A Workload-Dependent Performance Analysis of an In-Memory Database in a Multi-Tenant Configuration

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

Modern in-memory database systems begin to provide multi-tenancy features. In contrast to the traditional operation of one large database appliance per system, the utilization of the multi-tenancy features allows for multiple database containers running on one system. Consequently, the database tenants share the same system resources, which has an influence on their performance. Understanding the performance of database tenants in different setups with varying workloads is a challenging task. However, knowledge of the performance behavior is crucial in order to benefit from multi-tenancy. In this paper, we provide fine-grained performance insights of the in-memory database SAP HANA in a multi-tenant configuration. We perform multiple benchmark runs utilizing an online analytical processing benchmark in order to retrieve information about the performance behavior of the multi-tenant database containers. Furthermore, we provide an analysis of the collected results and show a more efficient usage of threads in an environment with less active tenants under specific workload conditions.

CCS CONCEPTS

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

KEYWORDS

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

ACM Reference Format:
Dominik Paluch, Harald Kienegger, and Helmut Krcmar. 2018. A Workload-Dependent Performance Analysis of an In-Memory Database in a Multi-Tenant Configuration. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion , April 9–13, 2018, Berlin, Germany.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3185768.3186290

1 INTRODUCTION

A rising number of enterprises migrate their enterprise applications at least partially to the cloud. This allows them to reduce administration costs since they do not need to set up and operate these applications on their own. By utilizing cloud offerings, they can even avoid having to operate an own data center. Databases are an essential part of enterprise applications. Thus, the advantages of cloud offerings lead to a rising interest in Database-as-a-Service (DaaS) offerings. Recently, cloud providers started to include in-memory databases to their on-demand offerings.

In order to use their resources efficiently, providers take advantage of virtualization and multi-tenancy to decrease hardware-, energy- and software licensing costs as well as reducing administration efforts. Thus, multi-tenancy has the potential to increase efficiency. To achieve this goal, cloud providers aim to operate a high number of tenants on their hardware. Thus, performance decreases and costly violations of Service Level Agreements (SLAs) might occur. When the software environment changes, providers have to analyze the effects of these changes on the performance and consider them in their landscape [1]. Gaining knowledge about the performance behavior of database tenants in different setups with varying workloads is a challenging task. However, knowledge of the performance behavior is crucial in order to benefit from multi-tenancy. Recent work focusses on performance behavior and performance predictions of databases in a multi-tenant configuration [2, 3, 5, 7, 9].

In this paper, we provide an in-depth analysis of the performance behavior of the in-memory database SAP HANA in a multi-tenant configuration through conducting a set of experiments. Since the performance of an in-memory database is mainly dependent on its workload [6], our work focusses on a fine-grained analysis of the performance impact of varying analytical workload on the performance of the database tenants. Being mainly independent on disk access rates, in-memory databases have advantages when processing analytical data in contrast to disk-based databases. Thus, these databases are favored for online analytical processing (OLAP). For this reason, we chose an OLAP benchmark for our experiments. The main contribution of this paper is a performance insight into an in-memory database in a multi-tenant configuration. We show that the usage of threads has a significant influence on the performance.

The remainder of the paper is structured as follows. Section 2 provides an overview over the utilized methodology describing the environment and the design of the conducted experiment. In Section 3, we discuss the results of the experiment giving fine-grained insights into the performance behavior of the database. Section 4 contains an overview of related work. Finally, we draw a conclusion and discuss future work in Section 5.

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5629-9/18/04…$15.00
https://doi.org/10.1145/3185768.3186290

Table 1: Parameters of the experiment

Parameter	Description
x_1	Executed TPC-H Query
x_2	Number of active tenant databases
x_3	Number of concurrent query executions

2 METHODOLOGY

2.1 Hardware and Software Setup

For our experiments, we used the TPC-H benchmark suite performing OLAP workload on a SAP HANA database [10]. This benchmark is a decision-support benchmark claiming to utilize queries and data with industry-wide relevance. We used the most recent HANA version 2.0 SP 2 utilizing its multi-tenancy features. We have set up five tenants using a TPC-H data set with a scale factor of 30 resulting in tenant sizes of 30 GB each. In our setup, we used SLES 12 SP2 as the operating system. Our hardware platform consisted of an IBM Power E870 server providing 40 physical CPU cores in four sockets with an eight-tread simultaneous multithreading (SMT-8) engine providing eight virtual CPUs per core. The CPU had a clock speed of 4.19 GHz. The server was equipped with 4.096 GB RAM. Furthermore, the server provided the firmware-based virtualization PowerVM. For our experiments, we used a virtual machine or logical partition (LPAR) with 256 GB RAM and 2 CPU cores with SMT-8. To exclude any network-related performance impact, we utilized a second LPAR on the same server for our benchmark driver. The benchmark driver utilized shell scripts in order to perform the benchmarks. To measure CPU utilization we used the tool *top* running on the database LPAR with a sampling rate of 1 second.

2.2 Experimental Design

In our experiment, we chose a set of parameters listed in Table 1 in order to get fine-grained information about the performance behavior of the database. Since the performance is mainly dependent on its workload, we chose not to conduct the TPC-H benchmark as intended. Performed in its intended way, the TPC-H benchmark defines a user as a set of queries executed in a predefined order. However, this does not allow an analysis of the individual impact of one query on the database's performance. Hence, we chose to run each TPC-H query isolated. In our context, we defined the number of users as the number of concurrently executed queries of one type. With parameter x_1 we considered the individual TPC-H query. To avoid including caching effects in our results, we executed the queries as regular, non-prepared statements. Parameter x_3 mainly determines the workload intensity. It is defined as the number of concurrent executions of a certain query. As stated earlier, we utilized the multi-tenant database containers of SAP HANA in our experiment. SAP HANA consists of multiple server processes. However, the so-called *index server* process is mainly responsible for the performance of a tenant. In the multi-tenancy concept of SAP HANA, every tenant database container is running its own *index server* process. Tenant databases are self-contained databases separated from each other regarding data, security and performance. We considered the number of active tenant databases in parameter x_2 with a maximum of five active tenants. This allows further insights on the performance behavior in a multi-tenant setup. In our experiment, we conducted multiple benchmark runs observing the response times of the executed queries.

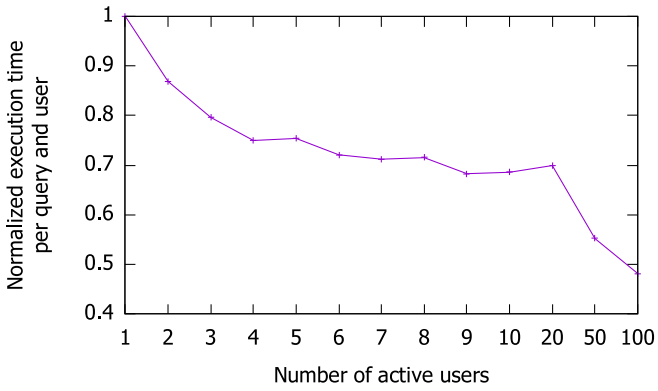

Figure 1: Normalized execution time with a single active tenant database.

The first run utilized a single active tenant database. We executed the single TPC-H queries and varied the number of users from one to 100. Because of a high benchmark runtime with larger user numbers, each user conducted five executions per query. The results show the performance behavior of each query and how well the performance scales in a single tenant. In our second run, we varied the number of active tenants from one to five in a setup with 20 active users. To execute a similar workload on the complete system during this run we utilized 20, 10, 7, 5 or 4 users per tenant to achieve 20 parallel executions. Furthermore, we raised the number of active users to 50 in our third benchmark run. In this case, we utilized 50, 25, 17, 13 or respectively 10 users per tenant. In these benchmark runs, each user conducted 50 executions per query in order to reduce measurement errors. The results show the performance behavior of the database in different multi-tenant scenarios. We chose to limit the number of concurrently active tenants, because the benchmark utilizes the full CPU capacity even with less active tenants. A higher number of active tenants should give no further performance insights. Since a benchmark run with 50 users took over one week to complete, we chose to limit the number of users to 50.

3 RESULTS

3.1 Discussion of the results

Figure 1 shows the results of the first benchmark run. We noticed the normalized execution time being constant up to 20 users. This implies a strict linear growth of response time within this range of users. The relatively high execution times with only one or two active users are the result of limited utilization of multithreading capabilities of certain queries. More interesting are the results with 50 and 100 users. The results in these cases are significantly lower than expected. However, a further analysis of the CPU usage has shown no difference between the scenario with 20 users, 50 users or 100 users. All those scenarios utilized the full CPU capacity.

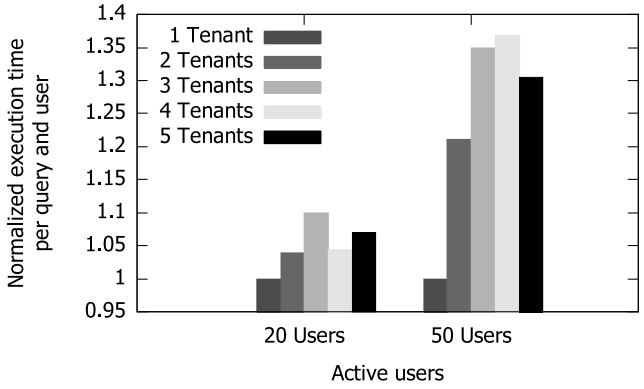

Figure 2: Normalized execution time with multiple active tenant databases in comparison to a single active tenant.

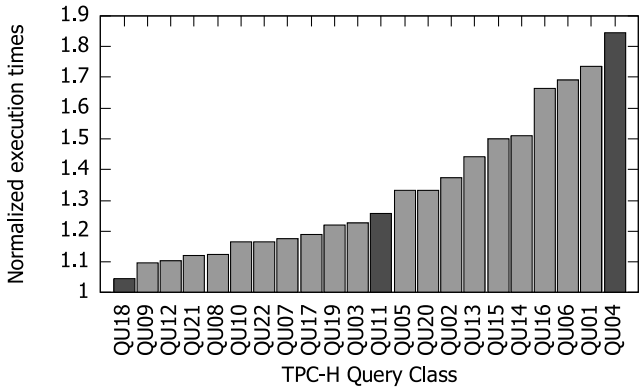

Figure 3: Normalized execution times with five active tenant databases compared to the execution time in one active tenant database with 50 users.

To further investigate the unusual performance behavior, we now analyze the results of the following benchmark runs. We show the results of the second and third benchmark runs in Figure 2. It is noticeable, that the execution of identical workload split to a varying number of active tenants is very similar in scenarios with 20 users, although we use different data sets in each tenant database. However, in the scenario with 50 active users, we noticed a significant difference, whether the workload has been executed in a single tenant or in multiple tenants. We show the dependency between the performance of the database, the number of active tenants and the executed workload in Figure 3. To highlight the performance differences, we normalize the results and compare the execution times of 50 users in a single tenant with the execution times of 10 users per tenant with five active tenants. The results show a very different performance behavior depending on the executed TPC-H query.

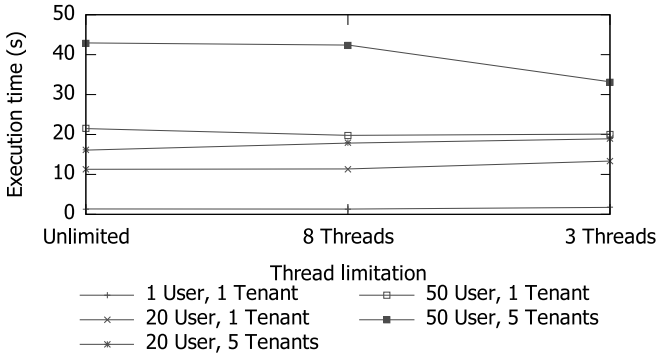

Figure 4: Execution time with *JobWorker* thread limitation.

3.2 In-depth analysis of the results

The obtained results indicate a difference in the thread handling depending on the workload. Caching effects seem unlikely, since they would not only show an impact in scenarios with a high number of active users. We utilized the performance trace, which is built into the SAP HANA database, to achieve insights into the performance behavior of a certain query. In the following, we further examine three queries with a different execution behavior in a single tenant environment and in a multi-tenant environment. We pick query 18 (QU18) as an example for a query without large performance differences. Furthermore, we choose query 11 (QU11) with intermediate differences in execution times. Finally, we closer examine query 04 (QU04) having significant performance differences in the respective environments. As mentioned before, Query 18 shows a relatively constant execution time independent on the number of active tenants as Figure 3 shows. This query returns customers ordering large volumes. It consists of multiple *inner-join* operations and two *aggregate* operations. The *aggregate* operations are handled by the method *BwPopAggregateParallel*, which creates a *Hashmap* structure and can be parallelized to a high degree [4]. When executing this query, the first aggregation is the largest contributor to the execution time. Query 11 returns the most important subset of the supplier's stock. The most relevant operations are two *aggregate* operations and one *inner-join* operation. In this case, the *BwPopJoin13* method creates intermediate data structures and handles the *inner-join* operation. The major contributors to the total execution time are one *BwPopJoin13* and two *BwPopAggregateParallel* methods. Query 04 performs a priority check on the orders. It consists of a *left-semi-join* operation followed by a *grouping* operation. The most time consuming method is *JEStep2*, which is part of the *join* operation and also creates intermediate data structures.

In the scenario with five active tenants, query 04 shows a rather constant execution time . In contrast, considering the scenario with only one active tenant, we notice an increasing execution time to a certain point. When reaching the maximum execution time, it returns to a lower value. This shows certain threads being executed much faster than others, resulting in a lower mean execution time. SAP HANA utilizes specific threads for handling OLAP workload, naming them *JobWorkers* [8]. We analyzed the number of *JobWorker* threads SAP HANA utilizes in each mentioned scenario.

In the scenario with five active tenants, we notice the database creating a significantly higher number of utilized *JobWorker* threads. Of course, this reduces the CPU time each thread receives, resulting in longer mean execution times. To validate these results, we conducted a small benchmark run with query 04. In this run, we limit the number of *JobWorker* threads, each tenant can utilize to three.

The result of the benchmark run is represented graphically in Figure 04. It shows lower execution times for query 04 with 50 active users when limiting the number of *JobWorker* threads to three. Thus, each thread can consume more CPU time. In this particular scenario the additional CPU time, each thread receives helps to accelerate the execution time. However, it still shows a noticeable difference between the execution times in an environment with one active tenant and the execution times with five active tenants. Although the number of *JobWorker* threads has decreased, it still utilizes more threads with five active tenants. Thus, we can conclude SAP HANA is able to improve the efficient usage of threads in an environment with less active tenants for a certain workload.

4 RELATED WORK

There are several attempts of benchmarking, analyzing and predicting the performance of a database in a multi-tenant setup. The authors in [3] propose a benchmark framework providing certain benchmark guidelines in order to analyze the performance of a database in a multi-tenant setup. However, the authors do not take into account similar workload being executed in a varying number of tenants. In addition, they propose utilizing the same data set in different tenants. This increases the probability of caching effects influencing the results. In a real world scenario, utilizing the same data set in multiple tenants is unlikely. The author in [9] focuses on SLA violations of SAP HANA in a multi-tenant environment. He utilizes a modified TPC-H benchmark using very small tenant sizes. Furthermore, he does not conduct a fine-grained analysis on how the workload affects the performance of the database. In [5] performance optimizations of a disk-based database in a multi-tenant environment are provided. With given performance SLAs and hardware capacities, their developed framework optimizes hardware assignments. Furthermore, it returns a tenant scheduling policy, optimizing the placement of a tenant in a system landscape with multiple servers. The authors in [6, 7] provide a fine-grained performance model for SAP HANA in a multi-tenant environment. In [6], they conduct a similar experiment to analyze the performance of SAP HANA in different multi-tenant scenarios. They provide further insights on memory consumption and energy consumption of the system. In addition, they evaluate different CPU assignment strategies. However, they do only consider a limited number of scenarios with 8 and 16 active users and 2 or 4 active tenants and do not provide a fine-grained analysis of the performance impact of the workload. In [7], they supplement their findings by providing strategies to optimize workload placement in SAP HANA clusters. They further analyze the executed workload and provide a queueing network performance model. However, they do only consider up to 32 active users. Summarizing, they do not consider the different performance behavior of identical workload with a varying number of active tenants.

5 CONCLUSION AND FUTURE WORK

In our work, we provided fine-grained performance insights on the in-memory database SAP HANA. We have shown the dependency between the performance of the data-base, the number of active tenants and the executed workload. In high load situations with multiple active users, we noticed a significant difference between the operation of a system with only a single active tenant and a system with multiple active tenants.

With less active users, this difference is much smaller. Thus, we concluded a more efficient usage of threads in a setting with less active tenants. This effect however is strongly dependent on the workload. The intensity of the effect depended strongly on the executed TPC-H query. By limiting the number of threads, we could observe a performance increase in a certain workload condition. As we could show, changes in the configuration (e.g. modifying the number of active tenants) can result in an unexpected performance behavior. However, the change of other configuration parameters, like the maximum number of threads, can have a positive impact on the performance depending on the workload.

Thus, we plan to conduct more experiments in heterogeneous environments with different tenant sizes in the future. In addition, we plan to consider the effects of varying tenant loads to get further insights on the correlation between executed workload and the performance of a database tenant. We furthermore plan to repeat the experiment with different CPU assignments. Moreover, it is a desired goal to provide an according performance model enabling simulations.

REFERENCES

[1] 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

[2] A. Floratou and J. M. Patel. 2015. Replica Placement in Multi-tenant Database Environments. In *International Congress on Big Data (BigData Congress 2015)*. 246–253. https://doi.org/10.1109/BigDataCongress.2015.42

[3] Tim Kiefer, Benjamin Schlegel, and Wolfgang Lehner. 2013. *MulTe: A Multi-Tenancy Database Benchmark Framework*. Springer Berlin Heidelberg, Berlin, Heidelberg, 92–107. https://doi.org/10.1007/978-3-642-36727-4_7

[4] Akash Kumar. 2015. *PlanViz: Improving SAP HANA Performance*. Rheinwerk Verlag, Bonn.

[5] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan. 2012. Towards Multi-tenant Performance SLOs. In *28th International Conference on Data Engineering (ICDE 2012)*. 702–713. https://doi.org/10.1109/ICDE.2012.101

[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] SAP SE. 2016. *SAP HANA Troubleshooting and Performance Analysis Guide*. Technical Report.

[9] 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

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