[2]
Hasso Plattner Institute, Potsdam, Germany
[1]firstname.lastname@student.hpi.uni-potsdam.de [2]firstname.lastname@hpi.uni-potsdam.de

Abstract

Duplicate detection is an integral part of data cleansing. In this project we developed a complete system to detect duplicates in very large datasets, using the capabilities of modern graphics processing units (GPUs). Our solution covers several algorithms for the tasks of pair selection, similarity-comparison of attribute values, aggregation of pairs, and clustering. We describe how each algorithm can be designed to run memory efficient and parallel on the GPU. Thereby, we exploit the increasing capabilities of modern graphics cards with their many cores. Our similarity-comparisons are based on strings with variable lengths. This is a difficult task for GPUs, because they cannot handle variable sized data structures and lose synchronism, but in return the duplicate detection process achieves higher precision values. Experiments demonstrate that our solution outperforms an equivalent CPU-based implementation on large, real-world datasets. For example, the GPU of a standard computer can compute a dataset with 1.8 million entries using our algorithm 10 times faster than the respective CPU. Furthermore, with an increasing size of the dataset the GPU becomes progressively faster than the CPU.

1. Introduction

Duplicate detection is the task of identifying multiple representations of the same real-world entities [1]. This is typically done by applying similarity functions to pairs of entries in datasets. Some algorithm carefully selects promising pairs of records. If the values of two records are sufficiently similar, according to some threshold, they are assumed to be duplicates. Due to the high number of comparisons and the ever increasing size of many databases, duplicate detection is a problem that is hard to solve efficiently. However, in most approaches the comparisons of record pairs are independent from one another – the problem is embarrassingly parallelizable. In this project a selection of duplicate detection algorithms and similarity measures are described and adapted in the context of general-purpose computation on Graphics Processing Units (GPUs).

Currently, there are only few frameworks available for GPGPU development. For our prototype, we decided to use the OpenCL 1.0 framework, as it allows development for both ATI and NVIDIA graphics cards. The framework allows the execution of so-called *kernels*, which are written in a variant of basic C. OpenCL kernels can be executed on different *devices*; usually the device is a graphics card, but other devices, in particular the CPU, are also possible if respective hardware drivers are available. Devices execute kernels as work items. A work item is a set of instructions that are executed on speciffic data by one thread.

When developing applications for GPUs, memory management is a key factor that needs much attention. The main reason for this necessity is the fact that GPUs have four different types of memory with different capacities and different access speeds. An additional difficulty lies in the fact that it is not possible to allocate memory dynamically on the GPU. Figure 1 presents a complete duplicate detection workflow, which combines common duplicate detection algorithms with the computation capacities of modern graphics cards. The work ow consists of the following steps:

- **Parsing:** This step converts the input data, e. g., given as a CSV file, into an internal format. Each attribute is thereby represented as a character array where all values are concatenated. Since we work on values with different lengths, an additional array containing the starting indices of the single attribute values is needed. This format is essential because GPU-kernels can only handle basic datatypes and arrays whose sizes are known beforehand.

- **Pair Selection:** This step selects record pairs for comparison. We adapt the Cartesian product and the Sorted Neighborhood algorithms to run on the GPU. To generate a sorting key for the Sorted Neighborhood algorithm, we present a simple *key-generation* function and an adapted *Soundex* algorithm; both running on the GPU.

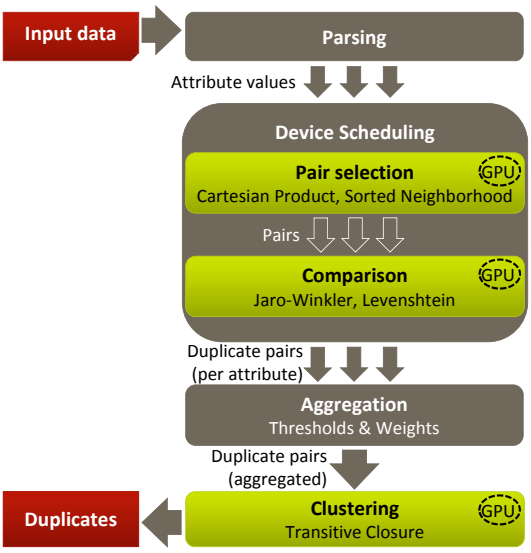


Figure 1. The overall workflow of the duplicate detection process

- **Comparison:** The previously selected record pairs are compared to decide whether the records are similar. Thereby, we process each attribute value individually and return a normalized similarity value for each pair of attribute values. We show two edit-based techniques on the GPU: *Levenshtein* and *Jaro-Winkler*.

- **Aggregation:** The attribute similarities are aggregated to a record pair similarity, which is used to decide whether the two records are duplicates or not. We calculate a weighted average and check similarity values before and after the aggregation against predefined thresholds.

- **Clustering:** The result of a pairwise duplicate detection process may not contain all transitively related record pairs. Thus, we calculate the transitive closure to obtain a complete list of duplicate clusters.

2. Evaluation

We performed our evaluation on four different graphics cards, two made by Nvidia and two by ATI. As ATI's OpenCL drivers also allow the execution of OpenCL kernels on CPUs, we additionally evaluated our implementation on two Intel CPUs (see Tab. 1 for specifics of all six devices; Configuration G2 represents the hardware provided by the FutureSOC Lab).
We use a subset of 1.9 million music CDs extracted from FreeDB[2] for our evaluation. This dataset

[1]http://www.alternate.de (August 2011)
[2]http://freedb.org

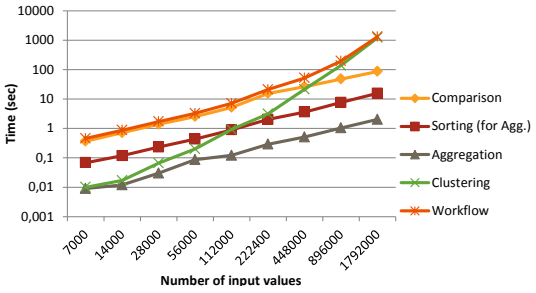


Figure 2. Execution times of the different components and the complete workflow

contains attributes such as artist, title, genre, year of publication, and multiple tracks. The similarity of two records is calculated based on the values of four attributes that contain strings of variable length (see example records in Tab. 2).

Artist	Title	Track01	Track02
Shane B.	Psalms	Unto You	Waiting Room
Metallica	Metallica	Enter Sandman	S.B.T.
Dido	No Angel	Here With Me	Hunter

Table 2. CD Example Data

To evaluate the duplicate detection workflow, we first analyze the execution times of its components. To this end, we execute all tests on the NVidia GeForce GTX 570 (G1), because our experiments showed that this device performs best.
Figure 2 shows the overall execution times of the different components of our work ow for various input sizes. We used Jaro-Winkler for comparison and Sorted Neighborhood for pair selection. The diagram shows that with an increasing amount of data, the execution time of the transitive closure step becomes the dominant part of the workflow.
We executed the same configuration on six different devices (see Tab. 1) to analyze the feasibility of using GPUs instead of traditional CPUs for duplicate detection.

Figure 3 shows that the best results are indeed achieved on GPUs. The fastest GPU G1 (see Tab. 1) takes 35 minutes (2,095 seconds) to process 1.792 million entries; this is about 10 times faster than the fastest CPU C2, which takes 335 minutes (20,128 seconds).
The first part of our workflow, i. e., the comparisons of multiple attributes, can be distributed over multiple graphics cards. In our experiments, the comparisons of four attributes on four graphics cards need only 30% of the time that they need on one graphics card. In theory, the comparisons on multiple graphics cards would only need $\frac{1}{|graphicscards|}$ of the time on one graphics card. This assumes that all comparisons

Id	Type	Device Name	Clock	Memory	Cores	System	Price[1]
G1	GPU	Nvidia GeForce GTX 570	732 MHz	1280 MB GDDR5	480	CUDA Win64	279 Euro
G2	**GPU**	**Nvidia Tesla C2050**	**1147 MHz**	**3071 MB GDDR5**	**448**	**CUDA Linux64**	**2149 Euro**
G3	GPU	ATI Radeon HD 5700	850 MHz	1024 MB GDDR5	800 SP	Win64	91 Euro
G4	GPU	ATI Mobility Radeon HD 5650	450 MHz	1024 MB GDDR3	400 SP	Win64	unknown
C1	CPU	Intel Core i5 750	2.67 GHz	8192 MB DDR3	4	Win64	185 Euro
C2	CPU	Intel Core i5 M560	2.67 GHz	8192 MB DDR3	2	Win64	200 Euro

Table 1. Evaluation devices

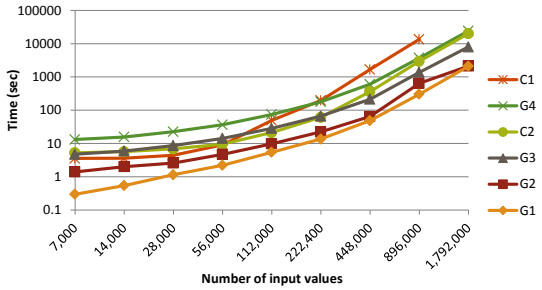


Figure 3. Performance of Sorted Neighborhood with Jaro-Winkler on different devices

have an equal execution time; in practice, the execution time in the parallel case depends on the longest comparison. This leads to an execution time of only 30% compared to the theoretically possible 25%.

Lee et al. have pointed out, that CPUs and GPUs cannot easily be compared in a fair way, without first optimizing for both devices [3]. Our measurements can therefore not be generalized for the overall relation of runtimes betwen GPUs and GPUs. However, our results are a good indication that graphics cards can compete well in the field of duplicate detection.

3. Conclusion

We have presented and evaluated a complete duplicate detection work ow that uses graphics cards to speed up the execution. The work ow uses either the Cartesian product or the Sorted Neighborhood Method for pair selection, and calculates the similarity of a record pair with measures such as Levenshtein, Jaro-Winkler, and Soundex.

The major challenge of using graphics cards for duplicate detection is the processing of input values with different lengths, as this leads to inefficient loops with different numbers of loop passes and branches in the GPU-kernels. Another challenge is memory management, as dynamic memory allocation is not possible and each hardware platform needs different optimizations.

The evaluation of our workflow shows that modern GPUs can execute the duplicate detection workflow faster than modern CPUs. Furthermore, the workflow and algorithms are scalable and can process large datasets. Additionally, experiments show that the access of global memory on graphics cards is the biggest bottleneck and has a great impact on the performance of our algorithms.

Multiple improvements of the duplicate detection workflow can be evaluated in the future. Memory access on the GPU can still be improved: Coalesced access patterns and the use of local memory could further speed up the execution. More optimizations concerning concrete hardware platforms are possible and could be applied for a concrete usage of the workflow (see also [2]). At the moment, only the comparisons of different attributes are distributed over all available devices. Thus, other algorithms, especially the computation of the transitive closure, could be further optimized to scale out on multiple devices. The implementation and evaluation of more similarity measures, e. g., token-based approaches, would allow the processing of real-world data with different properties and make the work ow more adaptable.

In summary, we showed that GPUs have a high potential to speed up duplicate detection tasks, especially if the GPU is used in combination with modern CPUs.

References

[1] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1):1–16, 2007.

[2] F. Feinbube, P. Tröger, and A. Polze. Joint Forces: From Multithreaded Programming to GPU Computing. *IEEE Software*, 28:51–57, 2011.

[3] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In *Proceedings of the International Symposium on Computer Architecture*, pages 451–460, Saint-Malo, France, 2010.

ECRAM (Elastic Cooperative RAM) HPI Future SOC Lab Project Report

Kim-Thomas Rehmann, Kevin Beineke and Michael Schöttner
Institut für Informatik, Heinrich-Heine-Universität Düsseldorf,
Universitätsstrae 1, 40225 Düsseldorf, Germany
Kim-Thomas.Rehmann@uni-duesseldorf.de

1. Project Idea

The ECRAM (Elastic Cooperative RAM) project, which started in 2011 at Heinrich-Heine-Universitt Dsseldorf, aims at providing distributed inmemory storage for cloud applications. ECRAM is designed to scale storage capacity depending on the dynamic load. By storing data in RAM, we reduce data-access latency and support arbitrary, highly concurrent accesspatterns. In contrast to traditional application-level caches, data is not flushed to disk, but always kept in memory. Persistence is realized by asynchronous logging and checkpointing on disk. Objects are accessible through a traditional file interface, such that applications can benefit from in-memory storage without modifications. An additional API provides several concurrency control mechanisms for applications. These range from explicit push-based or streamoriented updates to transactions combined with optimistic synchronization. The consistency unit size is dynamically configurable for each object.
ECRAMs target application domains include MapReduce programs. MapReduce is a computing model that simplifies the design and implementation of parallel algorithms [2]. By using ECRAM, I/O-bound MapReduce applications benefit from the low-latency data access. Furthermore, extended MapReduce applications that process data iteratively [3] or online [1] take even more advantage from ECRAM, because its transactions allow for efficient synchronization of the dynamic distributed state. While the Phoenix system [7, 8] also provides a MapReduce framework for shared-memory machines, it does not target extended MapReduce applications and does not use transactions for dynamic state synchronization. We have described the ECRAM-based in-memory MapReduce in two papers, which have been accepted for inclusion in international conference programs [6, 5]. Furthermore, another paper describes the ECRAM-based in-memory file-system [4].
Our experiments in the HPI Future SOC Lab investigate ECRAMs performance on a multicore machine with huge main memory. ECRAM was developed targeting conventional compute clusters, but it can as well run as a multi-process application on a single machine over the loopback network interface. If our experiments indicate that the TCP/IP stack is a performance bottleneck on a single machine, ECRAMs networking code could be extended to UNIX domain sockets or other mechanisms for inter-process communication.

2. Used Future SOC Lab Resources

We are executing our experiments on a Hewlett Packard ProLiant DL980 G7 Server that is called DL980-2. The server is equipped with 8 Xeon Nehalem X6550 CPUs, each having 8 hyper-threaded cores. The CPU clock rates are 2 GHz, the L3 caches 18 MB large, and Turbo Boost is enabled. The DL980-2 has 128 GB of RAM. Given that our experiments run in main memory of the single machine, they do not use the 2 x 146 GB hard disks, neither the Emulex 4Gb Fibre Channel network card. The DL980-2 boots Ubuntu Server 10.10 from a local harddisk and mounts the home file-system from a NAS device.
We compare the measurements on Future SOC Lab with the results on our local cluster installation called BSCluster, which consists of 33 computing nodes, each equipped with 2 AMD Opteron processors (16 x Opteron 246 @ 2 GHz, 17 x Opteron 244 @ 1.8 GHz) and 2 GB ccNUMA RAM. Our local cluster boots Debian Squeeze with Linux x86-64 kernel 2.6.32 in diskless mode via NFS.

3. Findings

After becoming acquainted to the lab resources, we needed to make small changes to the compilation process to be able to build ECRAM on the Ubuntu installation on the DL980-2. We are finally able to build and run ECRAM applications, the ECRAM MapReduce framework and different applications.

Figure 1 displays the latency of raytracing a moderately complex scene to a medium and large image using the MapReduce framework on the DL980-2. Figure 2 compares the scalability of the raytracer on the

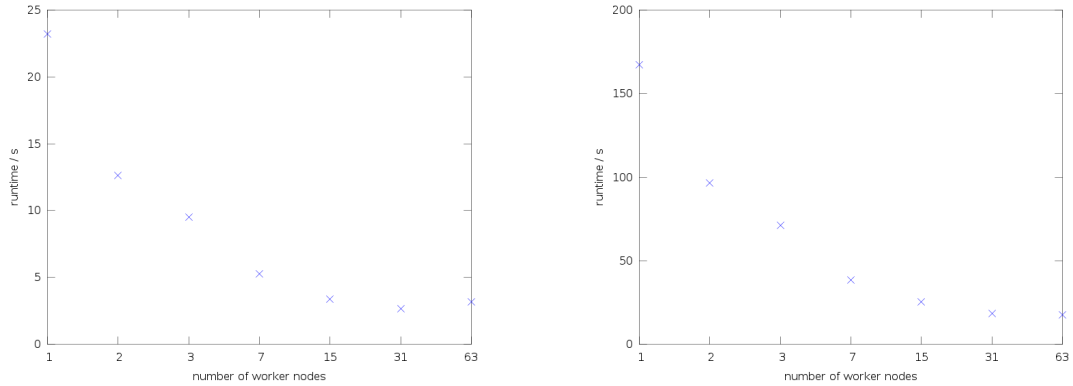


Figure 1. Run-time of ECRAM MapReduce raytracer on DL980-2, image size 4096x2048 pixels (left) and 8192x8192 pixels (right)

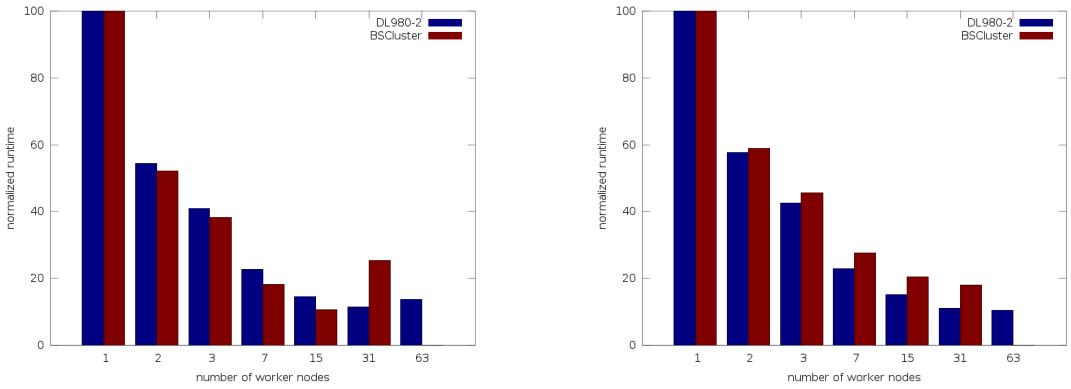


Figure 2. Scalability of ECRAM MapReduce raytracer on DL980-2 and BSCluster, image size 4096x2048 pixels (left) and 8192x8192 pixels (right)

DL980-2 and on our cluster.

These initial results show that ECRAM and the ECRAM MapReduce framework scale well not only on distributed systems, but also on multiprocessor machines. At large, HPI Future SOC Lab is a powerful platform for developing and evaluating parallel applications.

4. Next Steps

To continue our evaluation of in-memory storage on HPI Future SOC Labs multicore machines, we plan to run other parallel applications on the DL980-2. Besides the raytracer, there exist about 5 application for ECRAMbased MapReduce until now. We are currently extending ECRAM to provide persistency, adaptive replication and flexible conflict units. These features shall be evaluated on the HPI Future SOC Lab. If time and resources permit, we intend to evaluate our distributed in-memory file-system, which is based on ECRAM and FUSE.

References

[1] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation, NSDI'10*, page 2121, Berkeley, CA, USA, 2010. USENIX Association.

[2] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[3] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative MapReduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC'10*, page 810818, New York, NY, USA, 2010. ACM.

[4] K.-T. Rehmann, S. Dere, and M. Schöttner. Adaptive Metadata Management and Flexible Consistency in a Distributed In-memory File-System. In *Proceedings of the Twelfth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2011)*, pages 201–206, 2011.

[5] K.-T. Rehmann and M. Schöttner. An In-Memory Framework for Extended MapReduce . In *Proceedings of the Seventeenth IEEE International Conference on Parallel and Distributed Systems 2011 (ICPADS 2011)*, pages 17–24, Tainan, Taiwan, Dec. 2011. IEEE Computer Society.

[6] K.-T. Rehmann and M. Schöttner. Applications and evaluation of in-memory mapreduce. In *Third International Conference on Cloud Computing Technology and Science (CloudCom 2011)*, pages 67–74, Athens, Greece, 2011. IEEE Computer Society.

[7] J. Talbot, R. M. Yoo, and C. Kozyrakis. Phoenix++: modular MapReduce for shared-memory systems. In *Proceedings of the second international workshop on MapReduce and its applications, MapReduce '11*, page 916, New York, NY, USA, 2011. ACM.

[8] R. M. Yoo, A. Romano, and C. Kozyrakis. Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC '09)*, pages 198–207, Washington, DC, USA, 2009. IEEE Computer Society.

Performance Prediction for Main Memory Databases in Data Clouds

Jan Schaffner, Benjamin Eckart*, Hasso Plattner
Hasso Plattner Institute, University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
E-mail: {firstname.lastname}@hpi.uni-potsdam.de

Dean Jacobs
SAP AG
Dietmar-Hopp-Allee 16
69190 Walldorf, Germany
E-mail: dean.jacobs@sap.com

Abstract

In Software-as-a-Service, multiple tenants are typically consolidated into the same database instance to reduce costs. For analytics-as-a-service, in-memory column databases are especially suitable because they offer very short response times.
In this report, we develop a model for predicting whether the assignment of a particular tenant to a server in the cluster will lead to violations of response time goals. To do, the combined resource utilization incurred by assigning multiple currently active tenants onto a single server is captured using a regression model. While this model is fairly simple, we show its accuracy to be very high (i.e. the prediction error less than 10%).
A more detailed description of the presented performance model was first published by the authors in [11].

1. Introduction

Analyses over large data sets that require a high degree of inter-node parallelism (e.g. log file processing at Facebook [13]) have been a major focus of recent research on cloud computing. Cloud computing is, however, equally attractive when many relatively small data sets need to be handled. An important case here is Enterprise Software-as-a-Service (SaaS), where a service provider develops an enterprise application and operates the system that hosts it for many businesses. Enterprise SaaS solutions commonly maintain data in a farm of conventional databases. To reduce total cost of ownership, multiple businesses are consolidated into each database instance, a technique referred to as multi-tenancy [1].
In this report, we describe the following result: we develop a model for characterizing the load on an in-memory column database containing multiple tenants with different request rates. Previous cost models for in-memory databases are focused on characterizing the costs for individual queries [7, 6], while our model aims to characterize the database's ability to handle an on-going stream of queries within pre-defined response time goals. More specifically, our model predicts how much load a server can sustain before query response times in the 99th percentile exceed one second (which is our SLO).
Our approach to modeling database performance is experimental. Rather than characterizing all constituents of the system and their interactions, we extract the model from observations of a running system. As we will show, the resulting model is very accurate: our prediction of the response time in the 99th percentile is always within 10% of the actual value. The reason we are able to build such a robust model using relatively simple tools is that in-memory column databases behave very linearly. We will show that response times can to a large extent be derived from the number of bytes the database scans in a given time interval regardless of the distribution of sizes and requests rates of the tenants. The ability to accurately predict response times makes it possible to run servers at a higher utilization level, thereby decreasing costs.
Our study is based on SAP's in-memory column database TREX ([9, 10, 4]). To support SaaS, we developed a clustering infrastructure around TREX that replicates data for scalability and availability, supports multi-tenancy, and allows for dynamic cluster sizing. This work is part of a broader research project to adapt TREX to support on-demand data warehousing in a utility computing environment.
Note that there is considerable potential for consolidation through multi-tenancy even for in-memory column databases. As an example, we looked at one cube in the data warehouse of a large SAP customer, a Fortune 500 retail company. This cube contained all the sales records for three years and the fact table had approximately 360 million records. Using TREX's standard dictionary compression, this cube consumed only slightly more than 2 GB of memory, which is low given modern server hardware as present for example in the HPI Future SOC Lab. Moreover, SaaS often targets small to mid-sized businesses, which have orders of magnitude less data.

*now with LinQuire, mail@benjamin-eckart.de

This report is organized as follows: The next section introduces the TREX clustering infrastructure, called *Rock*, that is the basis for our experiments. Section 3 develops our basic model for predicting response times in the 99th percentile. Section 4 outlines next steps concludes this report.

2. The Rock Clustering Infrastructure

This section describes the Rock clustering infrastructure, which was used in the experiments in this report. It also describes the benchmark that was used for the experiments as well as all other relevant parts of our experimental setup.

2.1 The Rock Clustering Framework

The Rock clustering framework runs in front of a collection of TREX servers and provides multi-tenancy, replication of tenant data, and fault tolerance. Figure 1 illustrates the Rock architecture. Read requests are submitted to the cluster by an analytics application. Write requests are submitted by batch importers, which periodically pull incremental updates of the data from transactional source systems. The Rock framework consists of three types of processes: the *cluster leader*, *routers*, and *instance managers*. Each instance manager is paired one-to-one with a TREX server to which it forwards requests.

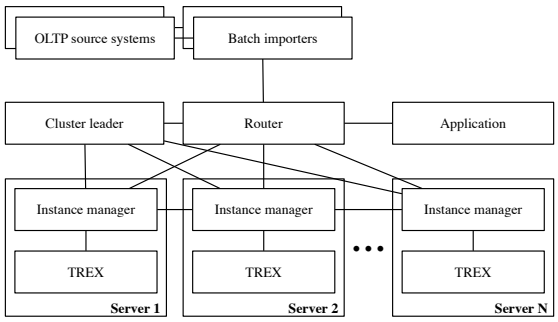


Figure 1. The Rock Analytic Cluster Architecture

The cluster leader exists only once in the landscape and assigns tenant data to instance managers. Each copy of a tenant's data is assigned to one instance manager and each instance manager is responsible for the data of multiple tenants. The cluster leader maintains the placement information in a *cluster map*, which it propagates to the routers and instance managers so all components share a consistent view of the landscape.
The routers accept requests from outside the cluster and forward them to the appropriate instance managers. If a tenant has multiple replicas, the router balances the load across them. The load on a server is taken to be the number of queries being processed at a given point in time (across all tenants on the server).
We assume there is a single batch importer per tenant and that writes are sequentially numbered[1]. They can therefore be sequentially applied at every server and there are never inconsistencies due to failures, lost messages or updates arriving multiple times. A router may forward a write request to any one of the instance managers for a tenant, which then propagates the write to the other instance managers for that tenant. Our target application requires consistent reads across multiple queries when drilling down into a data set, and all requests within a drill-down are issued for the same version number. Rock supports this by using multi-version concurrency control (MVCC) based on snapshot isolation [2], which TREX implements natively.
According to [3], multi tenancy can be realized in the database by adopting a *shared-machine, shared-process,* or *shared-table* approach. The shared-table approach, where each table has a `tenant_id` column, can be made efficient if accesses are index-based. However analytic queries on column databases generally entail table scans, and scan times are proportional to the number of rows in the table. Rock therefore uses the shared-process approach and gives each tenant their own private tables.

2.2 Experimental Setup

The experiments in this report are based on a modified version of the Star Schema Benchmark (SSB) [8], which is an adaptation of TPC-H.
To produce data for our experiments, we used the SSB data generator. The fact tables vary in size from 600,000 to 6,000,000 rows across tenants and the dimension tables increase linearly with the size of the fact tables. Using TREX's standard dictionary compression, the fully-compressed data sets in our experiments consume between 25 and 204 MB of memory.
While TPC-H has 22 independent data warehousing queries, SSB has four *query flights* with three to four queries each. A query flight models a drill-down, i.e. all queries compute the same aggregate measure but use different filter criteria on the dimensions. This structure models the exploratory interactions of users with business intelligence applications. We modified SSB so all queries within a flight are performed against the same consistent TREX snapshot.
In our benchmark, each tenant has multiple concurrent *users* that submit requests to the system. Each user cycles through the query flights, stepping through the queries in each flight. After receiving a response, a user waits for a fixed think time before submitting the next query. To prevent caravanning, each user is offset in the cycle by a random amount. For the experiments

[1] The batch importers need to maintain a consistent numbering scheme in the face of failure and recovery, which can be accomplished using algorithms such as Paxos [5].

presented in this report, a random think time drawn from a negative exponential distribution with a mean of 5 seconds was used.

The number of users per tenant is taken to be a relative size factor for that tenant times a system-wide *user scale factor*. Our experiments vary this scale factor to set the overall rate of requests to the system. Following [12], which studies web applications, we set the user think time to five seconds. This is perhaps too short for more complex applications, but the system behaves linearly in this respect: doubling the think time would double the maximum number of simultaneous users.

The batch importer for each tenant performs one write every five minutes. Each write makes a tenant's fact table grow by 0.05% of its size at the beginning of the run. Writes for different tenants occur at different times, so the overall amount of data in the system gradually increases. The execution times of writes was not measured because they are submitted by a batch process that is not visible to users.

The first ten minutes of each benchmark run were cut off to ensure that the system is warmed up. The next ten minutes after the warm-up are called the benchmark period. All queries submitted after the benchmark period were cut off as well. All experiments were run on *large memory* instances on Amazon EC2, which have 2 virtual compute units (i.e. CPU cores) with 7.5 GB RAM each. For disk storage, we used Amazon EBS volumes, which offer highly-available persistent storage for EC2 instances.

3. An Emprical Model for Response Time Prediction

Throughout this report, we use the term *workload* to refer to the actual amount of work a server receives[2]. The capacity of a server is reached when the workload becomes so high that response time goals are violated. We require that 99% of all queries have subsecond response times. The query with the highest response time among 99% of all queries with the lowest response time is called the *99th percentile value*.

We experimentally developed a model to determine how many tenants with different sizes and different request rates can be consolidated onto one server without violating the response time goal. Processing aggregation queries of larger tenants takes longer than processing aggregation queries of smaller tenants because more data needs to be scanned. We describe workload as a function of request rate and size for each tenant and show that it is related to the number of records that are processed in a given time interval.

To examine the maximum possible workload depending on request rate and tenant size, we conducted several tests using the Star Schema Benchmark.

[2]In particular, the term *workload* in this report does not refer to the queries in our benchmark.

3.1 Relation of Request Rate and Tenant Size

In this experiment, a single server was packed with a group of equally-sized tenants and the total number of users was distributed equally among all tenants. There were no writes. The benchmark was run multiple times using different tenant sizes (denoted as t_S) and different request rates (t_R). The same fixed amount of memory was used in all cases so the configuration contained more or less tenants depending on the chosen size. Figure 2 shows the maximum request rate per tenant that could be achieved without violating the response time goal.

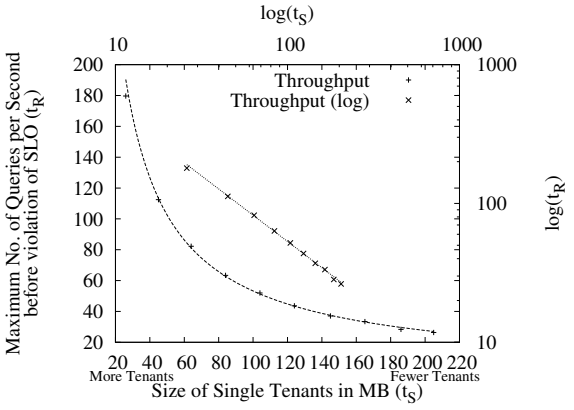


Figure 2. Maximum Request Rate before SLO Violation by Tenant Sizes

The maximum request rate decreases slightly exponentially in relation to the size of the individual tenants. Figure 2 shows the same dataset both on normal and logarithmic scales (note x2 and y2 axes). For the logarithmic plot, the shape of the curve is linear. The relation can thus be described as:

$$log(f(t_S)) = m \cdot log(t_S) + n \tag{1}$$

where the y-intercept $n = 3.617$ and the gradient $m = -0.945$ can be estimated using the *Least-Squares Method*. This equation can be rewritten as:

$$t_R = \frac{1}{t_S^{-m}} 10^n \tag{2}$$

Based on this equation, Equation (3) defines the workload incurred by a tenant as a function of its request rate and size.

$$w(t_R, t_S) := \frac{t_R t_S^{-m}}{10^n} \tag{3}$$

Our goal is to calculate the *workload* on a server given arbitrary values for size and request rate. Therefore, the function is normalized such that $w(t_R, t_S) = 1$

denotes the point where the 99th percentile value exceeds the response time goal of 1000 ms. Decreasing t_R or t_S would result in a function value smaller than 1 and, consequently, the server would not violate the response time goal.
This equation shows that, to a large extent, the workload corresponds to the number of bytes that are scanned in a given time interval regardless of the distribution of sizes and request rates of the tenants. The relation is not exactly linear (m is not exactly -1): there is a slight advantage to processing a larger number of smaller requests. Our response time goal is violated when 1% of all queries have a response time of more than 1 second. The likelihood of a query being slower than 1 second increases as tenants become larger given the distributions of response times we observed during our experiments. The processing overhead for individual queries is hidden behind this phenomenon.

3.2 Relation of Workload and 99th Percentile Value

Typically, the tenants on a server will have different sizes and request rates. Equation 4 defines the total workload on a server as the sum of the workloads over the set of all tenants $T = \{(t_S^1, t_R^1), \ldots, (t_S^n, t_R^n)\}$ on the server.

$$w(T) := \sum_{t \in T} \frac{t_R t_S^{-m}}{10^n} \tag{4}$$

To analyze the relation between workload and 99th percentile value, we tested four server configurations with a different total data set size ranging from 1.5 to 3.2 GB. In each configuration, the amount of data and the request rate for each tenant varied. Again, the experiments were conducted on a single server without writes. The results are shown in Figure 3.

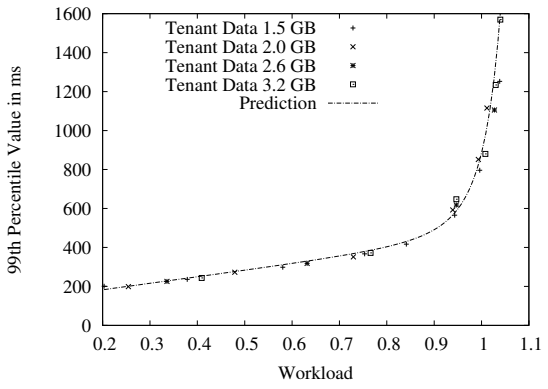


Figure 3. Capacity of a Single Instance (Read-only Queries)

The predicted workload in Figure 3 was calculated using Equation (4). Interestingly, the shape of the graph is the same even when varying the tenant mix on a server. Up to a workload of 0.9, the graph is increasing linearly. Afterwards, it is increasing exponentially. We assume that the 99th percentile starts to increase exponentially at the point where request rates are so high that queries start to queue up in the database. The graph can be described as a function shown in Equation (5) with the parameters $a = 333.982$, $b = 34.914$, $c = 2.537$, $d = 7$, and $e = 80.334$, which have been estimated using regression.

$$f(w) = aw + be^{cw^d} + e \tag{5}$$

As can be seen in Figure 3, the predicted 99th percentile value is always close to the measured 99th percentile value. This shows that the definition of workload as shown in Equation (4) is also valid for 99th percentile values other than the maximum allowed value of 1000 ms and for server configurations containing tenants with different sizes.

3.3 Accuracy of the Model

To determine the accuracy of the model, we split the measured data into a training set and a test set. The training data was used to generate the model while the test data was used only to validate the model. Figure 4 shows the estimated 99th percentile value using the training data in relation to the measured 99th percentile value using the test data in terms of a Q-Q Plot.

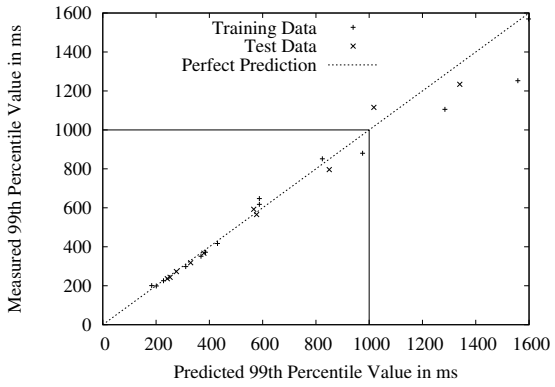


Figure 4. Q-Q Plot for Estimated and Measured 99th Percentile Value

Especially for low 99th percentile values, there is hardly any difference between the estimated and the predicted values. In the non-linear range of the function which starts at a 99th percentile value of approximately 400 ms (see Figure 3) the estimation is less precise. Predicting a value in the exponential range of the function is more difficult, because even small variations can have a strong impact. However, in the relevant range up to a 99th percentile value of 1000 ms, the variance is always less than 10%. In comparison to more sophisticated machine learning algorithms, regression is a rather simple technique but can be calcu-

lated quite fast and proved to be capable of reliably predicting the 99th percentile value in our case.

4. Conclusion

We developed a model for predicting the response time in the 99th percentile for an in-memory column database running a scan-intensive query workload. We showed how to use this model to predict whether a database instance will be able to meet a response time goal of 1 second in the 99th percentile given a particular assignment of tenants to this server. The parameters of the prediction model are the sizes and request rates of the tenants placed on a server, which corresponds to how many bytes an in-memory database instance needs to scan in a given interval. Our results show that the same 99th percentile value can be obtained for a set of tenants containing less data but high request rates and a setup with more data but lower request rates.
In the next Future SOC Lab period we plan to devise automatic placement algorithms on top of our model. The goal is to algorithmically respond to changes in the load situation of a cluster, which is driven by diurnal usage patterns.

References

[1] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1195–1206, 2008.

[2] H. Berenson, P. A. Bernstein, J. Gray, J. Melton, E. J. O'Neil, and P. E. O'Neil. A Critique of ANSI SQL Isolation Levels. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 1–10, 1995.

[3] D. Jacobs and S. Aulbach. Ruminations on Multi-Tenant Databases. In *Datenbanksysteme in Business, Technologie und Web (BTW 2007), 12. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Proceedings, 7.-9. März 2007, Aachen, Germany*, pages 514–521, 2007.

[4] B. Jaecksch, W. Lehner, and F. Faerber. A plan for OLAP. In *EDBT 2010, 13th International Conference on Extending Database Technology, Lausanne, Switzerland, March 22-26, 2010, Proceedings*, pages 681–686, 2010.

[5] L. Lamport. The Part-Time Parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[6] S. Listgarten and M.-A. Neimat. Modelling Costs for a MM-DBMS. In *RTDB*, pages 72–78, 1996.

[7] S. Manegold, P. A. Boncz, and M. L. Kersten. Generic Database Cost Models for Hierarchical Memory Systems. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 191–202, 2002.

[8] P. E. O'Neil, E. J. O'Neil, X. Chen, and S. Revilak. The Star Schema Benchmark and Augmented Fact Table Indexing. In *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, pages 237–252, 2009.

[9] 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 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 1–2, 2009.

[10] J. Schaffner, A. Bog, J. Krüger, and A. Zeier. A Hybrid Row-Column OLTP Database Architecture for Operational Reporting. In *Informal Proceedings of the Second International Workshop on Business Intelligence for the Real-Time Enterprise, BIRTE 2008, in conjunction with VLDB'08, August 24, 2008, Auckland, New Zealand*, 2008.

[11] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier. Predicting in-memory database performance for automating cluster management tasks. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 1264–1275, 2011.

[12] S. Sivasubramanian, G. Pierre, M. van Steen, and G. Alonso. Analysis of Caching and Replication Strategies for Web Applications. *IEEE Internet Computing*, 11(1):60–66, 2007.

[13] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. S. Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 1013–1020, 2010.

Downtime Analysis for Pro-Active Virtual Machine Migration

Felix Salfner
SAP Innovation Center Potsdam
August-Bebel-Str. 88
14482 Potsdam, Germany

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

Abstract

Live migration is a technique to move virtual machines from one physical host to another while they are continue to run. The HPI Future SOC Lab project "Towards an Architectural Pattern for Pro-Active Virtual Machine Migration" investigates live migration as a mean to handle imminent faults even before they result in a failure. However, this approach can only be used successfully if live migration meets certain requirements regarding the duration of the process. In this report, we present new experimental results about the factors determining the duration of live migration.

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. Systems for massively parallel processing (MPP) 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 MPP systems have shown a mean time between failures (MTBF) in the order of 6.5 to 40 hours, depending on installation maturity. Another example is the Google data center, which 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 systems such as blade centers, problems and challenges that once were of interest only to a small community of researchers and high-performance computing engineers will now seriously impact the computing environment of tomorrows average server environments.

One commonly agreed problem resulting from smaller structural sizes, extreme memory increase (as in the FutureSOC lab with 2TB machines) and dynamic frequency / voltage scaling in the CPU is the decreasing reliability of hardware components. Industry reacted on this upcoming challenge – which is already well-known in the MPP Exascale computing community with a set of new fault monitoring and fault tolerance solutions, which replace reactive fault tolerance schemes with their pro-active counterparts.

One interesting layer of reactive fault tolerance is the concept of a distributed virtualization-based failover cluster (see Figure 1). This approach can be utilized in addition to an existing set of solutions on hardware, firmware, operating system, middleware, and application level.

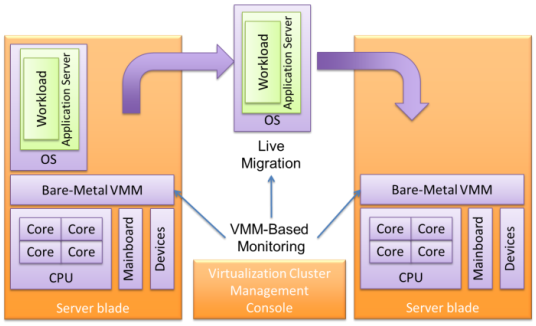


Figure 1. Reactive Live Migration of Virtual Machines

2. Approach

Live migration is a technique to move running virtual machines from one physical host to another, without disrupting running applications or the virtualized operating system. So far, live migration is primarily used for load balancing, server consolidation, for planned maintenance and for manual failure recovery.. However, reacting on partial failures that have already occurred leaves only a relatively small time window for error mitigation activities. Current approaches are also solely based on correctable error counter thresholds at single levels of the system stack (usually the VMM), and an according subsequent reaction.

Our FutureSOC lab project investigates an approach where live migration is used in a pro-active way to increase system dependability. Pro-active in this context means to act on the first symptoms, even before a problem has actually evolved into a more severe error state or failure. More specifically, we seek to preventively move running virtual machines away from failure-prone hosts.
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 hardware or software failures. Such anticipation requires the continuous monitoring and investigation of a systems state, in order to detect anomalies that indicate an upcoming failure.

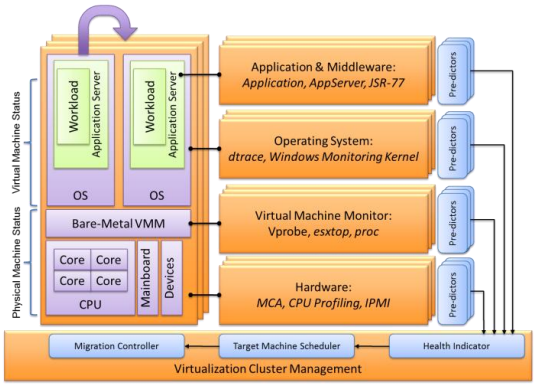


Figure 2. Pro-Active Virtual Machine Migration through Failure Prediction

One key concept is the integration of status assessments from all system levels, in order to utilize a maximum amount of system state information and domain knowledge. Until today, such an approach would have had a serious performance impact due to the monitoring and prediction computation overhead. However, the identified problem domain complex and powerful parallel hardware in server environments 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 into a new architecture for *anticipatory virtual machine migration*. The proposed 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, it is necessary to identify 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 process can be realized during a semi-automated training phase. Most prediction approaches anyway demand this training phase, in which a functional system is profiled for the "normal" pattern of monitored events. The prediction approach then puts this data into relation with the measured current system state at this time.
As the technique of live migration is already available in today's virtualization products, we focus in our work on properties of live migration rather than its technical implementation. More specifically, we investigated the factors that impact both the total duration of live migration as well as the downtime involved when switching from the source virtual machine to the destination. As we will show again in the following sections, the relationships are non-trivial and can affect migration times significantly.
It should also be noted that our investigation focus is on bare-metal virtualization only, since today's most capable live migration implementations are based on this model.
In our previous experiments, we have used three load generators to investigate how long it takes to perform live migration of a virtual machine. Specifically, we investigated two time intervals:

- *Total migration time* – The duration of the entire migration process.
- *Blackout time* – The length of the time interval when the virtual machine is not running during the live migration process.

In the phase of the HPI FutureSOC project summarized by this report, we have focused on a reality check of the previous results, i.e., we wanted to see whether the results obtained from using a load generator also hold for real world applications.
We have chosen two standard benchmark applications as foundation for this investigation. The reason why we did not use benchmarks in the first place is their lack of controllability: Our artificial load generators can be used to explore the entire parameter space, while application benchmarks are limited in their possibilities for configuration. By showing that our previous results match the benchmark results at significant points, we were able to reason about the validity for various settings and conditions.

3. Experiment Setup

Benchmarks typically come as combination of a benchmark server application and a load generator client. The client attempts to send as many requests to the server as possible, in order to determine how many requests the benchmark application can handle on a given system. In our case however, we are not interested in the performance result of the benchmark itself we only want to use the benchmark for simulating a standardized workload that comes close to a real-world scenario. Similar to our previous experiments, we are still interested in total migration

time as well as blackout time when performing live migration of a running virtual machine from one physical host to another one.
In comparison to our earlier work, running a benchmark application requires a slightly modified experiment setup (see Figure 3). While our previous experiments were relying on a memory load generator running inside of the migrated virtual machine, the benchmark load generator now has to run in a separate machine, in order to trigger the benchmark server situated in the migrated virtual machine. While the load generator client sends requests to the benchmark application, the virtual machine running the application is migrated from one host to another host. Total migration time and blackout time were obtained on the machine running the benchmark load generator.

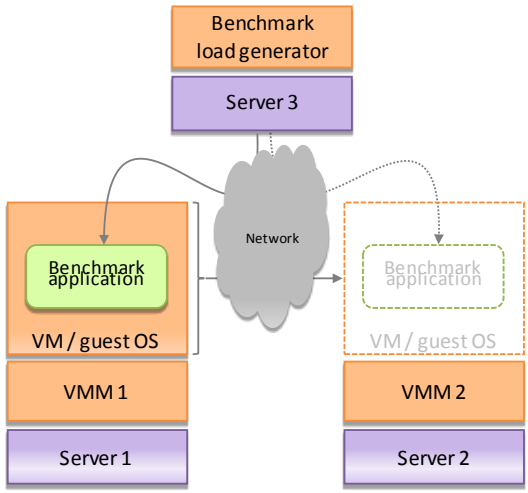


Figure 3. Experiment setup

4. Experiments and Results

In our previous experiments, we already found out that both migration time and blackout time are primarily influenced by three parameters:

- Size of the configured memory for the virtual machine
- Size of the working set of the virtual machine
- Rate at which memory pages are written (made "dirty")

The configured memory size of the virtual machine as well as the size of the working set can be easily obtained through standard mechanisms provided by the operating system. For the third parameter, there are no standard means to determine the rate at which the code running in a virtual machine (both application as well as guest operating system) modifies memory pages. Hence, we needed to develop a method for measuring the effective dirty page rate in a virtualized environment, which is described in the following section.

4.1. How to Measure the Dirty Page Rate

In our previous experiments, the dirty page rate was a configurable parameter of the load generator application being executed in the virtual machine. This approach no longer holds for a given benchmark application, since the dirty page load is an indirect effect of the benchmark code simulating real-world application behavior.
Our first approach was to leverage CPU hardware performance counters ("Performance Monitoring Units" PMU), which are available in the majority of modern processors. This solution would be able to form a hypervisor-agnostic solution. However, this approach first has the problem that performance counters are not passed through the hypervisor to the guest operating system, i.e it is not possible to access such hardware registers from inside a virtual machine. Nevertheless, since some virtualization solutions (such as KVM) map virtual machines onto operating system processes, the host operating system can inspect CPU performance counters for the specific process in this case, which makes the approach in principle viable.
We carried out multiple-stage experiments with the PMU idea in combination with the DPG load generator developed for earlier experiments. We first sought for candidate performance counters that increase linearly with increasing dirty page rate. In a second step, we used a modified version of the DPG that behaves identical to the unmodified version, but only reads the memory locations instead of writing. The idea here is that a performance counter that is a good measure for the dirty page rate grows linearly with the unmodified version of the DPG. At the same time it should remain constant for the modified version, or should show at least a significantly lower rate.
We identified about 20 counters with the described property. Up to here, we ran all tests with a constant page fill rate, i.e. the load generator always modified the same fraction of a page. In order to make the result more generalizable, we shifted our focus on finding a counter that is agnostic to the page fill rate. The according experiments resulted in only one counter that remains constant with different tested page fill rates: MEM_STORE_RETIRED:DTLB_MISS. This CPU hardware performance counter reports the number of retired stores that missed the DTLB.
In a second approach for determining the dirty page rate, we instrumented the KVM hypervisor to perform direct measurements of the parameter. The KVM kernel module has a built-in function to get a bit vector of

all pages being marked dirty since the last call. Based on this information, the number of dirty pages can be easily determined. Our modification took place in the qemu-kvm userspace application, in order to implement a function that allow to query for the current dirty page rate We modified the experiment setup so that the function was called once every second to determine the RATE parameter of the migrated virtual machine.
Our comparison of the two dirty page measurement approaches with the DPG as well as with the SPEC benchmark showed remarkable advantages of the second approach. This has several reasons. First, the hypervisor-based approach immediately reports the actual dirty page count, whereas the performance counter value needs to be translated into the number of dirty pages. Additional experiments showed that this mapping is clearly application-dependent, which contradicts our initial hypothesis that the counter reflects the dirty pages rate directly. Additionally, the hypervisor measurement has a much finer granularity and can be obtained directly for each virtual machine.

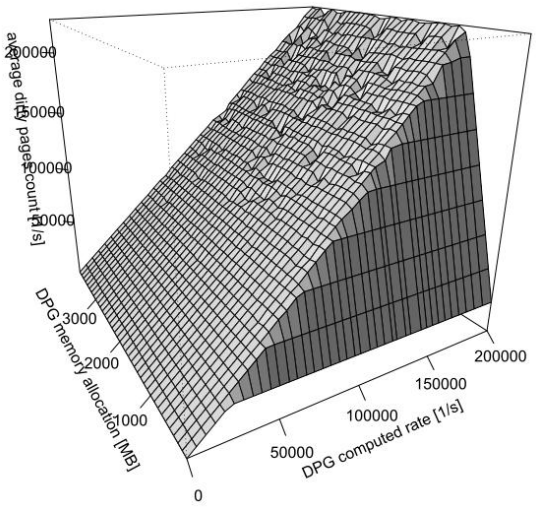


Figure 4. Comparison of measured dirty page rate (z axis) with the dirty page rate set as a parameter of the DPG load generator (x axis) for various sizes of the working set (y axis)

4.2. Investigating the Dirty Page Generator

To further evaluate the correctness of our measurement approach, we compared the dirty page rate that has been set as a parameter of the DPG load generator (DPG RATE) to the dirty page rate measured using the direct hypervisor measurement approach.
Our experimental results are shown in Figure 4. The measured dirty page rate matches the configured DPG RATE within the intended confidence interval, except for a small constant offset which is reasoned by memory operations of the remaining processes in the virtual machine.
A second interesting observation is that for small sizes of the working set, the measured dirty page rate stays constant with increasing values for the DPG dirty page rate. This is reasoned by the design of the DPG tool., which cycles over the working set memory pages with a specified access rate. If the working set size is too small, the implementation cycles the buffer several times in one round. Since we sample the dirty page rate with a fixed frequency (one second), it happens that DPG RATE memory pages get written several times between two calls of the KVM function. In this case, the modification of the same page for several times just leaves the page dirty, so there is no further increase in the monitored dirty page rate.

4.3. SPECjAppServer 2004 Benchmark

Virtualization and live migration are approaches that are primarily used in data center and cloud computing scenarios. In our experiments, we applied benchmarks that aim at resembling the workload found in such scenarios.
The first benchmark we used was the *SPECjAppServer 2004 version 1.08* application. It measures the performance of *Java 2 Enterprise Edition (J2EE)* technology-based application servers. The benchmark simulates a manufacturing process, supply chain management process, and a order/inventory business process [1].
In addition to typical parameters specifying the configuration of the server (number of threads, etc.), the *jAppServer* benchmark has one parameter called `txrate` to control the load applied to the system. We varied this parameter for bringing the system under test into various load conditions, then migrated the virtual machine in which the application server was running, and monitored total migration time as well as blackout time accordingly. The results are shown in Figure 5.
In the figure, we plotted migration time and blackout time respectively as a function of working set size and the measured dirty page rate. Each color in the plot corresponds to one setting for the `txrate` parameter. It might appear surprising that there are multiple stems in the plot for one setting of `txrate`. The reason for this behavior are effects within the application server and the guest operating system (such as caching) that let the size of the working set increase over time, even though the *jAppServer* workload was constant. A first analysis shows that both migration time as well as blackout time measurements resemble the behavior that could be measured with the DPG load generator. A more fine-grained analysis, including an analysis of the relative error for the different methods, is subject

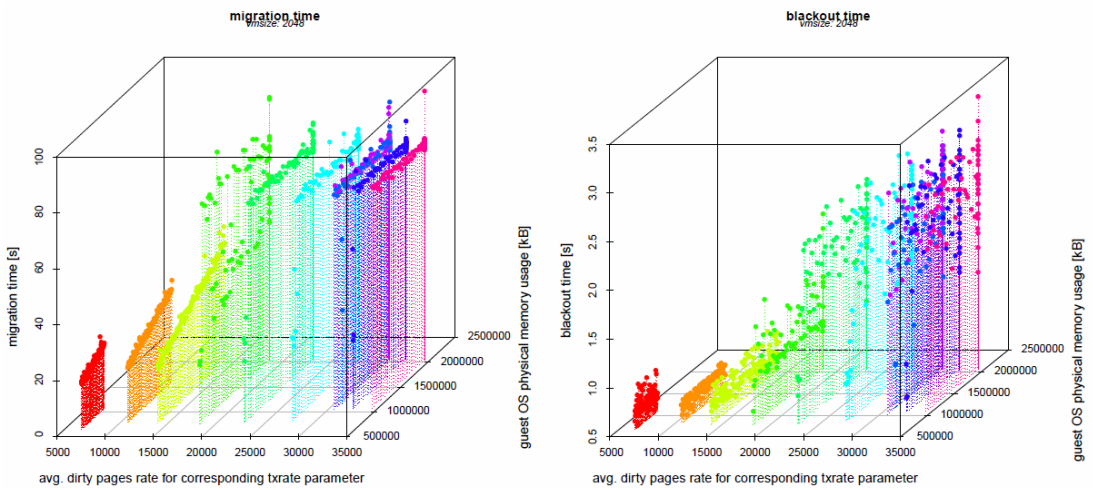


Figure 5. Migration time and blackout time plotted as stems for various settings of the SPECjApp txrate parameter (color coding). Each stem is located according to the measured average dirty page rate and working set size

of future work.

4.4. Results for the Mail Benchmark

Another typical scenario in data center environments is the hosting of mail servers. We used the Postal SMTP benchmark to generate SMTP requests for the widely used server-side combination of a UNIX / Linux Postfix SMTP server and a SpamAssassin spam filter. The postal benchmark has a variety of parameters controlling the SMTP load, such as maximum and minimum message size, the number of messages (emails) per minute and the number of connections that should be opened to the SMTP server. We varied three parameters and investigated their influence on the average dirty page rate. For each measurement, the other parameters have been set to a fixed value:

- Number of messages per minute
- Number of threads (connections to the server)
- Message size

The results for the Postal experiment runs are given in Figure 6 to Figure 8. From the experimental results, we first conclude that the number of messages sent per minute is the Postal configuration parameter with the strongest and most predictable impact on the measured dirty page rate.
Based on the initial results, we performed a series of live migration experiments where the virtual machine running the SMTP server and the SpamAssassin spam filter were migrated from one physical host to another, while measuring the according total migration time and blackout time under load. Based on the initial outcome, we have varied the number of messages per

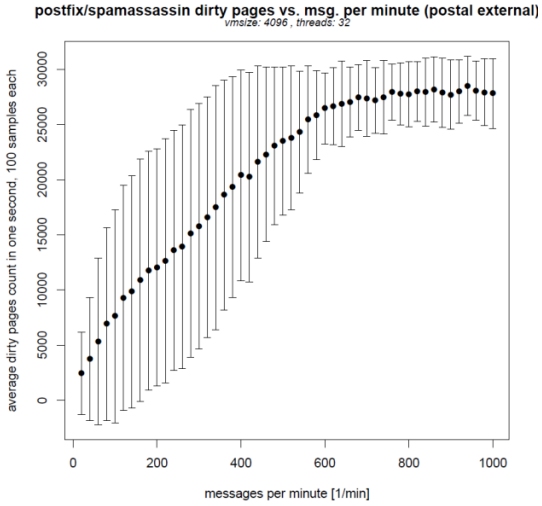


Figure 6. Measured dirty page rate as a function of the messages sent to the SMTP server per minute. Other parameters: Number of threads: 1, max. message size: 10kB

minute while using 32 threads and a maximum message size of 10kB.
The results are shown in Figure 9. Similar to the *jAppServer* benchmark, the figures for migration time as well as blackout time match previous results. A more fine-grained analysis of relative errors is currently under way.

5. Conclusions and Future Work

In continuation of our work on the duration of live migration in virtualized data center and cloud computing

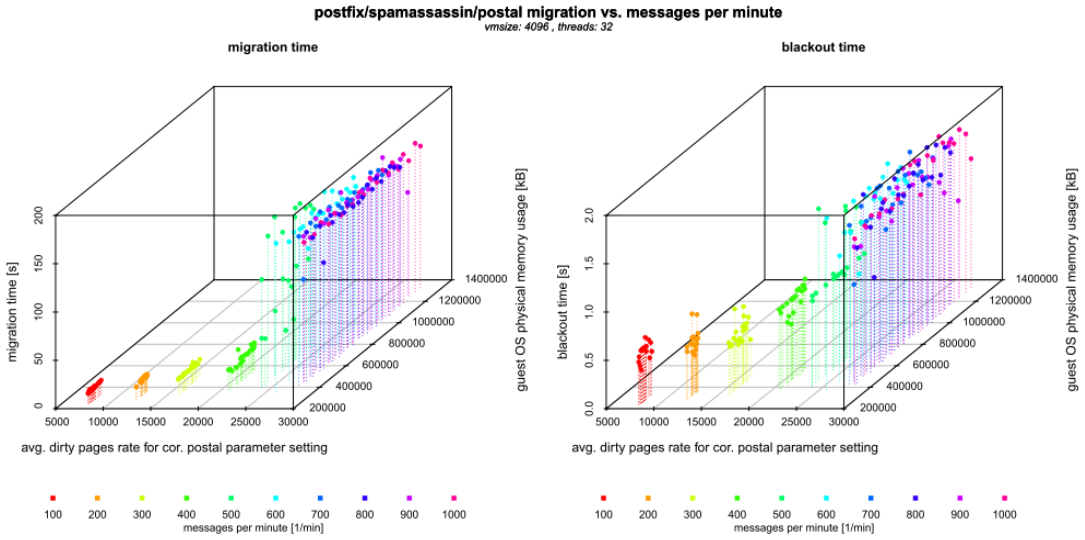


Figure 9. Migration time and blackout time for the mail server benchmark. The plot shows average migration times as a function of the number of messages sent to the server per minute. Error bars indicate standard deviation.

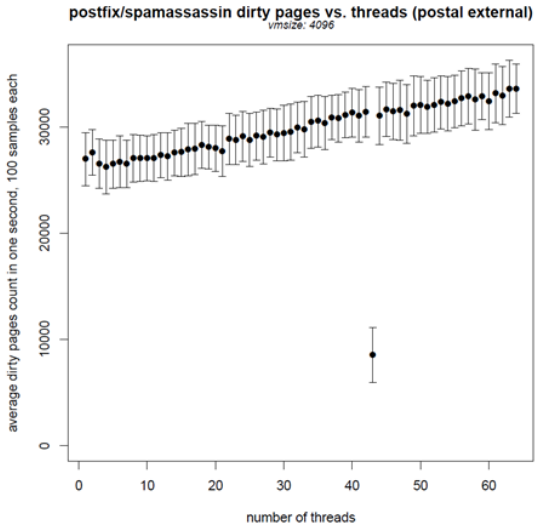


Figure 7. Measured dirty page rate as a function of the number of threads (connections to the SMTP server). Other parameters: Max. message size: 10kB, 110 messages per minute.

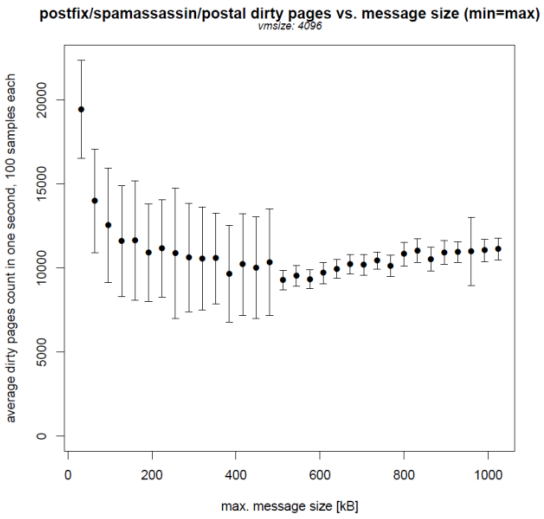


Figure 8. Measured dirty page rate as a function of maximum message size. Other parameters: 110 messages per minute, number of threads: 1

environments, we have recently focused on applying standardized benchmark workloads to check the applicability of our previous results, which were based on an artificial load generator. More specifically, we have used the SPECjAppServer 2004 benchmark and the Postal SMTP benchmark for this purpose. Results indicate that our previous findings match well with real-world workloads. Nevertheless, such a statement needs to be supported by further investigations which are planned for a subsequent phase of the project.

References

[1] SPECjAppServer2004 Design Document. http://www.spec.org/jAppServer2004/docs/DesignDocument.html, Sept. 2011.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
69	978-3-86956-229-2	**Akzeptanz und Nutzerfreundlichkeit der AusweisApp: Eine qualitative Untersuchung**	Susanne Asheuer, Joy Belgassem, Wiete Eichorn, Rio Leipold, Lucas Licht, Christoph Meinel, Anne Schanz, Maxim Schnjakin
68	978-3-86956-225-4	**Fünfter Deutscher IPv6 Gipfel 2012**	Christoph Meinel, Harald Sack (Hrsg.)
67	978-3-86956-228-5	**Cache Conscious Column Organization in In-Memory Column Stores**	David Schalb, Jens Krüger, Hasso Plattner
66	978-3-86956-227-8	**Model-Driven Engineering of Adaptation Engines for Self-Adaptive Software**	Thomas Vogel, Holger Giese
65	978-3-86956-226-1	**Scalable Compatibility for Embedded Real-Time components via Language Progressive Timed Automata**	Stefan Neumann, Holger Giese
64	978-3-86956-217-9	**Cyber-Physical Systems with Dynamic Structure: Towards Modeling and Verification of Inductive Invariants**	Basil Becker, Holger Giese
63	978-3-86956-204-9	**Theories and Intricacies of Information Security Problems**	Anne V. D. M. Kayem, Christoph Meinel (Eds.)
62	978-3-86956-212-4	**Covering or Complete? Discovering Conditional Inclusion Dependencies**	Jana Bauckmann, Ziawasch Abedjan, Ulf Leser, Heiko Müller, Felix Naumann
61	978-3-86956-194-3	**Vierter Deutscher IPv6 Gipfel 2011**	Christoph Meinel, Harald Sack (Hrsg.)
60	978-3-86956-201-8	**Understanding Cryptic Schemata in Large Extract-Transform-Load Systems**	Alexander Albrecht, Felix Naumann
59	978-3-86956-193-6	**The JCop Language Specification**	Malte Appeltauer, Robert Hirschfeld
58	978-3-86956-192-9	**MDE Settings in SAP: A Descriptive Field Study**	Regina Hebig, Holger Giese
57	978-3-86956-191-2	**Industrial Case Study on the Integration of SysML and AUTOSAR with Triple Graph Grammars**	Holger Giese, Stephan Hildebrandt, Stefan Neumann, Sebastian Wätzoldt
56	978-3-86956-171-4	**Quantitative Modeling and Analysis of Service-Oriented Real-Time Systems using Interval Probabilistic Timed Automata**	Christian Krause, Holger Giese
55	978-3-86956-169-1	**Proceedings of the 4th Many-core Applications Research Community (MARC) Symposium**	Peter Tröger, Andreas Polze (Eds.)
54	978-3-86956-158-5	**An Abstraction for Version Control Systems**	Matthias Kleine, Robert Hirschfeld, Gilad Bracha
53	978-3-86956-160-8	**Web-based Development in the Lively Kernel**	Jens Lincke, Robert Hirschfeld (Eds.)

ISBN 978-3-86956-230-8
ISSN 1613-5652