Monitoring Thread-Related Resource Demands of a Multi-Tenant In-Memory Database in a Cloud Environment

Dominik Paluch
Chair for Information Systems
Technical University of Munich
Garching, Germany
dominik.paluch@in.tum.de

Johannes Rank
Chair for Information Systems
Technical University of Munich
Garching, Germany
johannes.rank@in.tum.de

Harald Kienegger
Chair for Information Systems
Technical University of Munich
Garching, Germany
harald.kienegger@in.tum.de

Helmut Krcmar
Chair for Information Systems
Technical University of Munich
Garching, Germany
krcmar@in.tum.de

ABSTRACT

Estimating the resource demand of a highly configurable software system like an in-memory database is a difficult task. Many factors such as the workload, flexible resource allocation, multi-tenancy and various configuration settings influence the actual performance behavior of such systems. Cloud providers offering Database-as-a-Service applications need to monitor and apply these factors in order to utilize their systems in an efficient and cost-effective manner. However, only observing the CPU utilization of the database's processes, as done by traditional performance approaches, is not sufficient to accomplish this task. This is especially relevant for environments with multiple active database tenants, which adds another level of complexity to the thread handling on multiple layers like the database management system or the operating system. In this paper, we propose a fine-grained monitoring setup allowing us to analyze the performance of virtualized multi-tenant databases. Our focus is on extensively collecting and analyzing performance data on a thread level. We utilize this setup to show the performance influence of varying database configuration settings, different workload characteristics, multi-tenancy and virtualization features. Therefore, we conducted several experiments by applying the TPC-H benchmark generating OLAP workload on a SAP HANA database in a virtualized environment. In our experiments, we can show a strong dependency between the specific type of workload and the performance. Furthermore, we analyze the workload-dependent performance improvements and the performance degradation when changing the runtime configuration.

CCS CONCEPTS

• **General and reference → Measurement**; **Performance**; • **Information systems → Main memory engines**; **Database performance evaluation**;

KEYWORDS

In-memory Database; Performance Analysis; Cloud Computing; Multi-Tenancy; SAP HANA

1 INTRODUCTION

Cloud-based services are getting more and more attractive to enterprises by offering technical and financial advantages to in-house data centers. Furthermore, the demand for database services in the cloud, also defined as Database-as-a-Service (DaaS), has been increasing during recent years. Growing density of memory chips combined with decreasing prices permit the operation of databases holding an enterprise applications complete operational data set in memory. These so-called in-memory databases have advantages over traditional disk-based databases regarding performance when performing big data analytics or processing analytical data in general. For this reason, they are optimally suited for online analytical processing (OLAP) workload. In these cases, parallel computing methods are used to optimize the performance of OLAP workload resulting in the generation of multiple parallel threads [3].

Utilizing virtualization and multi-tenancy features allows cloud providers to decrease their operational costs, such as hardware-, energy- and software licensing. Furthermore, these features help them to reduce administration efforts in order to provide their services in a cost-efficient and scalable fashion [1]. However, it is crucial to understand the impact of different influence factors and setups on the performance behavior of in-memory databases when utilizing these features [2]. Recent work has shown that the efficient usage of threads is an important performance aspect for all in-memory databases that processes OLAP workload [7, 11]. Current research either puts a focus on the operation of a multi-tenant database system itself [9, 12] or puts a more generic focus on the underlying virtualization layer [4, 10]. However, cloud providers are often utilizing both concepts. They use virtualization in order to increase the efficiency and improve the flexibility of their hardware usage and multi-tenancy features on the database layer to allow further reductions in the maintenance effort and in the overhead of multiple virtual machines (VMs). Thus, it is expedient to consider not only the performance of a multi-tenant database or the virtualization layer, but considering both concepts.

Furthermore, cloud providers often utilize only high level monitoring, which does not allow to specifically monitor the utilization of threads through a process. In our work, we could notice a a fully utilized CPU even in scenarios with less intense workload. Thus,

31st GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 11.06.2019-14.06.2019, Saarburg, Germany.

monitoring only the CPU utilization is not enough to draw conclusions on the current workload intensity. In order to identify critical workload scenarios, a more fine-grained monitoring is necessary to capture all performance-relevant aspects.

This article is an extension of [11] and addresses the challenge to examine the performance behavior of the in-memory database SAP HANA in the context of a cloud provider's virtualized environment by considering various configuration changes. These include changes regarding the database system, the virtualization layer, the workload parametrization and the multi-tenancy features. Furthermore, a focus is set on thread efficiency taking into account the high parallelization when processing OLAP workload. Thus, following aspects were considered when designing our setup:

Workload Characteristics Regarding the workload, we consider the workload intensity and the definition of a user in the context of our benchmark setup. In this paper, we extend previous work by varying the type and intensity of the workload.

Multi-Tenancy We measure the threading characteristics of the single statements to get additional insights into threading efficiency. We also set various limits of concurrently active threads in the HANA database to show the performance impact on response time of the individual statements. In addition, we vary the number of concurrently active tenant databases to consider multi-tenant scenarios as well as single-tenant scenarios.

Virtualization Aspects When considering threading, it is important to include aspects of the virtualization layer. We consider the dynamic assignment of processing resources to a VM running a multi-tenant in-memory database. Since the simultaneous multithreading (SMT) technology has an influence on the processing time of threads [5], we also perform benchmarks to quantify this impact.

2 METHODOLOGY

2.1 Hardware and Software Setup

We used the established OLAP benchmark suite TPC-H on a SAP HANA database for our experiments [13]. To conduct the benchmarks, we used HANA version 2.0 SP 2 in a multi-tenant configuration. In total, five tenants have been configured. We filled the created database tenants with data sets created with a scale-factor of 30 as proposed by the benchmark guidelines. This resulted in tenant sizes of 30 GB each. To avoid unwanted performance interferences between the tenants, we created individual data sets for each tenant.

We chose SUSE Linux Enterprise Server (SLES) 12 SP2 as the underlying operating system. Our experiments were conducted on two VMs on an IBM Power E870 server with four CPU sockets populated with Power8 CPUs. In total, the server offered 40 physical CPU cores operating at a clock frequency of 4.19 GHz. To operate the VMs, the server utilized the firmware-based hypervisor platform IBM PowerVM. In this context, the VMs are also referred to as logical partitions (LPARs). The server was equipped with 4096 GB RAM. We assigned 256 GB RAM to the LPAR running the HANA database. Furthermore, we utilized different CPU assignments and varied between two CPU cores and four CPU cores. Through the utilization of the IBM tool *ppc64_cpu* we modified the configuration of the SLES operating system and varied the SMT configuration. We conducted benchmark runs with SMT turned off, SMT-2, SMT-4 and SMT-8. The different SMT configurations denote a hardware-based multithreading feature of the Power8 CPUs. In this context, the digit of the identifier equals the number of threads per core, i.e. SMT-4 equals four threads per core. In order to minimize the performance impact on the database through our benchmark-setup, we utilized another LPAR on the same server for our benchmark driver. In this setup, both LPARs are connected via a virtual switch, which allowed us to exclude any network-related performance impact, since this aspect is not included in the scope of this paper. In order to perform the benchmarks, we utilized customized shell scripts as the benchmark driver.

Table 1: Layers of our monitoring solution

Layer	Description
l_1	The resource consumption of the *jobworker* processes
l_2	The resource consumption of the *indexserver* processes of the individual tenant databases
l_3	The total resource consumption of the LPAR

2.2 Experimental Design

In previous work, we could show that the performance of database tenants is strongly dependent on thread-related parameters [11]. These parameters have a major impact on the performance behavior. Thus, we created our experimental design with the objective to extensively collect thread-related metrics influencing the CPU-related resource demand.

2.2.1 Monitoring setup. Our monitoring setup aims to collect fine-grained information about the thread-usage of the HANA database. We utilized the virtual file system */proc* to collect performance data regarding relevant database processes and their thread utilization. The database processes an OLAP query via the so-called *indexserver* process, that is part of every database tenant. In a first step, the *indexserver* process invokes an SQL executor thread, which performs query optimizations, prepares the execution plan and identifies the query as an OLAP query. After the query has been identified as a complex OLAP query, it is delegated to the *job executor* thread. For parallel processing, the *job executor* assigns the query to multiple idle threads from a predefined thread pool. These threads are utilized as *jobworker* threads. Based on these processing steps, we decided to put a focus on the three layers described in Table 1 when processing the raw data from our script-based monitoring solution. Monitoring l_1 allowed us to achieve fine-grained insights about the thread-usage while processing OLAP queries. The *jobworker* threads consume most of the system resources when processing an OLAP query. Thus, we have put a clear focus on analyzing the resource usage of these *jobworker* threads. However, in order to consider the resource consumption of other threads, we decided to analyze the monitoring data from l_2 and l_3 in addition.

After conducting our benchmarks, we noticed an identical memory usage pattern in the database's various tenants. Furthermore, an in-memory database is generally designed to keep its operational data set in memory. Thus, we decided to exclude memory usage from our work. Consequently, we put a focus on CPU utilization in our resource demand analysis. Thus, we decided to extract the following performance-relevant metrics from our raw data:

query: Each TPC-H query reflects individual performance relevant aspects of the database processing OLAP workload due to differences in the execution plan. Thus, each query has an individual performance behavior and needs to be monitored separately.

response time: We utilized this metric as the main indicator for the performance behavior of an individual query. We defined this metric as the time from which the query was sent to the database until a result has been returned.

processing time: In addition to the response time, we also considered the actual processing time of a query.

active jobworker threads: We noticed the *jobworker* threads were most relevant for processing an OLAP query. Thus, we counted the number of *jobworker* threads, which have been involved when processing a query.

rs ratio: To get further insights about the CPU utilization of the *jobworker* threads, we also analyzed the ratio between the total count of *jobworker* threads in the status running and those in the status sleeping.

rs jumps: In our analysis, we also counted the number of times the *jobworker* threads changed their status from running to sleeping. This is important, since it allows us to draw conclusions about the thread-utilization of a query.

jw cpu: Furthermore, we considered the total CPU time consumed by the *jobworker* threads.

total cpu: We included this metric to compare the CPU time consumed by the *jobworker* threads with the CPU time consumed by the whole system to ensure that no major interference by another thread has occured.

cpu jumps: Threads get assigned to different CPU cores by the operating system. However, the utilization of different cores by one thread increases the processing time of the thread. Thus, we counted the number of times the OS assigned a *jobworker* thread to a different CPU core.

context switches: This metric is only available for the whole system. It describes the switching of the CPU from one thread to another. The system has to perform multiple time-consuming steps, when performing a context switch. Thus, this metric indicates a negative impact on the performance.

2.2.2 *Benchmark design.*

We designed our experiments in order to get fine-grained information about the thread usage of the database tenants in the virtualized environment of a cloud provider. The parameters of the experiment are listed in Table 2. In our first benchmark run, we aimed at getting fine-grained information about the performance behavior of the individual TPC-H queries. We assigned two CPU cores to our database LPAR for the first benchmark run and executed all 22 queries consecutively on a single tenant. Since we wanted to exclude caching effects from our results, we executed the queries as regular, non-prepared statements. To avoid any interferences between the queries, we configured a waiting time of 10 seconds after each query execution. Furthermore, we were interested in monitoring data with a very high resolution in order to consider all relevant processing phases. Thus, we configured our monitoring setup to collect data at intervals of 0.0079 seconds on average. Lower monitoring resolutions would result in a lack of accuracy and would not allow us to identify the exact resource demand for each query. This aspect is especially relevant for the queries, which have only a short runtime. In later benchmark runs, we decided to decrease the monitoring resolution to 0.179 seconds to reduce the size of our raw data sets. In scenarios with higher loads the query runtime increased, which allowed us to decrease the monitoring resolution but still collect enough data to allow us a fine-grained analysis.

Table 2: Parameters of the experiment

Layer	Description
x_1	Single Query User / Multi Query User
x_2	Number of active Users
x_3	Number of active Tenants
x_4	Thread-Limit On / Off
x_5	Number of assigned CPU cores
x_6	Number of Threads / Core (SMT)

In order to show the influence of workload characteristics on the performance behavior, we decided to compare benchmark runs in single-tenant scenarios with those in multi-tenant scenarios. In [11], we could show that performance differences between single- and multi-tenancy scenarios exist. In addition, we could show the largest performance differences occur in scenarios with high load. However, we demonstrated the performance impact being much smaller when we additionally increased the number of active tenants in our multi-tenant setup. Thus, we decided to compare only the performance of scenarios with five active tenants to the performance with only one active tenant in this paper. This helped us to limit the number of long running benchmark runs. We varied the number of users in the performed benchmark runs in both scenarios. For a low load scenario we utilized 5 concurrent users, for a medium load scenario we utilized 20 concurrent users and for a high load scenario we utilized 50 concurrent users. For further performance insights, we decided to vary parameter x_1. Thus, we utilized two different user definitions. With the first definition, we did not conduct the TPC-H benchmark as intended. We chose to run each TPC-H query isolated and defined the user as the number of concurrently active executions of the same query. In the second definition, we defined the TPC-H user as intended. Thus, each user represented a different set of queries in in a specified sequence, which was unique for each user. Summarizing, we varied parameter x_1, x_2 and x_3 in this second set of benchmark runs.

SAP HANA is a highly configurable software system. Hence, it offers various parameters to optimize the performance of the database. The parameter *max_concurrency* limits the maximum number of *jobworker* threads a database tenant can utilize. SAP recommends setting the parameter to a value equal to the number of available CPU cores divided by the number of tenant databases. In this third set of benchmark runs, we varied parameters x_2, x_3 and x_4. We set parameter x_4 to either three, which limited the number of *jobworker* threads or no value, which did not set any limit.

Furthermore, we set parameter x_1 to the user definition according to the TPC-H benchmark for this run and all following runs.

In the fourth set of benchmark runs, we aimed to analyze the performance behavior in a virtualized cloud environment. In this setup, it is possible for administrators to assign more CPU resources to a VM dynamically. For this reason, we increased the CPU assignment from two to four CPUs during this benchmark run. It is also possible to migrate the VM to a server with a different CPU, which for example offers different capabilities regarding SMT. Thus, we considered the performance-impact of simultaneous-multithreading in this paper. Summarizing, we decided to vary the parameters x_2, x_3, x_5 and x_6. Parameter x_4 has not been set during these runs to avoid performance restrictions.

3 RESULTS

3.1 Analyzing the resource demand of the individual TPC-H queries

In this section, we analyze the thread-related resource demand of the individual TPC-H queries utilizing our collected monitoring data. Figure 1 shows the thread-related resource demand through the previously described performance metrics in a single tenant environment. We analyzed each query individually, formed groups and pointed out any anomalies.

grp1 (Query 1, 18): Both queries stand out due to their high response time. Furthermore, both queries are rather CPU intensive. The status of the utilized *jobworker* threads are comparatively rarely set from running to sleeping, which enhances the efficiency. These threads are also rarely assigned to a different CPU core. The number of context switches performed during the execution of these queries is also comparatively low. However, query 18 utilizes a much higher number of *jobworker* threads indicating a better parallelizability.

grp2 (Query 9, 13, 21): The resource utilization of these queries is similar to the previous set of queries. In this case, all queries utilize a high number of *jobworker* threads. In addition, the processing phase of these threads is interrupted only rarely through sleeping phases. The number of context switches is higher than in the previous set.

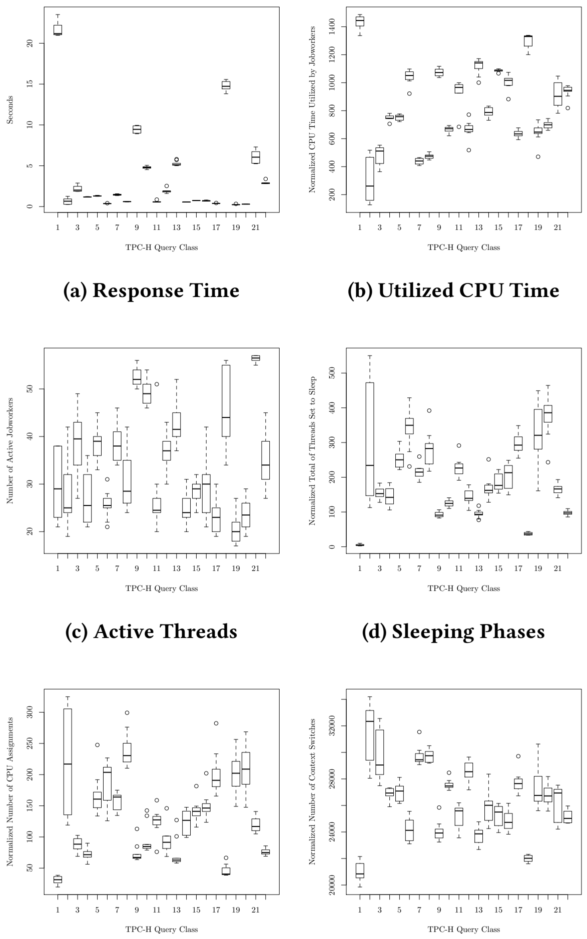
grp3 (Query 15, 16, 22): The major differences to the resource demand of the previous query set are the low response times. The number of sleeping phases and the number of context switches are rising to a value slightly below the average value.

grp4 (Query 6, 13): In contrast to the previous set, these queries show a much lower utilization of *jobworker* threads. The number of sleeping phases also increases.

grp5 (Query 4, 14, 17, 19, 20): Compared to the previous set, these queries show only an average CPU utilization. All queries only utilize a low number of *jobworker* threads. Except for the queries 4 and 14, the processing phases are often interrupted through sleeping phases. This also results in a higher number of different assignments to the CPU cores.

grp6 (Query 5, 10, 12): The resource demand of these queries is very similar to the previous query set. However, they utilize a higher number of *jobworker* threads and are less often assigned to different CPU cores.

grp7 (Query 2, 3, 7, 8): These queries show the lowest utilization of the CPU. In this case, this also results in a high number of context switches. Query 2 shows a very high variance regarding the assignment of CPU cores, the number of sleeping phases and the CPU time.



(a) Response Time

(b) Utilized CPU Time

(c) Active Threads

(d) Sleeping Phases

(e) CPU Assignments

(f) Context Switches

Figure 1: Resource demand

In most cases, the response time is very close to the actual processing time in this scenario with only one active user. However, query 2, 3, 12 and 21 show differences between these times. In conclusion, our fine-grained monitoring solution allows cloud providers to examine the resource demand of the specific workload in detail.

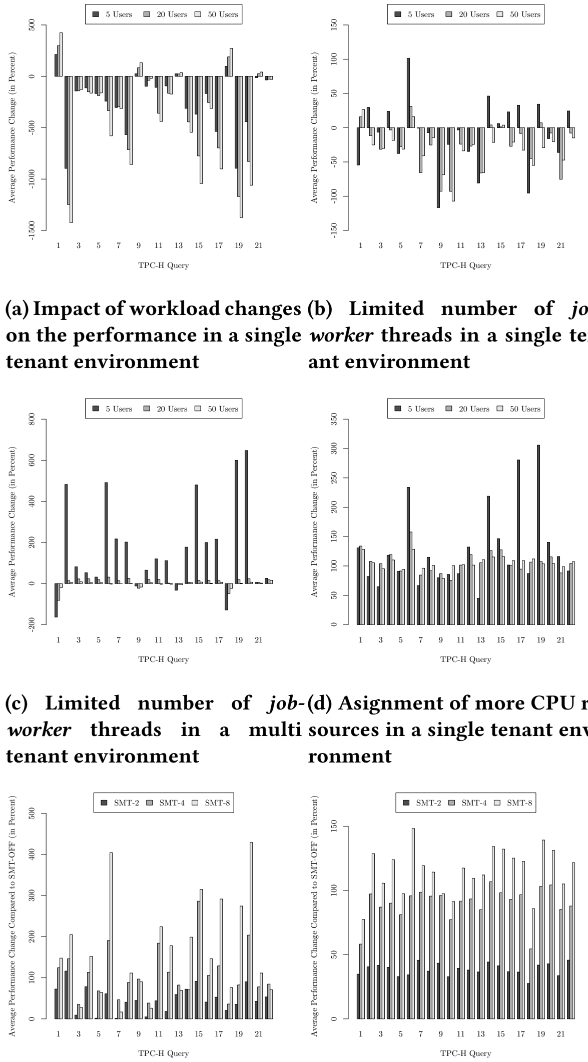
3.2 Performance behavior in different workload scenarios

In this section, we analyze the thread-related resource demand of the individual queries when changing the workload scenarios. Figure 2a shows the effects of workload changes on the query performance when a user is running all queries in a predefined sequence compared to the repeated execution of only a single query. It is noticable, that only CPU intensive queries (i.e. *grp1 and 2*) can benefit from the new workload scenario. However, a low CPU utilization does not necessarily result in a large loss of performance as for example query 3 shows. The number of sleeping phases also affects the performance. In most cases, the effect is stronger in high load scenarios with 50 active users. In multi-tenant environments, the effect is also stronger than in single-tenant environments. For CPU intensive queries, changing the user definition results in a decreased probability for the CPU being blocked by another CPU intensive query. Thus, these queries benefit from the workload change. However, for less CPU intensive queries (i.e. *grp7*) the propability for the CPU being blocked by a more CPU intensive query increases. In conclusion, these benchmark results show the importance of performance predictions in the cloud context. Changes in the workload which can occur i.e. when the usage profile of one tenant changes. However, these simple changes can result in major performance losses depending on the specific type of workload.

3.3 Performance behavior in different runtime configurations

In this section, we describe the performance influence of runtime environment factors. In a first experiment, we limited the number of *jobworker* threads the database can utilize and compared the results to the setup with unlimited *jobworker* threads.

Figure 2b shows mixed results in the environment with only one active tenant. In general, queries with an increased number of sleeping phases (i.e. *grp5*) seem to benefit from the parameter especially in scenarios with only a low load. Through the static parameter *max_concurrency* a single database tenant cannot longer fully utilize the CPU resources in many workload scenarios. This results in the decreased performance for most OLAP queries. In multi-tenant environments, Figure 2c shows a much clearer picture of the performance behavior with the *limiting parameter* enabled. In scenarios with low load, queries with a higher number of sleeping phases benefit clearly from the parameter change. However, CPU intensive queries with less sleeping phases (i.e. *grp1 and 2*) show a performance loss. In a less intense workload, the probability of the CPU resources being blocked through an CPU intensive query decreases. Thus, the duration of the sleeping phases can be decreased. In scenarios with higher loads, the effect does not continue, as there are only slight differences in the performance behavior in these cases. Additionally, we noticed a significant lower difference between the response time and the processing time during the benchmark runs with a limited number of *jobworker* threads. This can be explained by an increased resource availability for other relevant database threads. In order to analyze the performance improvement through the assignment of more CPU resources, we changed the CPU assignment of the LPAR from two to four for the next benchmark runs. Figure 2d shows the performance improvement through the additional CPU resources. In general, queries with more sleeping phases especially profit from the additional resources in the scenario with only five active users. In these scenarios, the huge difference between the performance improvements of the individual queries is noticeable. The effect is much less intense in high load scenarios. The performance improvement is slightly higher in multi-tenant scenarios. This performance behavior can be explained by a decreasing duration of the sleeping phases. Threads in the sleeping status can be assigned faster to a processing unit due to the increased CPU resources. To further analyze the performance improvement through hardware multithreading, we changed the setup of our database LPAR to utilize no SMT at all. Afterwards, we set the LPAR to utilize SMT-2, SMT-4 and SMT-8. Figure 2e and Figure 2f show the resulting performance improvements of the different SMT-settings compared to the benchmark run with SMT disabled. It is noticeable, queries with a high CPU demand combined with a high count of sleeping phases and a relatively low number of active *jobworker* threads (i.e. *grp5*) usually benefit from SMT. This effect is very intense in low-load scenarios as Figure 2e shows. In multi-tenant scenarios, this effect further increases. Queries with a low CPU demand and a high number of active *jobworker* threads



(a) Impact of workload changes on the performance in a single tenant environment

(b) Limited number of *jobworker* threads in a single tenant environment

(c) Limited number of *jobworker* threads in a multi tenant environment

(d) Asignment of more CPU resources in a single tenant environment

(e) Performance improvement through SMT in a single tenant environment with 5 active Users

(f) Performance improvement through SMT in a single tenant environment with 50 active Users

Figure 2: Resulting performance changes

(i.e. *grp6 and 7*) show almost no benefit with SMT-2 enabled. They also show a lower performance with SMT-8 compared to SMT-4. This is the result of these queries not being able to benefit from the additional SMT capabilities. Increasing complexity regarding the access of the CPU cache results in a slight performance decrease in such cases. Figure 2f shows the performance improvements in high load scenarios. In general, the differences between the individual queries regarding their performance improvement through SMT are much lower than in low load scenarios. Additionally, all queries benefit from SMT-4 and SMT-8 under high workload. This performance behavior can be explained by the higher resource demand related to this workload scenario.

In conclusion, these results show the dependency of the performance on multiple workload-related aspects. Depending on the workload, identical changes in the database configuration can either improve the performance or result in performance losses. Our monitoring solution allows cloud providers to closer analyze their workload. In order to operate the databases in an efficient and cost-effective manner, this analysis is crucial. Detailed knowledge about the resource usage of the specific workload allows cloud providers to deploy database tenants efficiently. Furthermore, valuable resources can be assigned dynamically where they are needed. To assign too many or too few resources is disadvantageous, since either performance goals are not met or unneeded resources are assigned. With detailed knowledge about the workload, cloud providers can avoid both situations. Changing CPU-related resources i.e. by migrating the database to a more powerful server or by assigning more CPU-resources in a virtualized environment also results in differing degrees of success depending on the specific workload scenario.

4 RELATED WORK

In [12], the author provides performance insights into the in-memory database SAP HANA in a multi-tenant configuration. However, he only considers the database and the applied workload as a black box and give no further insights about performance-relevant factors. Furthermore, he does not consider the efficiency of thread usage in his work. In his experiments only small sized tenants are used, which is unlikely in a real world scenario.

The authors in [6] provide more fine-grained performance insights into SAP HANA in a multi-tenant configuration considering amongst other factors differently sized tenant databases, a varying workload and different CPU assignments. In [8] they extend their work by providing new models for the prediction of memory occupancy. In [7], they further extend their work and provide insights into the usage of threads. However, they only measure the average number of utilized CPU cores to obtain the thread usage through the queries. Furthermore, they utilize a lower monitoring resolution resulting in a lower accuracy when considering the resource demand of the individual queries. In this paper, we could show the limitations of this approach by performing benchmarks in various scenarios. Considering only the utilization of CPU cores is not sufficient to explain the performance behavior of the TPC-H queries in our scenarios. Thus, we extended their work by providing more fine-grained insights into thread usage in varying hardware environments.

5 CONCLUSION AND FUTURE WORK

In our work, we provided fine-grained performance insights on the in-memory database SAP HANA. We have built a monitoring setup allowing us to perform a detailed analysis of the thread-utilization of the database. Our setup is also capable of collecting data with a very high resolution, preventing any losses through inaccurate monitoring data. We have shown the dependency of several metrics on the performance behavior in multiple scenarios. Furthermore, our monitoring setup allowed us to group the TPC-H queries according to their resource demand. The fine-grained analysis of the resource-demand of different queries allowed us to explain anomalies when observing their performance behavior in different workload scenarios as well as in different runtime environments.

In further work, we plan to create a fine-grained performance prediction model allowing us to simulate the performance behavior in different scenarios.

REFERENCES

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A View of Cloud Computing. *Commun. ACM* 53, 4 (April 2010), 50–58. https://doi.org/10.1145/1721654.1721672

[2] Andreas Brunnert, Christian Vögele, Alexandru Danciu, Matthias Pfaff, Manuel Mayer, and Helmut Krcmar. 2014. Performance Management Work. *Business & Information Systems Engineering* 6, 3 (01 Jun 2014), 177–179. https://doi.org/10.1007/s12599-014-0323-7

[3] F. Dehne, Q. Kong, A. Rau-Chaplin, H. Zaboli, and R. Zhou. 2015. Scalable real-time OLAP on cloud architectures. *J. Parallel and Distrib. Comput.* 79-80 (2015), 31 – 41. https://doi.org/10.1016/j.jpdc.2014.08.006 Special Issue on Scalable Systems for Big Data Management and Analytics.

[4] Martin Grund, Jan Schaffner, Jens Krueger, Jan Brunnert, and Alexander Zeier. 2010. The Effects of Virtualization on Main Memory Systems. In *Proceedings of the Sixth International Workshop on Data Management on New Hardware (DaMoN '10)*. ACM, New York, NY, USA, 41–46. https://doi.org/10.1145/1869389.1869395

[5] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh. 1998. An analysis of database workload performance on simultaneous multithreaded processors. In *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235)*. 39–50. https://doi.org/10.1109/ISCA.1998.694761

[6] Karsten Molka and Giuliano Casale. 2015. Experiments or simulation? A characterization of evaluation methods for in-memory databases. In *11th International Conference on Network and Service Management (CNSM 2015)*. IEEE, 201–209. https://doi.org/10.1109/CNSM.2015.7367360

[7] Karsten Molka and Giuliano Casale. 2016. Contention-Aware Workload Placement for In-Memory Databases in Cloud Environments. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS 2016)* 2, 1, Article 1 (Sept. 2016), 29 pages. https://doi.org/10.1145/2961888

[8] K. Molka and G. Casale. 2016. Efficient Memory Occupancy Models for In-memory Databases. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 430–432. https://doi.org/10.1109/MASCOTS.2016.56

[9] K. Molka and G. Casale. 2017. Energy-efficient resource allocation and provisioning for in-memory database clusters. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 19–27. https://doi.org/10.23919/INM.2017.7987260

[10] Tobias Mühlbauer, Wolf Rödiger, Andreas Kipf, Alfons Kemper, and Thomas Neumann. 2015. High-Performance Main-Memory Database Systems and Modern Virtualization: Friends or Foes?. In *Proceedings of the Fourth Workshop on Data Analytics in the Cloud (DanaC'15)*. ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/2799562.2799643

[11] Dominik Paluch, Harald Kienegger, and Helmut Krcmar. 2018. A Workload-Dependent Performance Analysis of an In-Memory Database in a Multi-Tenant Configuration. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18)*. ACM, New York, NY, USA, 131–134. https://doi.org/10.1145/3185768.3186290

[12] Jan Schaffner. 2014. *Multi Tenancy for Cloud-Based In-Memory Column Databases: Workload Management and Data Placement.* Springer International Publishing, Heidelberg. https://doi.org/10.1007/978-3-319-00497-6_1

[13] Transaction Processing Performance Council. 2018. TPC-H benchmark specification. http://www.tpc.org/tpch/.