Simulating Multi-Tenant OLAP Database Clusters

Jan Schaffner[1], Benjamin Eckart[1], Christian Schwarz[1], Jan Brunnert[1], Dean Jacobs[2], Alexander Zeier[1], and Hasso Plattner[1]

[1]Hasso Plattner Institute, University of Potsdam, August-Bebel-Str. 88, 14482 Potsdam, Germany, Email: {firstname.lastname}@hpi.uni-potsdam.de
[2]SAP AG, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany, Email: {firstname.lastname}@sap.com

Abstract: Simulation of parallel database machines was used in many database research projects during the 1990ies. One of the main reasons why simulation approaches were popular in that time was the fact that clusters with hundreds of nodes were not as readily available for experimentation as it is the case today. At the same time, the simulation models underlying these systems were fairly complex since they needed to capture both queuing processes in hardware (e.g. CPU contention or disk I/O) and software (e.g. processing distributed joins). Todays trend towards more specialized database architectures removes large parts of this complexity from the modeling task. As the main contribution of this paper, we discuss how we developed a simple simulation model of such a specialized system: a multi-tenant OLAP cluster based on an in-memory column database. The original infrastructure and testbed was built using SAP TREX, an in-memory column database part of SAP's business warehouse accelerator, which we ported to run on the Amazon EC2 cloud. Although we employ a simple queuing model, we achieve good accuracy. Similar to some of the parallel systems of the 1990ies, we are interested in studying different replication and high-availability strategies with the help of simulation. In particular, we study the effects of mirrored vs. interleaved replication on throughput and load distribution in our cluster of multi-tenant databases. We show that the better load distribution inherent to the interleaved replication strategy is exhibited both on EC2 and in our simulation environment.

1 Introduction

Implementing distributed systems and conducting experiments on top of them is usually both difficult and a lot of work is required to "get things right". When conducting research on a distributed system, such as for e.g. a multi-node database cluster, the turnaround time for changing an aspect of the system's design from implementation to testing is thus often high. At the same time, research on distributed systems is often experimental, i.e. the cycle of implementing and validating ideas on different system designs is repeated fairly often.

The simulation of software systems can serve as one possible tool to shortcut the evaluation of system designs, although it cannot replace building (and experimenting with) actual systems. Especially in the light of new hardware becoming available and being deployed

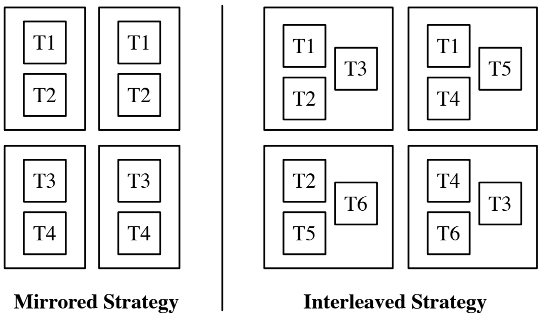


Figure 1: Example Layouts of Tenant Data

at cloud infrastructure providers simulation allows the prediction of cluster behavior based on predicted performance increases on a cloud platform. In fact, simulation models were a prominent means to evaluate parallel database systems such as Bubba [BAC$^+$90] and Gamma [DGS$^+$90] in the late 1980ies and 1990ies. These simulation models underlying these systems are fairly complex, since they capture the most important components of computer systems and their inter-dependencies, from the CPU and the main-memory sub-system to disk and network I/O, not to forget the multitude of software components involved in database query processing. The recent trend in database research towards specialized systems with simplified architectures [Sto08] does, however, also simplify the creation of simulation models.

In this paper, we describe our experience with building a simulation model for a multi-tenant OLAP cluster based on TREX, SAP's in-memory column database [Pla09, SBKZ08, JLF10]. TREX was designed to support interactive business intelligence applications that require a) sub-second response times for ad-hoc queries to facilitate exploratory analysis and b) incremental insertions of new data to provide real-time visibility into operational processes. In previous work, we ported TREX to run in the Amazon EC2 cloud [Ama] and built a clustering framework round TREX, called Rock, that supports multi-tenancy, replication, and high availability.

In-memory databases perform disk I/O only during write transactions to ensure durability. Our data warehousing workload is however read-mostly in the sense that writes occur only during ETL periods and have batch character. Also, column-databases are known be CPU-bound for scan-intensive workloads (such as for e.g. data warehousing) [SAB$^+$05]. All this allows us to build a much simpler simulation model which is yet accurate in comparison to execution traces of the real system.

Similar to some of the parallel systems of the 1990ies, we are interested in studying different replication and high-availability strategies with the help of simulation. This paper experimentally compares two data placement strategies for Analytic Databases in a Cloud Computing environment, mirroring and interleaving. Example layouts from these strategies are shown in Figure 1, where the large boxes represent databases and the small boxes within them represent data for individual tenants.

When using the mirrored strategy, two copies of each database are maintained, both of which contain the same group of tenants. To ensure acceptable response times during recovery periods, each server must have sufficient capacity to handle the entire workload on its own, thus the system must be 100% over-provisioned [MS03]. This strategy is used by many on-demand services today.

Two copies of each tenant's data are maintained, when using the interleaving strategy. The data is distributed across the cluster so as to minimize the number of pairs of tenants that occur together on more than one server. This strategy reduces the amount of over-provisioning that is required to handle failures and load surges because the excess work is distributed across many other servers.

We show that, without failures or variations in the request rate, the interleaved strategy achieves higher throughput than the mirrored strategy. For the moderately-sized tenants used in our experiments in the real system, the improvement is 7%. This improvement occurs because the interleaved strategy smoothes out statistical variations in the workload that depend on which queries are submitted to which servers. We wanted to make sure that this effect is a result of the chosen placement strategy and not a random effect coming from random variations in capacity of Amazon EC2 VMs. We therefore parameterized our simulator with a similar setup and were able to produce a similar result. We also evaluate the impact of server crashes for both mirrored and interleaving on the real cluster and using simulation.

This paper is organized as follows: Section 2 describes the Rock clustering infrastructure. Section 3 introduces the benchmark which was used for all experiments in this paper. Section 4 we analyze the requirements and discuss our implementation of our discrete event simulator based on the Rock clustering infrastructure and the benchmark. Section 5 discusses or data placement experiments both in the real system and the simulator. Section 6 discusses related work. Section 7 concludes the paper.

2 The Rock Framework

The Rock clustering framework runs in front of a collection of TREX servers and provides multi-tenancy, replication of tenant data, and fault tolerance. Figure 2 illustrates the architecture of the Rock framework. Read requests are submitted to the cluster by the analytics application. Write requests are submitted by the batch importers, which periodically pull incremental updates of the data from transactional source systems. The Rock framework itself consists of three types of processes: the *cluster leader*, *routers*, and *instance managers*. Each instance manager is paired one-to-one with a TREX server to which it forwards requests.

The cluster leader exists only once in the landscape and assigns tenant data to instance managers. The cluster leader as well as the batch importer are assumed to be highly available by replicating state using the Paxos[Lam98] algorithm, which would provide fail-safe distributed state for these critical components. The actual implementation is considered future work at this point. Each copy of a tenant's data is assigned to one instance manager

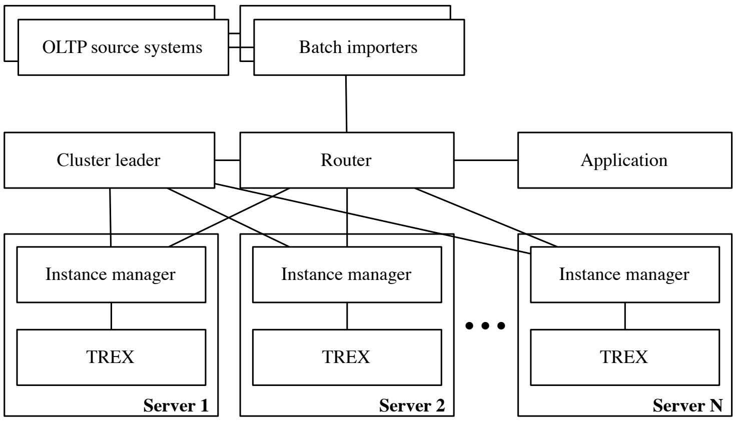


Figure 2: The Rock Analytic Cluster Architecture

and each instance manager is responsible for the data from multiple tenants. The cluster leader maintains the assignment information in a *cluster map*, which it propagates to the routers and instance managers so all components share a consistent view of the landscape. The cluster leader tracks changes to the state of the cluster based on information it collects from the Amazon EC2 API such as IP addresses, instance states, and geographic location. The cluster leader is not directly involved in request processing.

The routers accept requests from outside the cluster and forward them to the appropriate instance managers. Routing is based on the tenant who issued the query and the chosen load balancing strategy. Our current implementation supports round-robin, random, and server-load-based load balancing. The experiments in this paper use the latter algorithm. Load is taken to be the CPU idle time of the TREX server averaged over a 10 second window. The small window size is crucial for the router's ability to re-direct queries to the least utilized replica during a load burst. Load information is piggy-backed onto query results as they are returned to the router.

Rock offers master/master replication [GHOS96]: a router may forward a write request to any one of the instance managers for a tenant, which then propagates the write to the other instance managers for that tenant. We assume there is a single batch importer per tenant and that writes are sequentially numbered, thus master/master replication is straightforward to implement without introducing inconsistencies in the data. Read consistency is required to support multi-query drill down into a data set, and TREX implements it using multi-version concurrency control (MVCC) based on snapshot isolation [BBG+95].

According to [JA07], multi tenancy can be realized in the database by adopting a *shared-machine, shared-process,* or *shared-table* approach. The shared-table approach, where each table has a `tenant_id` column, can be made efficient if accesses are index-based. However analytic queries on column databases generally entail table scans, and scan times are proportional to the number of rows in the table. Rock therefore uses the shared-process approach and gives each tenant their own private tables.

3 Experiments on the Amazon EC2 Cloud

The experiments in this paper are based on a modified version of the Star Schema Benchmark (SSB) [OOC07], which is an adaptation of TPC-H [TPC].

To produce data for our experiments, we used the data generator of SSB, which is based on the TPC-H data generator. As stated in Section 2, we give each tenant their own private tables, thus there is one instance of the SSB data model per tenant. In the experiments presented in this paper, all tenants have the same size, i.e. 6,000,000 rows in the fact table. As a point of comparison, a Fortune 500 consumer products and goods enterprise with a wholesale infrastructure produces about 120 million sales order line items per year, which is only a factor of 20 greater than the tenant size chosen for this paper. Using TREX's standard dictionary compression, the fully-compressed data set consumes 204 MB in main memory.

While TPC-H has 22 independent data warehousing queries, SSB has four *query flights* with three to four queries each. A query flight models a drill-down, i.e. all queries compute the same aggregate measure but use different filter criteria on the dimensions. This structure models the exploratory interactions of users with business intelligence applications. We modified SSB so all queries within a flight are performed against the same TREX transaction ID to ensure that a consistent snapshot is used.

In our benchmark, each tenant has multiple concurrent *users* that submit requests to the system. Each user cycles through the query flights, stepping through the queries in each flight. After receiving a response to a query, a user waits for a fixed think time before submitting the next query. To prevent caravanning, each user is offset in the cycle by a random amount.

The number of users for a given tenant is taken to be the size of that tenant multiplied by a scale factor. Our experiments vary this scale factor to set the overall rate of requests to the system. In reporting results, we give the maximum number of simultaneous users rather than the throughput, since users are the basis of pricing and revenue in the Software as a Service setting. TPC-DS also models concurrent users and think times [PSKL02]. Following [SPvSA07], which studies web applications, we draw user think times from a negative exponential distribution with a mean of five seconds.

A benchmark run is evaluated as follows. The first ten minutes are cut off to ensure that the system is warmed up. The next ten minutes after the warmup are called the benchmark period. All queries submitted after the benchmark period are cut off as well. A run of the benchmark is considered to be successful only if, during the benchmark period, the response times at the 99-th percentile of the distribution are within one second. Response times are measured at the router. Sub-second response times are essential to encourage interactive exploration of a dataset and, in any case, have become the norm for web applications regardless of how much work they perform. The focus on performance at the 99-th percentile is also common; see [DHJ$^+$07] for example.

The results presented in this paper are highly dependent on specific configuration choices described in this section. Nevertheless we believe these results are applicable in most practical situations. Our tenants are relatively large by SaaS standards and, for smaller tenants, interleaving would distribute excess work more evenly across the cluster. Five second think times are perhaps too short for more complex applications, but the system behaves linearly in this respect: doubling the think time would double the maximum number of simultaneous users.

All experiments are run on *large memory* instances on Amazon EC2, which have 2 virtual compute units (i.e. CPU cores) with 7.5 GB RAM each. For disk storage, we use Amazon EBS volumes, which offers highly-available persistent storage for EC2 instances. The disks have a performance impact only on recovery times. An EBS volume can be attached to only one EC2 instance at a time.

4 Simulation Model

For the simulation, we need to model the real system and benchmark, which have been described in Sections 2 and 3. In this section, we analyze the requirements and discuss our implementation of a discrete event simulator.

4.1 Problem Statement

Given a special-purpose clustering framework (Rock) and a commercial in-memory database system (TREX), we can assess the viability of using simulation techniques to estimate the performance characteristics exhibited by such a system. The goal is to accurately model the most relevant environmental parameters as well as the different load balancing, data placement and high availability techniques employed in the real cluster system.

The simulation should provide results that allow a relevant assessment of various strategies in the context of a cluster setup. The accuracy of the simulation results shall be validated against the empirical results for a static cluster configuration. The simulation does not take into account message passing latency between system components or network bandwidth, but focuses on the kernel execution time of the in-memory column database, which is composed of CPU execution time and time waiting for the operating system to schedule a CPU for the execution thread.

We will begin with discussing the fundamentals of the simulation model, such as the modeling of the query processing components and the user load model as well as describing the implementation of the simulator. The simulation results will be presented in the following section.

4.2 Simulation Model of the In-Memory Database Cluster

Discrete Event Simulation using a process-oriented paradigm allows an integrated simulation of the most important components and processes in the cluster. In a process-oriented simulation model, different active components are modularized in processes. The execution of parallel processes is serialized by explicit wait statements that allow simulation time to skip ahead to the next occurring event. This approach is more modular and CPU efficient than the activity oriented paradigm and, therefore, allows simulation of user activities using a more fine-grained queuing model of the involved components and their users.

The simulation model consists of *resources* and *processes*. In the case of the simulated Rock cluster, the resources are compute *Nodes* (virtual machine instances running instance manager/TREX pairs). Nodes have an immutable number of processors and amount of main memory. It is assumed that these virtual machines are used for serving database requests exclusively. Queues are established when simulation processes need to wait for a shared resource.

Processes are actors within the simulator. For example, the activity of a single user, of which there are multiple for each tenant data-set, is modeled as a *User* process. Multiple processes of the User type are active in parallel. Users create *Query* processes that simulate the execution of queries on the limited Node resources.

To simulate a behaviour of a system, we have to understand and model the behaviour of a system. One common approach to model a system is *Queueing Network Modeling*. According to Lazowska et al., *Queueing Network Modeling* is a an approach in which a computer system is represented as a network of queues which is evaluated analytically [LZGS84]. A network of queues consists of several *service centers* which are system resources and are used by *customers* that are the users of the system. If customers arrive at a higher rate than the service center can handle, customers are queued. The time which is necessary for a transaction to be finished is, then, not only the time the service center requires, but also the waiting time in the queue.

If more queries arrive than can be executed by all query threads, subsequent queries will be queued. In a queueing network model, each query thread is represented by a service center. Each query thread uses one of the two CPUs which itself are represented by two further service centers. The threads are sharing the processing unit resources using a time slice model.

4.3 Modeling Query Processing Components

The goal of the simulation is to model a cluster of in-memory columnar database instances. The cluster's response time profile has been studied empirically using the SSB benchmark, which yields as raw data the query processing times for individual requests. At the core of the discrete event simulation is the statistical model of the kernel execution times, or service-center processing times, based on query type. For the purpose of establishing the internal processing times, we have analyzed a long-running benchmark on the experimental framework with only a single user in order to establish a baseline without queuing interference. Based on this data we determined which statistical distribution best matches the real distribution. In general, one often uses exponential distributions for "neutral" simulation-to-simulation comparisons of scenarios, because the exponential distribution has favorable properties in regards to calculations. However, for modeling our Rock cluster infrastrcture, it turns out that the gamma distribution is the best choice.

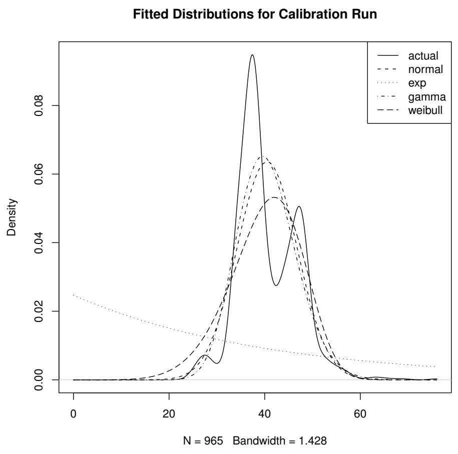


Figure 3: Distribution Fitting for Query 2.2

As can be seen for the example of SSB Query 2.2 in Figure 3, the distribution of query times from the single-user baseline run shows that the real distribution resembles a head-and-shoulders pattern, but has a strong peak around the mean processing time. We observe similar peaks for all other SSB query types as well. The diagram also shows how various distributions are fitted to the query.

We model the processing times of all queries fitting a gamma distribution to each query type using different parameters for shape k and scale Θ. Table 1 shows the corresponding parameters for each query. The following equation shows the gamma distribution:

$$f(x; k, \Theta) = x^{k-1} \frac{e^{-x/\Theta}}{\Theta^k \Gamma(k)}, \quad \text{for } x \geq 0 \text{ and } k, \Theta > 0 \tag{1}$$

One essential challenge is that the distribution of response times in the real cluster contains spikes, whereas statistical distributions typically look smooth. The gamma distribution is useful for our scenario, as it best resembles most of the queries shapes and allows us to smooth out the smaller spikes occuring in the real system. Still, the sample space remains usable because a greater variation is introduced around the mean due to the continuous random sampling in the simulator, which imitates the effect of the discrete hot-spots in the real system. The distribution drives a separate random number generator for each Node to generate internal kernel execution times for each query type. Because the sampled times from the calibration run are gross times that include networking and processing overheads, which are not part of the actual internal service center times, we establish an internal speed-up factor[1], which is variably adapted for the baseline test, that shifts the distribution in favor of a faster internal execution, while preserving the system-inherent distribution characteristics. This approach has been superior to using a distribution based on the minimum response time, which did not accurately reflect the overheads resulting in occasional processing slowdowns in the real systems.

	Shape k	**Scale Θ**
Query 1.1	343.794	2.685
Query 1.2	18.452	0.685
Query 1.3	3.547	0.54
Query 2.1	188.744	2.257
Query 2.2	42.997	1.061
Query 2.3	15.319	0.564
Query 3.1	379.154	2.525
Query 3.2	96.046	1.595
Query 3.3	13.693	0.568
Query 3.4	12.536	0.529
Query 4.1	311.531	2.28
Query 4.2	70.306	0.636
Query 4.3	122.473	1.705

Table 1: Gamma Distribution Parameters for SSB Queries

4.4 Simulation Accuracy

In order to evaluate our simulation model's accuracy we use a benchmark trace taken from experiments on a Rock cluster instance running on Amazon EC2 and compare the trace against the output of our simulation, which mimics the real system trace output. A trace contains query response times for all tenants' users' queries submitted during a 900 second test period, from which all data after a warmup period of 300 seconds is analyzed.

When comparing non-aggregated query execution times as shown in the Q-Q plot in Figure 4, one can see that the plot forms an almost identical line with the reference line, indicating that the individual query times generated by the simulation come from the same distribution as the execution times in the empirical system. This also validates our assumption that by closely modeling the underlying internal execution with a statistical distribution, reducing these times by measured overhead, and then adding queuing-theoretic waiting times, we can model the multi-tenant cluster with good accuracy.

The fact that the execution time plot is slightly above the reference line for faster queries shows that the real system has a larger fractional overhead for smaller queries than we are actually simulating. For a minority of slow queries, the simulator again returns too fast response times, indicating that these have a larger overhead on the real system, in spite of

[1]The factor for the comparisons in this paper was 0.85

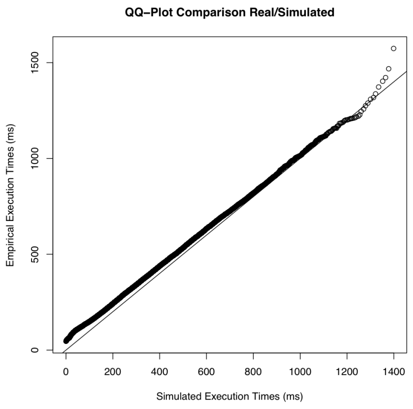


Figure 4: QQ-plot of simulated query times and actual cluster query times

our threading implementation, but this only applies to few queries. Nevertheless, the plots show that the simulator produces a very close match of the distribution indicated by the straight nature of the plot.

Therefore we can support our implementation goal to have a simulator that is accurate enough to enable the comparison of multiple scenarios with varying parameters and configurations to each other, while maintaining a close match to results obtained from real systems. This result is possible due to the very predictable performance characteristics of in-memory databases, due to the absence of complicated disk I/O scheduling.

Nevertheless, real systems have many influencing factors, which require re-calibration of the simulator after major changes in the underlying database software or virtualized PaaS environment. As a consequence, effects discovered in simulation still need to be backed by experiments on the real system.

	Simulation	**Real execution**
# Users	3500 users	4000 users
# Queries	397265	341560
Mean response time	305.9 ms	338.6 ms

Table 2: Maximum Number of Concurrent Users Before SLO Violation

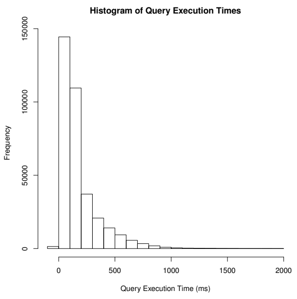
We can see in the Table 2 that the number of queries in a given period of time is higher in the simulation than in the real test. The reason is that we are using an idealized model which is a strong simplification of the system. For example, some locking interdepencies that might cause queries to stall in the real system are not captured in the simulation model. We only model a single queue in front of the processors, which is not accurate since there is also queuing around network resources.

In summary, the response times of individual queries are accurately reproduced by the simulator, as shown in the Q-Q plot in Figure 4, while the total number of queries executed within a benchmark run and the maximum number of users in the system before violating the response time goal are not perfectly aligned. However, the goal of our simulator is to test different cluster deployment options such as for e.g. mirrored vs. interleaved replica placement. We are interested in the relative performance difference of a choice of deployment options in the simulator. Our simulation model is suitable for this purpose, as we shall see in the next section.

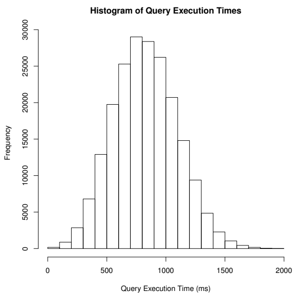
5 Simulation Results

In this section, we analyze the simulation results and compare them with measurements conducted on our real system.

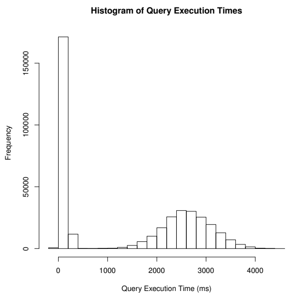
5.1 Distribution of Response Times Under Low and High Load



(a) Distribution in non-overload situation with threading



(b) Distribution in overload situation without threading



(c) Distribution in overload situation with threading

Figure 5: Query time histogram (frequency over response time in ms) for normal behavior (a), overload (b) and overload with threading (c)

The behavior of the simulated database cluster is greatly influenced by the cluster's capacity in terms of available CPU resources. Without load spikes, the system shows a response time profile as shown in the histogram in Figure 5(a), looking much like an exponential distribution with many fast queries. The simulator can simulate time-slice scheduling, and we are currently using a time slice of 30 ms, modeled after the scheduling quantum of the adapted Xen environment of Amazon EC2. Of course, scheduling in real systems would also include the priority-based scheme of the operating system, with more complex interactions between processes, but in our experimental research system all processes are CPU-bound and not mixed with I/O bound tasks on the same kernel instance, allowing us to get our good matches between real and simulated benchmarks using the fixed time slice. Yet, when threading is enabled, the absolute number of queries processed during the overall simulation run is much higher (528413 vs 351422 queries), but the mean processing time is also higher (180 ms vs. 125 ms mean). Threading allows faster queries to fast-track slow-running queries and therefore increases overall throughput at the expense of execution speed for slower queries.

When looking at a simulated run where we have increased the number of total users hitting the system at once beyond the capacity limit, the benefit of admitting more queries at once is clearly visible. While in Figure 5(b) the overload in the dedicated CPU system produces a load profile that is shaped like a normal distribution with a much higher mean. This is due to the fact that too many queries are queuing up in the system and are pushing all following queries up in their response time. The shared CPU run with 4 threads on 2 simulated CPUs in Figure 5(c) shows clearly that smaller queries are still being processed quickly, while only the slower queries suffer from the overload. Therefore, threading clearly reduces the visible latency for users with fast queries at the expense of those with slower queries.

Valuable insight can be gained from the behavior under overload conditions, when the resources required for all user requests exceed the available capacity. The way the system behaves in such overload conditions depends on system configuration parameters such as the maximum response time before a query is considered to have failed or how quickly additional resources can be acquired from the underlying cloud infrastructure. Another fundamental decision is whether to enable simulated time-slice multitasking in such a CPU-bound processing problem. The real system uses a maximum of four computing threads on a system with two virtual processors, therefore overcommitting the CPU resource while at the same time throttling the maximum number of parallel requests in the execution state. This two-times overcommitment has been shown to deliver the best results for the SSB workload and is explained by the fact that generated plan operations synchronize well before the activation of the next plan step.

5.2 Distribution of Response Times in the Presence of Failures

As stated in Section 2, Rock uses an active/active load balancing scheme in the presence of multiple replicas. If a server goes down, the workload which was handled by the crashed server is re-distributed to the servers holding the other copy of the tenants' data. The redistribution of workload in the event of a server failure differs depending on how the tenant replicas are assigned to the servers in the cluster.

Using the off-the-shelf replication capabilities as offered by most modern databases would result on replicating the data on the granularity of a whole server. In doing so, all tenants appearing together on one server will also co-appear on a second server in the cluster. This technique is often referred to as *mirroring* (cf. Figure 1). The downside of mirroring is that in case of a failure all excess workload is re-directed to the other mirror server. In that case, the mirror server becomes a local hotspot in the cluster until the failed server is back online. A technique for avoiding such hotspots is to use *interleaving*, which was first introduced in Teradata [Ter85]. Interleaving entails performing replication on the granularity of the individual tenants rather than all tenants inside a database process. This allows for spreading out the excess workload in case of a server failure across multiple machines in the cluster.

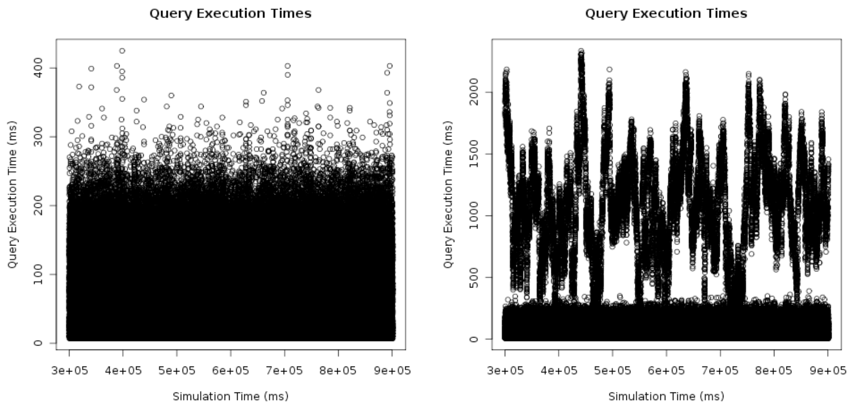
The following experiment in the real system demonstrates the impact of the chosen replica placement strategy on a cluster's ability to serve queries without violating the SLO both during normal operations and failures: We set up a cluster with 100 tenants, where we put 10 tenants on each server. All tenants had exactly the same size (6 million rows in the fact) table and there were two copies per tenant, hence 20 servers in total. We assigned the tenant replicas to the server both using the mirrored strategy, where groups of 10 tenants where mirrored on one pair of servers each, and the interleaved strategy, where we manually laid out the tenants such that no two tenant replicas appear together on more than one server. Automatic generation of interleaved placements and incremental self-configuration of the cluster is ongoing research in our group and not in the scope of this paper, but discussed in our work on performance prediction[SEJ$^+$ar]. We than ran both placement configurations under normal conditions and under failures. In the failure case, 1 out of the 20 TREX instances in the cluster was killed every 60 seconds. Given the average recovery time in our experiment, 1 out of 20 servers was thus unavailable for approximately 50% of the benchmark period in the failure case. Note that this is a very high failure rate which is unlikely to occur in practice.

Table 3 shows the results of the experiment on the EC2 cluster. Even under normal operating conditions, interleaving allows for 7% more throughput before the response time goal of one second in the 99th percentile is violated. The reason is that statistical variations occur when the number concurrently active users is high. These variations create short-lived load spikes, which the interleaved configuration spreads out better in the cluster than mirroring. As expected, the maximum throughput that the mirrored configuration can sustain in the failure case before an SLO violation occurs drops by almost 50% when compared to normal operations. Interleaving, in contrast, completely hides the failure from a throughput perspective. Notably, the interleaved configuration can even support 32 more users than the mirrored configuration without failures.

	Mirrored	**Interleaved**	**Improvement**
Normal operations	4218 users	4506 users	7%
Periodical failure	2265 users	4250 users	88%

Table 3: Maximum Number of Concurrent Users Before SLO Violation

On the real system in the Amazon EC2 cloud it could be shown that the layout, meaning how tenants are placed on the nodes in the cluster, had an impact on system performance, especially in the event of failures. We are interested in proving that this effect is a real property of the system, rather than a random effect which stems from external factors, such as for example non-uniformity in the capacity of the virtual machines procured by EC2. To do so, we enhance the queuing model of the simulator to model node crashes: The *Fault* is modeled as an optional process that can inject fault events into the query process based on its own failure distribution model. In our case we chose to inject failures at static times during the simulation, although an exponential distribution could also be used in repeated experiments to study the independence of failure behaviors and failure time.

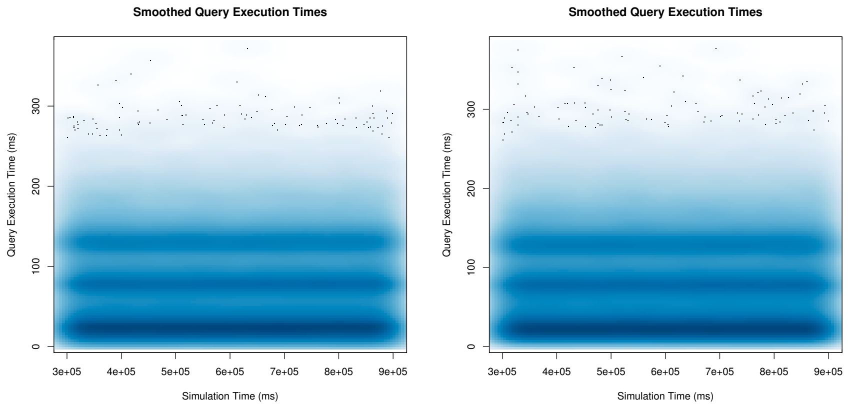


(a) mirrored/no failure (b) mirrored/failure

Figure 6: Simulated response times (ms) over simulation time (ms) for mirrored placement without failure(a), with injected failure (b)

Figure 6 shows the trace of query response times produced by our simulator using mirrored replica placement. During normal operations, as shown in Figure 6(a), the query load is distributed evenly among the nodes to which a tenant is assigned. Almost all queries are executed in less than 300 ms. Figure 6(b) shows the trace of a simulation run where 1 out of 20 nodes was failed at the beginning of the simulation. After the failure event, all queries were sent only to the remaining node until the failed node was unavailable. The simulation does not drop queries after a timeout but rather tries to process all queries to completion. As can be seen in Figure 6(b), the system does not recover from the 30 second failure injected at the beginning of the simulation using a mirrored configuration. While most of the queries are still executed in less than 300 ms, there is a considerable number of queries in the system which take up to 2 seconds to execute. This is a result of the two nodes affected by the failover trying to "catch up" with the query load. The load exceeds the capacity of the two nodea and therefore the amount of lost work cannot be regained. This leads to very high response times for the affected tenants, which can be seen in the scatterplot, which negatively affect the mean response time.

Figure 7 shows simulator response time traces based on the interleaved layout. Figure 7(a) shows the same behavior as the mirrored setup under normal operating conditions, therefore there is no inherent disadvantage to an interleaved setup. In fact, the mirrored configuration processes 359181 simulated queries with a 78 ms mean response time, while the interleaved setup processes slightly more queries (359577) with a slightly better mean response time of 72 ms. The improvement of the mean response time in the interleaved setup amounts to 8%. Although both numbers are not directly comparable, recall that a 7% improvement in throughput was observed in the real system under normal operating conditions when using an interleaved layout. As it can be seen in Figure 7(b), the injecting a failure has almost no impact for the interleaved configuration. The mean response time is 75 ms in spite of the failure, which is still lower than the mean response time in



(a) interleaved/no failure (b) interleaved/failure

Figure 7: Simulated response times over time for interleaved placement without failure(a), with injected failure (b)

the non-failure mirrored configuration. The mean response time for the mirrored configuration with failure was 158. This 50% drop in performance is also consistent with our experiments on the EC2 cluster.

In this section, we have argued that the performance effects of deployment choices such as mirrored vs. interleaved replica placement can be shown using simulation. The simulation environment is however less complex than the real system. While changing the replication strategy in the real cluster is tedious to implement, changes to the simulator can be done much faster. Therefore, we see simulation as a tool for fast exploration of cluster configuration trade-offs, using which we are able to identify what configurations are worth implemented in our EC2 environment.

6 Related Work

Discrete event simulation has been successfully applied to many research areas in computer science [Cas05]. Discrete event simulation is based on the idea, that the observed system can be modeled as queuing network processes. Events created within or outside the system are based on a reference dataset. Therefore, reference datasets heavily influence the accuracy of the simulation.

Many standard tools supporting the design of such simulations have been made available to the research community, such as simulation languages (i.e. SimScript[KVMR75]) and simulation libraries like Simjava[HM98] or SimPy[Tea06] as used in our project. Frameworks and underlying techniques are continuously improved towards higher simulation accuracy and performance. In order to speedup the simulation methods for distributed an parallel discrete event simulations have been developed [FT96]. One actual framework to be mentioned here is Parsec, which aims at parallel simulation of complex systems[BMT+98].

A wide variety of application specific simulators have been developed that incorporate the specifics of certain technologies, such as large-scale wired or wireless networks [ZBG98]. In the study of specific network-intensive workloads Saidi et al. "determine how accurately we can model the overall behavior of a complex system using appropriately tuned but relatively generic component models" [SBHR05], an approach we are trying to adapt for the modeling of the well-predictable in-memory execution components.

Also, widely distributed, job-oriented grid computing environments and the effects of scheduling on overall grid performance have been studied using simulation based on the modeling of applications [Cas02, BM02]. Many other examples exist, each showing that for the comparison of scenarios and the study of the impact of parameters simulation remains a verifiable complementing activity to empirical study.

Distributed systems and in particular Web server farms have previously been studied using simulation. Particularly similar to our method of studying in-memory database systems, which resemble dynamic web page generation in their CPU-bound nature, Teo studied the effect of load balancing strategies in clusters using simulation [TA01]. In this work, the client and Web server service times used in the simulator were also determined by carrying out a set of experiments on an actual testbed.

More specifically in the field of distributed database systems simulation has been used on a micro-level to study the performance of operators in a CPU-bound context [MD95], where it can also be seen that a CPU-bound application profile is always preferred in simulation, and that closed-loop systems always follow similar patterns in modeling user think times in their load model.

The data placement problem for relations has been previously studied in the context of parallel databases with large relations and a fixed cluster size. The Bubba parallel database prototype uses a statistics-driven variable declustering algorithm, that takes the access frequency and size of the relations into account [CABK88]. It therefore focuses on a single tenant placement problem within a fixed cluster of nodes and shows that load-balancing improves with increasing declustering. The prototyping of the Bubba system was supported by a simulation to "accurately predict the scalability of Bubba's performance over the entire range of configuration sizes" [BAC+90]. A comprehensive simulation study of data placement in shared-nothing systems [MD97] has been conducted to find a consensus on the most efficient placement algorithm, following previous simulation studies specialized on data placement strategies such as multi-attribute declustering [GDQ92].

An autonomic and self-tuning cloud-based data warehousing framework has been described in our work on performance prediction[SEJ+ar]. By applying a load model to the entire cluster state, the framework can automatically conduct administrative actions on the cluster to optimize overall performance. Even though existing systems often contain self-management components that optimize threading, query admission and memory allocation, these systems do not consider data placement in a dynamically sized cluster in a multi-tenant context, where incremental re-organization is required rather than "big-bang" reorganization. Also, our research does not focus on distributing large relations, but rather on heuristics for optimal redistribution of small relations. Also, the case for a cloud database service provider requires that optimization is not for minimizing response times but for maximizing utilization of resources under response-time constraints.

7 Conclusion

In this paper, we have presented the implementation of a simulation of a static cluster serving multiple tenants with analytic database services. The simulation results have been evaluated against the real system results and show that the simulator delivers adequate results for the evaluation of scenarios, such as failure conditions or overload. This simulation data can be used for system planning and design, for the detection of unexpected runtime behavior of real-life systems or to identify and validate hypotheses on multi-tenant database systems. Especially for the validation of proposed Service Level Agreements, simulation can compare many scenarios in parallel and compare the resulting economic benefit. For autonomic systems, a simulator can be used to train artificial intelligence algorithms, such as neural networks, in much shorter time and at lower cost, than using a real system. Also, a simulator has predictable runtime behavior that is not influenced by the measurement itself. Therefore simulation complements the empirical study of real systems in many useful ways. In our particular case, we could show that the interleaved placement performs much better than the mirrored placement when failures occur using a real experiment setup as well as using the simulation.

Future work on the simulation will involve the integration of cluster control in the Rock framework, with live system visualization and the simulation facilities. This might require the enhancement of the simulator to include on-line placement of tenants, cluster expansion, and memory resource management on an individual tenant basis. Additional simulation enhancements could include the simulation of merging the columnar data structures [KGT$^+$10] using the simulated threading to study the impact of such maintenance tasks on the cluster performance, especially when considering the trade-off of losing excess capacity in the cluster vs performance improvements yielded by the merge. Another very interesting enhancement of the simulation would be to include the impact of disk I/O resource contention when using dynamic loading of inactive tenants to main memory or in situations where failures require re-loading the data from disk.

On the query simulation end, the support for delta-table performance impact simulation and its resulting write performance penalty because of queuing disk I/O for log-writing is regarded as future work, as is the impact of network communication overhead for multi-node joins.

Generally, the simulation evaluation component could be extended to apply the results to various SLA scenarios to calculate a profit or cost, which could be a basis to compare various configurations to each other on the basis of a single monetary figure. The dynamically adaptable cloud computing environment is especially suited for such a cost model, because the resources in the cluster have a clearly defined pricing based on their usage and the financial profile of each simulated scenario heavily depends on its computing resource allocation and actual usage. The difficulty in such an assessment lies in the fact, that today's SaaS offerings usually define neither clear agreements in terms of clear boundaries for cases when the reliability of the service is insufficient (other than complete unavailability) nor any failure indemnification reimbursement policies which would make such an SLA-based loss-reduction calculation possible.

References

[Ama] Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2.

[BAC+90] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez. Prototyping Bubba, a highly parallel database system. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):4–24, 1990.

[BBG+95] Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. A Critique of ANSI SQL Isolation Levels. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 1–10, New York, NY, USA, 1995. ACM.

[BM02] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[BMT+98] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song. Parsec: A parallel simulation environment for complex systems. *Computer*, pages 77–85, 1998.

[CABK88] G Copeland, W Alexander, E Boughter, and T Keller. Data placement in Bubba. *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 99–108, 1988.

[Cas02] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 430–437. IEEE, 2002.

[Cas05] CG Cassandras. Discrete-Event Systems. *Handbook of networked and embedded control systems*, pages 71–89, 2005.

[DGS+90] D. J. Dewitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H. I. Hsiao, and R. Rasmussen. The Gamma Database Machine Project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, 1990.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-Value Store. In *SOSP*, pages 205–220, 2007.

[FT96] A Ferscha and SK Tripathi. Parallel and distributed simulation of discrete event systems. *Parallel and distributed computing handbook*, pages 1003–1041, 1996.

[GDQ92] S Ghandeharizadeh, DJ DeWitt, and W Qureshi. A performance analysis of alternative multi-attribute declustering strategies. *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 29–38, 1992.

[GHOS96] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The Dangers of Replication and a Solution. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 173–182, New York, NY, USA, 1996. ACM.

[HM98] F. Howell and R. McNab. simjava: a discrete event simulation package for Java with applications in computer systems modelling. In *Proceedings of the First International Conference on Web-based Modelling and Simulation*, 1998.

[JA07] Dean Jacobs and Stefan Aulbach. Ruminations on Multi-Tenant Databases. In *BTW*, pages 514–521, 2007.

[JLF10] Bernhard Jaecksch, Wolfgang Lehner, and Franz Faerber. A plan for OLAP. In Ioana Manolescu, Stefano Spaccapietra, Jens Teubner, Masaru Kitsuregawa, Alain Léger, Felix Naumann, Anastasia Ailamaki, and Fatma Özcan, editors, *EDBT*, volume 426 of *ACM International Conference Proceeding Series*, pages 681–686. ACM, 2010.

[KGT+10] Jens Krueger, Martin Grund, Christian Tinnefeld, Hasso Plattner, Alexander Zeier, and Franz Faerber. Optimizing Write Performance for Read Optimized Databases. In *Database Systems for Advanced Applications, Japan*, 2010.

[KVMR75] P.J. Kiviat, R. Villanueva, H.M. Markowitz, and E.C. Russell. *SIMSCRIPT II. 5 programming language*. CACI, 1975.

[Lam98] Leslie Lamport. The Part-Time Parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[LZGS84] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1984.

[MD95] M. Mehta and D.J. DeWitt. Managing intra-operator parallelism in parallel database systems. In *PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES*, pages 382–394. INSTITUTE OF ELECTRICAL & ELECTRONICS ENGINEERS (IEEE), 1995.

[MD97] M Mehta and DJ DeWitt. Data placement in shared-nothing parallel database systems. *The VLDB Journal*, 6(1):53–72, 1997.

[MS03] E Marcus and H Stern. *Blueprints for High Availability*. Wiley, 2003.

[OOC07] P. E. O'Neil, E. J. O'Neil, and X. Chen. The Star Schema Benchmark (SSB), 2007. http://www.cs.umb.edu/p̃oneil/StarSchemaB.PDF.

[Pla09] Hasso Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul, editors, *SIGMOD Conference*, pages 1–2. ACM, 2009.

[PSKL02] Meikel Poess, Bryan Smith, Lubor Kollar, and Paul Larson. TPC-DS, Taking Decision Support Benchmarking To The Next Level. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 582–587, New York, NY, USA, 2002. ACM.

[SAB+05] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. C-store: a column-oriented DBMS. In *Proceedings of the 31st international conference on very large data bases*, pages 553 – 564, 2005.

[SBHR05] A.G. Saidi, N.L. Binkert, L.R. Hsu, and S.K. Reinhardt. Performance validation of network-intensive workloads on a full-system simulator. In *Proc. 2005 Workshop on Interaction between Operating System and Computer Architecture (IOSCA)*, pages 33–38. Citeseer, 2005.

[SBKZ08] Jan Schaffner, Anja Bog, Jens Krüger, and Alexander Zeier. A Hybrid Row-Column OLTP Database Architecture for Operational Reporting. In *BIRTE (Informal Proceedings)*, 2008.

[SEJ+ar] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier. Predicting In-Memory Database Performance for Automating Cluster Management Tasks. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, to appear.

[SPvSA07] Swaminathan Sivasubramanian, Guillaume Pierre, Maarten van Steen, and Gustavo Alonso. Analysis of Caching and Replication Strategies for Web Applications. *IEEE Internet Computing*, 11(1):60–66, 2007.

[Sto08] Michael Stonebraker. Technical perspective - One size fits all: an idea whose time has come and gone. *Commun. ACM*, 51(12):76, 2008.

[TA01] Y.M. Teo and R. Ayani. Comparison of load balancing strategies on cluster-based web servers. *SIMULATION-CALIFORNIA-*, 77(5/6):185–195, 2001.

[Tea06] S.P.D. Team. Simpy homepage. *http://simpy. sourceforge. net/,[Last accessed*, 18(03):2007, 2006.

[Ter85] DBC/1012 Database Computer System Manual Release 2. Teradata Corporation Document No. C10-0001-02, 1985.

[TPC] TPC-H. `http://www.tpc.org/tpch/`.

[ZBG98] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. *ACM SIGSIM Simulation Digest*, 28(1):154–161, 1998.