# MulTe: A Multi-Tenancy Database Benchmark Framework

Tim Kiefer, Benjamin Schlegel, and Wolfgang Lehner

Dresden University of Technology
Database Technology Group
Dresden, Germany
{tim.kiefer, benjamin.schlegel, wolfgang.lehner}@tu-dresden.de

**Abstract.** Multi-tenancy in relational databases has been a topic of interest for a couple of years. On the one hand, ever increasing capabilities and capacities of modern hardware easily allow for multiple database applications to share one system. On the other hand, cloud computing leads to outsourcing of many applications to service architectures, which in turn leads to offerings for relational databases in the cloud, as well.

The ability to benchmark multi-tenancy database systems (MT-DBMSs) is imperative to evaluate and compare systems and helps to reveal otherwise unnoticed shortcomings. With several tenants sharing a MT-DBMS, a benchmark is considerably different compared to classic database benchmarks and calls for new benchmarking methods and performance metrics. Unfortunately, there is no single, well-accepted multi-tenancy benchmark for MT-DBMSs available and few efforts have been made regarding the methodology and general tooling of the process.

We propose a method to benchmark MT-DBMSs and provide a framework for building such benchmarks. To support the cumbersome process of defining and generating tenants, loading and querying their data, and analyzing the results we propose and provide MulTe, an open-source framework that helps with all these steps.

## 1 Introduction

Academia and industry have shown increasing interest in multi-tenancy in relational databases for the last couple of years. Ever increasing capabilities and capacities of modern hardware easily allow for multiple database applications with moderate requirements to share one system. At the same time, cloud computing leads to outsourcing of many applications to service architectures (IaaS, SaaS), which in turn leads to offerings for relational databases hosted in the cloud as well [1, 2]. We refer to any database system that accommodates multiple tenants by means of virtualization and resource sharing, either on a single machine or on an infrastructure of machines, as a multi-tenancy database management system (MT-DBMS). All MT-DBMSs lead to interesting challenges like, (1) assigning logical resources to physical ones; (2) configuring physical systems (e.g.,

database design, tuning parameters); and (3) balancing load across physical resources, all of which require thorough testing to ensure scalability and system quality.

Although there have been several works on how to build multi-tenancy systems, little work has been done on how to benchmark and evaluate these systems—partly due to the diversity of the systems and the complexity of possible benchmark setups. There are many well-accepted database benchmarks, e.g., TPC benchmarks like TPC-C or TPC-H [3] or the XML benchmark TPoX [4]. These benchmarks concentrate on a certain scenario (e.g., OLTP or OLAP) and measure a system's peak performance with respect to the given scenario. The key challenge for multi-tenancy systems is usually not to provide the highest peak performance, but to scale well and deal with multiple changing workloads under additional requirements like performance isolation and fairness. A first attempt has been made to propose a benchmark for database-centric workloads in virtual machines (TPC-V) [5]. This benchmark is still under development and although it has the same goal as our work, i.e., to benchmark multi-tenancy systems, the means and priorities differ significantly. TPC-V, like all other TPC benchmarks, provides strict rules and conditions to ensure a comparable and long-lasting benchmark. At the same time, many implementation details and efforts are left to the benchmark sponsor. In contrast, our work provides a very flexible environment and tools for rapid developments and implementations of new multi-tenancy benchmarks. We will discuss our work's relation to TPC-V in more detail in Section 6.

In this work, we propose and provide methodology, workflow, and associated tools to benchmark MT-DBMSs. A great variety of MT-DBMSs exists and different systems are complex and diverse. Hence, we do not propose a single benchmark that fits all systems, but rather provide the framework MulTe[1] that allows for generating specific benchmarks quickly and easily. As shown in Figure 1, our approach is to re-use existing benchmarks, including their schemas, data, and queries/statements and to generate instances of these benchmarks to represent different tenants. Each tenant is given an individual, time-dependent workload that reflects a user's behavior. A workload driver is used to run all tenants' workloads against any multi-tenancy database system and to collect all execution statistics and benchmark metrics. All steps are supported with tools which together form our multi-tenancy benchmark framework MulTe.

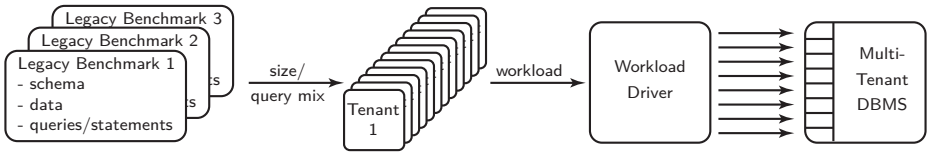



Fig. 1: Multi-Tenancy Benchmark Workflow

[1] http://wwwdb.inf.tu-dresden.de/research-projects/projects/multe/

To demonstrate the ability to create meaningful benchmarks, we will sketch the process of building three different exemplary benchmarks throughout the paper. We will try to answer the following three questions related to MT-DBMSs:

1. ScalabilityBenchmark: Does the system scale well with the number of tenants? How many tenants are able run in parallel? What is the individual performance? What is the overall performance?
2. FairnessBenchmark: How fair is the system, i.e., are the available resources equally available to all tenants? If tenants have different priorities, how well are they implemented?
3. IsolationBenchmark: How well are tenants isolated from one another with respect to performance? How do individual tenants influence other tenants' performance?

MulTe is flexible enough that many other aspects and components, e.g., for load balancing, of MT-DBMSs can be tested as well.

To summarize, our key contributions in this paper are:

- MulTe, an easy-to-use, extensible framework to build and run benchmarks for MT-DBMSs. MulTe comes with an example implementation for all components (e.g., TPC-H is supported as benchmark type), but is designed to allow for future extensions.
- The introduction of time-dependent workloads to allow for realistic, dynamic tenants and the analysis of effects caused by them. This includes a brief discussion of how time-dependent workloads should be defined.
- A new performance metric *relative execution time* for multi-tenancy benchmarks that considers time-dependent workloads.

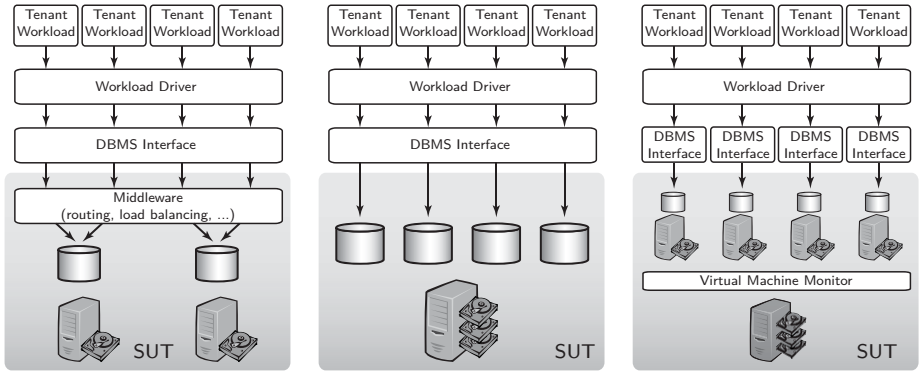
The rest of the paper is organized as follows: In Section 2, we recapitulate the fundamentals of MT-DBMSs before we introduce our multi-tenancy benchmark concept in Section 3. Following in Section 4, we provide an overview of design and implementation decisions for MulTe. In Section 5, we describe the process of building and running exemplary benchmarks with MulTe before we reference related work and conclude in Sections 6 and 7, respectively.

## 2 Multi-Tenancy Database Management Systems

In this section, we briefly recap the fundamentals of MT-DBMSs for the interested reader. We further demonstrate the range of multi-tenancy systems that can be evaluated with MulTe by naming some examples.

The layered system stack of a DBMS—from the database schema to the operating system—allows for consolidation at different levels. Previous works have classified virtualization schemes, leading to classes like *private process*, *private database*, or *private schema* [6, 7]. A variety of specific MT-DBMSs can be built following this classification, all of which can be tested with MulTe. Figure 2 shows three possible Systems Under Test (SUT). The topmost two layers (Tenant Workload and Workload Driver) are provided by MulTe and are the same

for all systems, whereas the implementation of the multi-tenancy functionality differs significantly.




(a) Private Schema (b) Private Database (c) Private OS

Fig. 2: MT-DBMSs Under Test

*Private Schema:* The system shown in Figure 2a is an example for a private schema virtualization. A middleware is responsible for the routing and load balancing of different tenants. Each tenant is implemented as a schema in any of a small number of physical databases. In this setup, MulTe can be used to test the performance of the middleware with respect to routing and load balancing as well as each backend DBMS. All three exemplary benchmarks—ScalabilityBenchmark, FairnessBenchmark, and IsolationBenchmark—are of great interest in this scenario.

*Private Database:* The system shown in Figure 2b implements a private database virtualization. Here, a single large machine hosts a number of private databases—a common setup for many database servers. MulTe, especially the ScalabilityBenchmark, can be used to test the system's ability to scale with the number of tenants and hence databases. The DBMS is the main focus for benchmarks in such MT-DBMSs.

*Private Operating System:* The system shown in Figure 2c implements a private OS virtualization where each tenant is implemented with a complete stack of virtual machine, operating system, and database management system. Consequently, MulTe can not only test DBMSs in virtualized environments, but also provide insights on the performance of the virtual machine monitor and its ability to balance and isolate multiple virtual machines that run database workloads (cf. Soror et al. [8]). Again, all three exemplary benchmarks are relevant for such systems.

## 3 Benchmark Framework Conception

To explain our concepts of multi-tenancy benchmarks, we introduce the general workflow first. The idea of time-dependent workloads is described in Section 3.2. Finally, the metric that we propose for multi-tenant benchmarks is presented in Section 3.3.

### 3.1 General Benchmark Workflow

Based on the idea to re-use existing database benchmarks, the general workflow of benchmarks built with MulTe is shown in Figure 3. The characteristics of a tenant are defined with a basic set of parameters. This includes, e.g., the benchmark type (like TPC-H), the size of the raw data, and the query mix. Based on these descriptions, instances of the benchmarks are generated and loaded as tenants into the MT-DBMS. All three steps are supported by a set of Python scripts. Once populated, a Java workload driver runs the tenants' workloads against the MT-DBMS and collects all relevant performance statistics. These statistics are analyzed, evaluated, and visualized with Python and R.

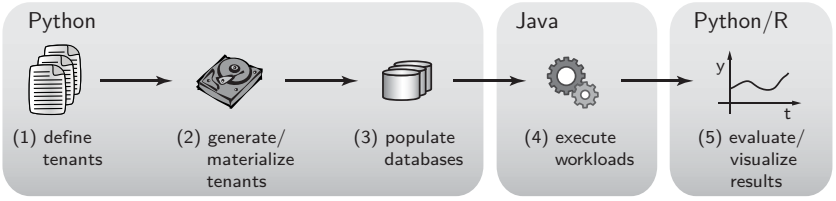



Fig. 3: MulTe—General Benchmark Workflow

### 3.2 Time Dependent Workloads—Activities

One approach to test MT-DBMSs is to let all tenants execute constant (peak) workload over a period of time. While this is perfectly valid to evaluate the system's ability to scale with the number of tenants in extreme load situations, e.g., for the ScalabilityBenchmark, it presents a somewhat pathologic scenario and more importantly does not help to evaluate system characteristics like fairness and performance isolation. A key assumption for multi-tenancy systems is that at no time, all tenants are working simultaneously. Figure 4 shows two tenants' activities, i.e., their workload over time, and it can be seen that their peak loads are at different times. However, the aggregated workload for the entire system, which is less than the sum of both peak workloads, dictates each tenant's performance. A MT-DBMS should take the specific behavior of its tenants into account (as much as possible) and provision for realistic, average load scenarios. In our work, we would like to acknowledge that tenants are different to

one another and over time. To relate the concept of activities to our exemplary benchmarks, we briefly discuss their need in context of the benchmark purposes. All three benchmarks will be further discussed in Section 5.

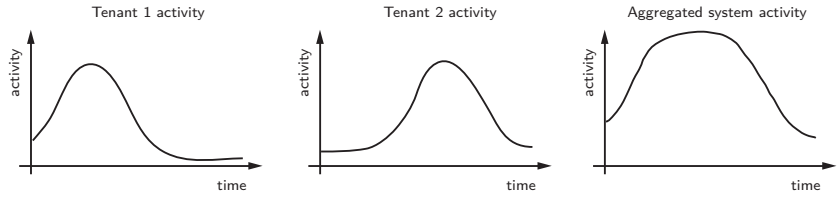



Fig. 4: Time Dependent Tenant Workloads—Activities

1. ScalabilityBenchmark: A simple constant workload is sufficient to test a system's scalability.
2. FairnessBenchmark: To evaluate fairness, tenants can execute constant load over time. However, to evaluate how resources are split among tenants, they must execute possibly different workloads at different stress-levels.
3. IsolationBenchmark: To evaluate performance isolation, a tenant's workload must change over time. For example, Tenant 1 can run a constant workload while Tenant 2 executes resource-intensive queries sporadically. The reaction of the first tenant's performance to the second tenant's activity gives an important insight into the MT-DBMS.

To support varying tenant loads, we introduce *time-dependent workloads* or *activities* in our benchmarks. As indicated before, we understand a tenant's *activity* as the combination of the queries that the tenant executes and the timeline that defines, when queries are executed. A design decision in MulTe is that activities are defined per tenant (as opposed to for the whole system) so that tenants can be driven independently and possibly in a distributed fashion.

Given these pre-requisites, the question rises how activities should be defined; an interesting research topic on its own. We shortly introduce how we solve the problem in our implementation of MulTe. However, the framework is flexible enough to support other, more complex approaches in future releases.

*Aggregated or disaggregated workload definition:* The overall goal of the activities is to simulate a system of multiple tenants to get a realistic global view on the system. However, there seem to be two general approaches to achieve this. The *aggregation approach*—which we decided to implement—defines tenants' activities individually such that the resulting aggregated workload fulfills certain requirements or shows certain characteristics. The challenge is to define individual activities that aggregate to the expected global workload. The *disaggregation approach* on the other hand starts with a global load characteristic for the whole multi-tenancy system. Here, the challenge is then to split (or disaggregate) this global workload in a meaningful way to a set of local tenant activities.

*Set of parameters:* A second challenge when defining activities is to find a compromise between the complexity and the expressiveness of the description. To provide a function definition to calculate the activity at any point in time would be one extreme. Although most powerful, this approach seems to be hard to implement. Defining detailed activity functions for tens or possibly hundreds of clients is not a feasible task. The other extreme would be to rigorously simplify a tenant's activity so that it results in a constant or possibly random workload—only one or two parameters are necessary to implement that, but the possibilities for interesting workloads are limited.

We decided to pick the query to execute randomly (with weights). To describe the timeline of an activity, we use four parameters: MEANSLEEPTIME, PARALLELUSERS, ACTIVITY, and ACTIVITYCONSTRAINT—their meanings are shown in Figure 5. This simple set of parameters allows to model tenants with an on/off behavior, i.e., tenants that are active periodically and idle (for a certain amount of time, MEANSLEEPTIME) in between. When tenants are active, multiple parallel users (PARALLELUSERS) run queries against the database, either constraint by a number of transactions or by an amount of time (depending on ACTIVITY and ACTIVITYCONSTRAINT). In our opinion, this approach is a good compromise between complexity and expressiveness but we continuously work on new methods.

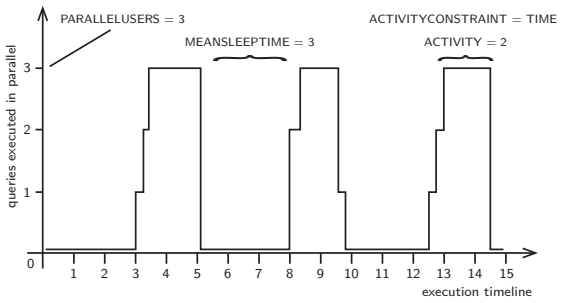



Fig. 5: Workload Parameters

*Independent or correlated parameters:* Once the method for describing activities is chosen, the question remains how to determine good values for the parameters. Again, the challenge is to find a compromise between expressiveness and the feasibility of the task to come up with parameters for possibly hundreds of instances. Our approach is to pick parameter values independently from one another. For example, the MEANSLEEPTIME may follow a Gaussian distribution over all tenants while the raw data size is evenly distributed between a minimum and a maximum. However, a tenants's MEANSLEEPTIME is independent from its raw data size and the other parameters. In the future, more complex extensions of MulTe may find it useful to correlate certain parameters, e.g., to simulate that larger tenants access data more often.

### 3.3 A Performance Metric for Multi-Tenant Database Systems

We argue that MT-DBMSs are so diverse and complex that different aspects need to be considered when a benchmark is designed. A single, classic performance metric, like transactions executed per second, is valid for peak performance evaluations and hence can be used for the ScalabilityBenchmark and the FairnessBenchmark. However, it is not sufficient to answer the question of how well tenants are isolated from one another, which is the intention of the IsolationBenchmark. Hence, with tenants that have activities, a multi-tenancy benchmark needs a new performance metric. Since the tenants' activities are time-dependent, it follows that the new metric should also be time-dependent, i.e., it may differ over the run of the benchmark. Depending on the intention of the executed benchmark, this time-dependent metric may be aggregated to a single value. We propose the *relative execution time over time* as the basic metric for MT-DBMSs.

*Relative execution time:* To learn about a query's execution time under optimal conditions, a *baseline run* is performed in a simulated single-tenant environment. During this baseline run, each tenant is allowed to execute its workload on the otherwise idle system (possibly a couple of times) to collect information about its best-case execution time. During actual *test runs*, each query's execution time is recorded and compared to the baseline execution time. This leads to a *relative execution time* for this query, i.e., a penalty caused by the current overall system load, compared to the best-case scenario. Obviously, this relative execution time can differ significantly for a single query over time depending on the system load and configuration at that time. To reason about a tenant's performance or the overall system performance, individual relative execution times from the queries/transactions can be aggregated.

Having this metric, the question remains how to interpret it to answer different performance questions. We discuss our exemplary benchmarks to illustrate the usage of the new performance metric.

1. ScalabilityBenchmark: To evaluate a system's ability to scale with the number of tenants, different (average) relative execution times measured with different numbers of tenants can be compared.
2. FairnessBenchmark: When multiple tenants run their individual workloads, the differences of the relative execution times can be used to reason about the fairness of the system. Similar relative execution times can indicate a fair system. Large differences can indicate the absence (or poor performance) of a load-balancing component. However, large differences may be intentional, e.g., because different tenants have different service levels.
3. IsolationBenchmark: To evaluate how well tenants are isolated from one another, all tenants execute a steady, moderate workload. At some point in time, a single tenant starts to execute heavy workload. The changes of other tenants' relative execution times after the workload change indicate how well they are isolated from the respective tenant.

## 4 Benchmark Implementation

In this section, we give an overview of design and implementation decisions that we made for the implementation of MulTe. Detailed information about the implementation can be found in the framework package and a related technical report (to appear with this paper).

### 4.1 Framework Design Principles

Our intention is to provide MulTe as a framework for MT-DBMSs that will be widely used and extended by the community. We committed our work to a number of design principles that shall help to increase the value of the framework.

*Easy-to-use:* We provide an example implementation to create tenants that run the TPC-H benchmark against MySQL or PostgreSQL databases. A user only needs to provide a small number of configuration parameters (e.g., host, port, paths, ...) and a workload definition, e.g., following one of the provided example scripts. The framework then generates and loads tenants. The Java workload driver can be used with a provided sample configuration to run the workload.

*Component Re-use:* We re-use existing components wherever possible to increase a benchmark's acceptance and reliability. From the given database benchmark, we re-use, e.g., data generators and schema definitions. The Java workload driver in MulTe is a modification of the workload driver of the TPoX database benchmark [4]. Hence, the strengths of this well-tested tool as well as the experiences and improvements gathered from using it help to improve MulTe.

*Extensibility:* The framework components to generate and load tenants can be replaced such that both, other benchmarks and other database systems can be supported. Python as the scripting language allows users to easily adopt our example implementations and modify them to fit their needs. The workload driver is designed to be able to run a wide variety of workloads, thus supports different benchmarks. Extending the workload driver to support different database systems is also easily possible as long as they provide a JDBC interface.

### 4.2 Python Scripts—Define, Generate, and Load Tenants

The Python scripts/classes to define, generate, and load tenants follow the structure shown in Figure 6. The full Python functionality can be used to define a tenant's set of activity parameters. We provide two implementations that either specify the parameters' values explicitly or pick them randomly (following a distribution) but independent from one another. Given the tenants' definitions, instances of the respective database benchmarks are generated using both, provided data generators and templates. Other instance generators for other benchmark types can be added by implementing the methods `generateData`, `generateSchema`, and `generateWorkload`. A DBMS-specific database executor

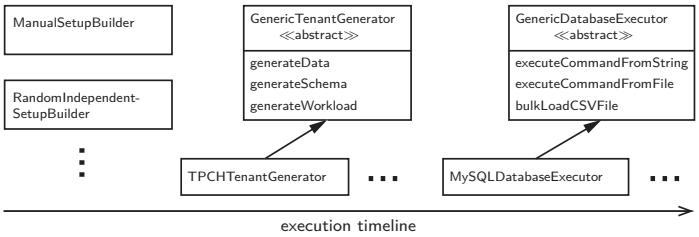



Fig. 6: Python Scripts Overview

is used to load all tenants' data into the databases. Our example implementation uses the command line interfaces of MySQL and PostgreSQL to execute SQL statements and to bulk-load data from CSV-files.

### 4.3 Java Workload Driver

As mentioned before, the Java workload driver, which executes all tenants' workloads against a MT-DBMS, is a modification and extension of the workload driver provided with the TPoX database benchmark. A detailed documentation of the original workload driver and its capabilities can be found on the TPoX website. Here, we would like to briefly show some of the modifications that we made (Figure 7). The class structure is closely coupled to the framework conception. The `WorkloadSuite` uses `GlobalSuiteParameters` for the benchmarks, taken from a simple configuration file (here `suite.xml`). The `WorkloadSuite` furthermore drives multiple tenants (`WorkloadInstance`) that in turn use multiple `ConcurrentUsers` to execute statements in the database. All of a tenant's parameters—stored in `WorkloadInstanceParameters`—are taken from an XML configuration file that is automatically generated by MulTe.

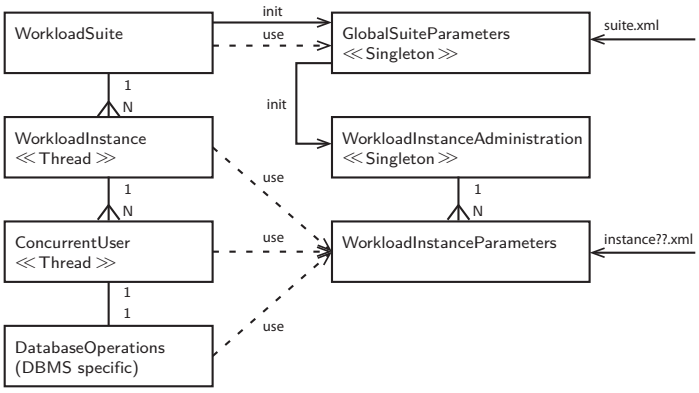



Fig. 7: Java Workload Driver Overview

## 5 Exemplary Benchmark Development and Execution

To show the ability of MulTe to help with the development of multi-tenancy benchmarks, we have implemented the three exemplary benchmarks ScalabilityBenchmark, FairnessBenchmark, and IsolationBenchmark on a dual-socket AMD Opteron machine running Ubuntu 11.10 server, Java, Python, R, and MySQL. We used MulTe to generate up to 8 tenants that all run the TPC-H benchmark with 500MB of raw data. All tenants' data is loaded to private databases in a single MySQL server.

In the following, we are only interested in the process of building and executing a benchmark and not the particular results, which is why we do not further detail the system under test and only show relative results without baselines. We do not intent to make any statement about MySQL's multi-tenancy capabilities.

### 5.1 A Multi-Tenancy Scalability Benchmark

Our first benchmark is the ScalabilityBenchmark described earlier. To run the benchmark, we generated and loaded 8 identical tenants that run constant workload with the parameters denoted in Table 1.

Table 1: Tenant Parameters for the ScalabilityBenchmark

| Parameter | Value |
|---|---|
| TYPE | TPC-H |
| SIZE | 500MB |
| QUERY | TPC-H Query 1 |
| MEANSLEEPTIME | 0 |
| PARALLELUSERS | 5 |
| ACTIVITY | 300 |
| ACTIVITYCONSTRAINT | seconds |

We then collected a baseline for the execution time of TPC-H Query 1 in single-tenant mode. Afterwards, we used the workload driver to run 1, 2, 4, 6, or 8 tenants at the same time and to collect the execution times of all queries. Last, we computed the average relative execution time, an aggregate of our basic metric, the relative execution time over time, as an indicator for the system's ability to scale. The results are shown in Figure 8.

### 5.2 A Multi-Tenancy Fairness Benchmark

The second benchmark that we built is the FairnessBenchmark. We generated and executed two different flavors of the benchmark: one with identical tenants, the other one with two different (groups of) tenants that differ in the query they

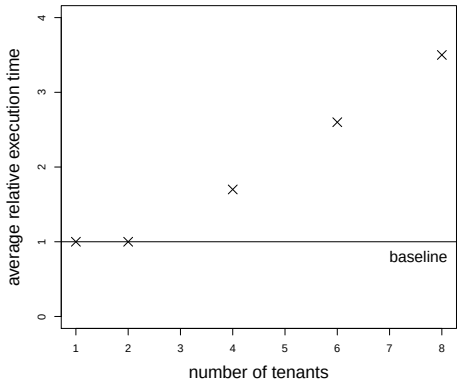
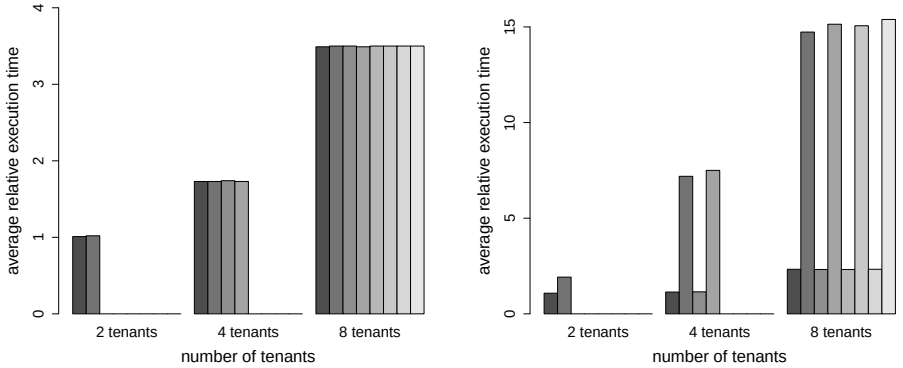



Fig. 8: Scalability Benchmark

execute. In FairnessBenchmark 1, shown in Figure 9a, all tenants execute TPC-H Query 1. In FairnessBenchmark 2, shown in Figure 9b, half of the tenants execute TPC-H Query 1 while the other half executes TPC-H Query 8. The metric shown in the bar charts is the average relative execution time per tenant (note the different scales of the y-axes). It can be seen that, on the one hand, with identical queries the relative execution time goes up with the number of tenants, but the available resources are distributed evenly. On the other hand, with tenants running different queries, one group observes a considerably higher relative execution time.




(a) Fairness Benchmark 1 (b) Fairness Benchmark 2

Fig. 9: Fairness Benchmarks

### 5.3 A Multi-Tenancy Isolation Benchmark

The last of our exemplary benchmarks is the IsolationBenchmark. To evaluate whether and how much a tenant can influence another tenant's performance we generated two different tenant types using the parameters shown in Table 2.

Table 2: Tenants used for the IsolationBenchmark

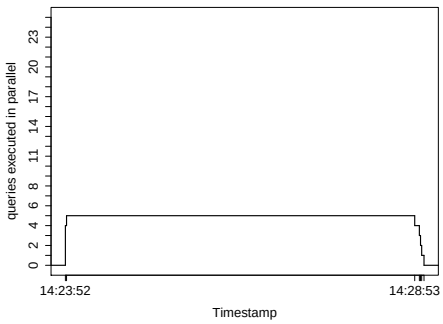
| Parameter | Tenant 1 | Tenant 2 |
|---|---|---|
| TYPE | TPC-H | TPC-H |
| SIZE | 500MB | 500MB |
| QUERY | TPC-H Query 1 | TPC-H Query 8 |
| MEANSLEEPTIME | 0 | 60 seconds |
| PARALLELUSERS | 5 | 20 |
| ACTIVITY | 50 | 10 |
| ACTIVITYCONSTRAINT | transactions | seconds |

It can be seen that both tenants have different activities. While Tenant 1 is running a constant workload, Tenant 2 is idle for 60 second intervals interrupted by short bursts of heavy query execution. The resulting individual loads imposed on the DBMS as well as the aggregated load are shown in Figures 10a–10c. To learn more about the system's ability to isolate tenants, we collected and charted the relative execution times. The results are shown in Figures 10d and 10e.

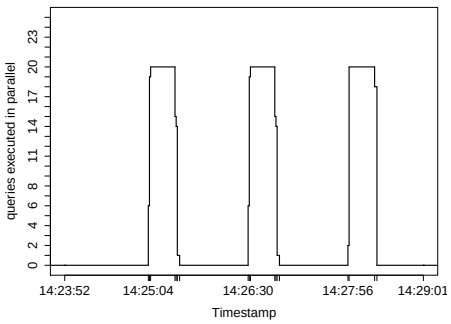
## 6 Related Work

There are several standardized benchmarks like TPC-C, TPC-E, TPC-DS, and TPC-H that are provided by the Transaction Processing Performance Council (TPC) [3]. All of these benchmarks aim at benchmarking database servers for different scenarios; TPC-C, for example, provides a database schema, data population, and workload for a typical OLTP scenario. Unfortunately, none of the TPC benchmarks can be used directly to measure the performance of a MT-DBMS. All of them use a fixed schema for only a single tenant. The queries, schema, and data of these benchmarks, however, can be re-used to create load for multiple tenants. Thus, they can form the foundation for any MT-DBMS benchmark that is built and run using MulTe.

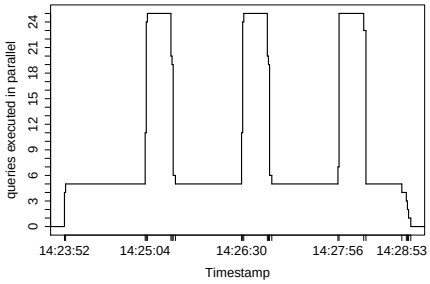
Aulbach et al. [9] describe a testbed that is tailor-made for benchmarking MT-DBMSs. The workload relies on a fixed schema (i.e., a customer-relationship schema) and provides OLTP and OLAP queries for multiple tenants that are hosted on a service provider. Curino et al. [10] define multiple fixed workloads to benchmark their MT-DBMS. The workloads are based on TPC-C and have certain time-varying patterns. MulTe allows to use arbitrary and mixed workloads and schemas, hence allows to build more elaborate benchmarks.



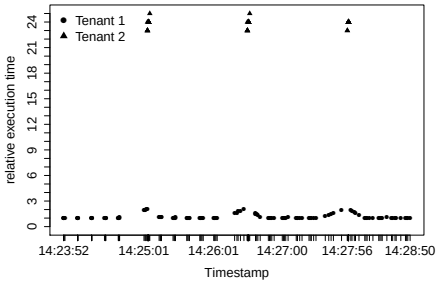

(a) Tenant 1 Activity



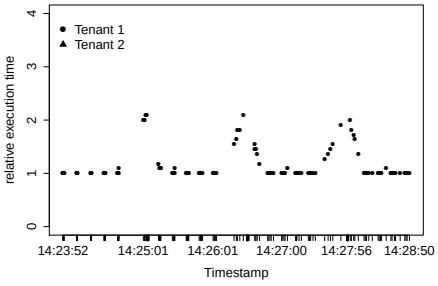

(b) Tenant 2 Activity




(c) Aggregated Activity




(d) Relative Execution Times




(e) Relative Execution Times (zoomed)

Fig. 10: Performance Isolation Benchmark—Activities and Relative Execution Times

TPC-V [5] is a benchmark under development of the TPC suite. It is intended to benchmark databases that are hosted in virtual environments. The main focus lies on virtual machines (cf. Figure 2c), but the same concepts are applicable to other multi-tenancy implementations as well. TPC-V uses different database sizes and numbers of queries to emulate tenants with different loads. Furthermore, the load is varied within twelve 10-minute periods to shift the tenants' peak load to different periods as it is usually the case in real-life applications. Although TPC-V aims for the same direction as MulTe does, it has significantly different priorities. Like other TPC benchmarks, it provides strict guidelines to ensure a comparable and long-lasting benchmark. Consequently it is less flexible with respect to the tested scenarios, executed workloads, and other aspects of the benchmark. Moreover, TPC-V rather concentrates on the benchmark specification than the actual implementation which is left to the benchmark sponsor. In contrast, our goal is to support building benchmarks for all classes of MT-DBMSs (cf. Figure 2) and benchmarks that can be adapted to certain scenarios and questions. With the tools provided by MulTe, new benchmarks can be implemented very fast. This flexibility allows users to quickly expose performance, fairness, and other issues in MT-DBMSs.

## 7 Summary

We introduced and provided MulTe, a framework to help with the process of building and running benchmarks for MT-DBMSs. A multi-tenancy benchmark consists of multiple tenants, each running an arbitrary database benchmark. MulTe provides extensible tools to define, generate, and load all tenants as well as drive workloads and analyze results. Our example implementation of the MulTe components supports the TPC-H benchmark on MySQL or PostgreSQL.

In contrast to classic database management systems, multi-tenancy systems have additional requirements for, e.g., fairness and isolation, that need to be addressed in benchmarks. We introduced activities, i.e., time-dependent workloads, and a related performance metric (relative execution time) to account for these new requirements. We showed the necessity for the time-dependent workloads and a new performance metric as well as the capabilities of MulTe with the three exemplary benchmarks ScalabilityBenchmark, FairnessBenchmark, and IsolationBenchmark.

We would like the community to actively extend MulTe. Additionally, we are also working on extensions for further benchmarks and database systems. To allow all components of the framework to run in a distributed fashion is another planned extension of MulTe that shall ensure its scalability to arbitrarily large MT-DBMSs.

# Bibliography


[1] Microsoft: Microsoft Windows Azure (2012) Available at: http://www.windowsazure.com/en-us/.

[2] Amazon: Amazon Relational Database Service (2012) Available at: http://aws.amazon.com/rds/.

[3] TPC: Transaction Processing Performance Council (2012) Available at: http://www.tpc.org/.

[4] TPoX: Transaction Processing over XML (TPoX) (2012) Available at: http://tpox.sourceforge.net/.

[5] Sethuraman, P., Reza Taheri, H.: TPC-V: A Benchmark for Evaluating the Performance of Database Applications in Virtual Environments. In: Proceedings of the 2nd TPC technology conference on Performance evaluation, measurement and characterization of complex systems - TPCTC '10, Singapore, China, Springer (2010) 121–135

[6] Jacobs, D., Aulbach, S.: Ruminations on Multi-Tenant Databases. In: Fachtagung für Datenbanksysteme in Business, Technologie und Web - BTW '07, Aachen, Germany (2007) 5–9

[7] Kiefer, T., Lehner, W.: Private Table Database Virtualization for DBaaS. In: Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing - UCC '11. Volume 1., Melbourne, Australia, IEEE (December 2011) 328–329

[8] Soror, A.A., Minhas, U.F., Aboulnaga, A., Salem, K., Kokosielis, P., Kamath, S.: Automatic Virtual Machine Configuration for Database Workloads. ACM Transactions on Database Systems (TODS) **35**(1) (February 2010) 1–47

[9] Aulbach, S., Grust, T., Jacobs, D., Kemper, A., Rittinger, J.: Multi-Tenant Databases for Software as a Service: Schema-Mapping techniques. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08, Vancouver, BC, Canada, ACM (2008) 1195–1206

[10] Curino, C., Jones, E.P., Madden, S., Balakrishnan, H.: Workload-Aware Database Monitoring and Consolidation. In: Proceedings of the 2011 ACM SIGMOD international conference on Management of data - SIGMOD '11, Athens, Greece, ACM (2011) 313–324