Providing High Availability and Elasticity for an In-Memory Database System with RAMCloud

Christian Tinnefeld, Daniel Taschik, Hasso Plattner
Hasso-Plattner-Institute
University of Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

Abstract: Stanford's RAMCloud is a large-scale storage system that keeps all data in DRAM and provides high availability as well as a great degree of elasticity. These properties make it desirable for being used as the persistence for an in-memory database system. In this paper, we experimentally demonstrate the high availability and elasticity RAMCloud can provide when it is being used as a storage system for a relational in-memory database system: a) We utilize RAMCloud's fast-crash-recovery mechanism and measure its impact on database query processing performance. b) We evaluate the elasticity by executing a sinus-shaped, a plateau, and an exponential database workload. Based on our experiments, we show that an in-memory database running on top of RAMCloud can within seconds adapt to changing workloads and recover data from a crashed node - both without an interruption of the ongoing query processing.

1 Introduction

The storage capacity of an in-memory database management system (DBMS) on a single server is limited. A shared-nothing architecture enables the combined usage of the storage and query processing capacities of several servers in a cluster. Each server in this cluster is assigned a partition of the overall data set and processes its locally stored data during query execution. However, deploying a shared-nothing architecture comes at a price: scaling out such a cluster is a hard task [CJMB11]. Chosen partitioning criteria have to be reevaluated and potentially changed. While automated solutions exist to assist such tasks, often manual tuning is required to achieve the desired performance characteristics.

Recent developments in the area of high-performance computer networking technologies narrow the performance gap between local and remote DRAM data access to and even below a single order of magnitude. For example, InfiniBand specifies a maximum bandwidth of up to 300 Gbit/s and an end-to-end latency of 1-2 microseconds while an Intel Xeon Processor E5-4650 has a maximum memory bandwidth of 409.6 Gbit/s and a main memory access latency of 0.1 microsceonds. The research project RAMCloud [OAE+11] demonstrates that the performance characteristics of a high-performance computer network can be exploited in a large distributed DRAM-based storage system to preserve the narrowed performance gap between local and remote data access at scale.

Based on these premises, we use RAMCloud as the persistence layer for our prototypi-

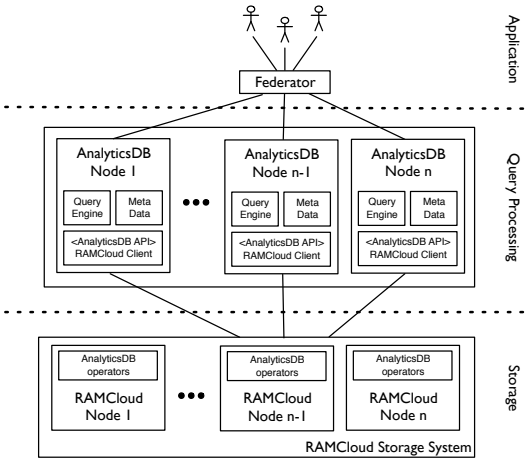

Figure 1: Architectural Overview

cal in-memory database system AnalyticsDB. The intention is to evaluate to what extent an in-memory database application can leverage the high availability and elasticity features provided by RAMCloud. Consequently, the remainder of the paper is structured as follows: Section 2 provides an architectural overview and explains AnalyticsDB and RAMCloud in detail. The aspect of high availability is much discussed in the context of database management systems (see related work in Section 5), however the aspect of elasticity is not: therefore, we discuss in Section 3 the term elasticity in the context of a database management system. Section 4 presents the actual experiments: the section is divided into high availability and elasticity experiment subsections. Section 5 lists the related work, Section 6 closes with the conclusions.

2 Architectural Overview

As introduced in previous work [TKG+13], our system consists of a set of AnalyticsDB nodes which access a shared storage layer established by a RAMCloud storage system.

Figure 1 depicts the architectural overview of our system: AnalyticsDB nodes receive application queries dispatched by a central federator. Every query is assigned to a single AnalyticsDB node, while a local query processor controls its execution. Each AnalyticsDB node holds the meta data describing the relational structure of all data contained in the storage layer to allow for query validation and planning. The query processor accesses the RAMCloud storage system through a RAMCloud client component that transparently maps the AnalyticDB API to operations on specific RAMCloud nodes.

The RAMCloud in-memory storage system consists of multiple nodes and takes care of

transparently distributing the stored data among participating nodes and manages replication, availability, and scaling of the cluster. To allow for push-down of operations to the storage layer, each RAMCloud node is extended by a set of AnalyticsDB operators.

In the following subsections the AnalyticsDB and RAMCloud systems are described in more detail.

2.1 AnalyticsDB

AnalyticsDB is our prototypical in-memory DBMS written in C++. It is built based on the common design principles for online analytical query engines such as MonetDB [BKM08] or SAP HANA [FML+12]. Data is organized in a column-oriented storage format and dictionary compressed. AnalyticsDB uses dictionary compression for non-numeric attributes (e.g. string) and encodes them to int64 values. This results in only integer values being stored in RAMCloud. The dictionary is kept on the AnalyticsDB nodes. The operators of our execution engine are optimized for main memory access and allow to achieve a high scan speed. In addition, our system defers the materialization of intermediate results until it becomes necessary following the pattern of late materialization. The biggest difference from our prototype system to a text-book analytical database system is how the execution engine operates on main memory. Instead of directly accessing the memory, e.g. by using the memory address, we introduced the AnalyticsDB API to encapsulate storage access. This API can be implemented by e.g. using a local data structure or by using the client of a separate storage system such as the RAMCloud client.

AnalyticsDB is designed for online analytical processing (OLAP) allowing data analysts to explore data by submitting ad-hoc queries. This style of interactive data analysis requires AnalyticsDB to execute arbitrary queries on multi-dimensional data in sub-seconds. For our experiments we use the Star Schema Benchmark (SSB) [O'N] [OOCR09]. The Star Schema Benchmark data model represents sales data and consists of Customer, Supplier, Part, and Date dimension tables and a Lineorder fact table.

2.2 RAMCloud

RAMCloud is a general-purpose storage system that keeps all data entirely in DRAM by aggregating the main memory of multiple commodity servers at scale [OAE+11]. All of those servers are connected via a high-end network such as InfiniBand which provides low latency [ROS+11] and a high bandwidth. RAMCloud employs randomized techniques to manage the system in a scalable and decentralized fashion and is based on a key-value data model: in the remainder of the paper we refer to the data items in RAMCloud as key-value pairs or objects. RAMCloud scatters backup data across hundreds or thousands of disks or SSDs, and it harnesses hundreds of servers in parallel to reconstruct lost data. The system uses a log-structured approach for all its data, in DRAM as well as on disk; this provides high performance both during normal operation and during recovery [ORS+11].

3 Elasticity

Elasticity, in a database context, has recently emerged with the rise of cloud computing. In the world of traditional relational DBMS, elasticity is mostly unknown and has not earned a lot of attention. As defined by Minhas et al. elasticity is the ability to grow and shrink processing capacity on demand, with varying load. A more precise definition has been given by the Nation Institute of Standards and Technology [MG]:

> *Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out, and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.*

In traditional databases, scaling out an application is a planned process. Spontaneous increase of load could only be handled if additional, available machines can be started and integrated. Hence, a scale-out requires to acquire new machines which causes costs for purchasing as well as their operations. Because of varying system load, these machines are not required anymore once load peak is over. Then they are held on standby and hence most of the time unused. Cloud computing on the other hand has the abilities to allocate computing resources very fast, in an virtually unlimited amount and at any time. Unlike having physical machines at work, the platform operator is paying for the actual use. In the case of decreasing load, a cloud environment allows to de-provision computing power, freeing not necessary machines and therefore saving costs at the same time.

As Weinman proposed in [Pap11], it is in every business interest to match capacity and demand as good as possible. Too few resources lead to unfulfilled demand which turns into lost revenue. Surplus capacities, on the other hand, cause unnecessary costs which cut down profits. This state of excesively unused resources will be referred to as over-provisioning, hereas the state of unmet demand, due to not available resources, will be called under-provisioning.

Weinman describes demand as a function over time on how many resources are necessary to satisfy it. There is also a function for available resources over time. As explained by Islam et al. in [ILFL12], a perfect elastic system would have a resource function which matches the demand function.

Therefore, we are defining elasticity as a quality of a database to allocate immediately just as many resources as required to satisfy the demand and to operate within specified boundaries in a most cost-efficient fashion.

As we can see there are two major characteristics of elasticity - time and cost. A system is more elastic if it is able to react faster on changing conditions. E.g. the faster a system can de-provision excess resources, the more costs can be saved, not having to pay for excess resources any longer.

In contrast to traditional on-site data centers or servers, cloud computing as provided by e.g. Amazon EC2 service, offers a flexible pay-as-you-go pricing model. That means

that consumers only need to pay for the time and the amount of resources they actually consumed. What has been consumed is mostly measured in uptime, CPU, memory or disk usage or accounted network traffic. The time period of accounting varies from provider to provider. E.g. Amazon charges based on an hourly accounting model whereas Jiffybox[Jif13] uses a per second accounting.

In the following subsections we quantify the elasticity of RAMCloud as a persistency layer for a RDBMS in-memory column store such as AnalyticsDB.

3.1 Measuring Elasticity

To determine elasticity, Islam et al.[ILFL12] proposed a way for consumers to measure elasticity by defining financial penalties for over- and under-provisioning. The process of measuring is summarized as following. In an experiment, a specific workload is executed on the benchmarked system. A framework supports measuring the time when the system is in an undesired state - either under- or over-provisioned. As unmet demand has a much bigger business impact than a satisfied demand with excess computing resources, under-provisioning is penalized in a much higher way than over-provisioning. The penalties recorded during the benchmark are aggregated. A normalization provides comparable results.

We took the financial reflection model of Islam et al., adapted it to fit our needs and applied it to quantify the elasticity of AnalyticsDB on RAMCloud.

3.1.1 Calculating Elasticity

To build an applicable calculation model, a few assumptions need to be stated. As already said, costs are an important factor for elasticity. We define the costs for one utilized node to run for one hour to 100 cents. The price is derived from the Amazon EC2 pricing list [Ama13] for a *Amazon M3 Double Extra Large Instance*. The machine has almost the same hardware specifications as the servers in the conducted experiments and can therefore be taken as a reasonable price. At Amazons EC2, a resource is allocated for a minimum time period of one hour. To avoid complexity in situations where a resource is allocated and therefore charged but not available, because it is already de-provisioned or not yet booted, we reduce the chargeable time period to 1 second. Consequently, we discard the calculation of chargeable supply as used by Islam et al.[ILFL12]. The moment the machine is requested, it is charged for until the second it is de-provisioned.

To calculate the elasticity for our experimental system, it is required to track how many servers are actually charged. The moment a server is utilized and therefore chargeable is handled differently by various providers. A resource can be charged from the moment of provisioning of a node to the moment a service on that node is serving. Because booting time of a prepared image with all required services pre-installed can be done in a constant time, we assume for easier consideration that the chargeable time is from the moment the framework detects an under-provisioned state and stops at the moment the framework

de-provisions the node.

The already mentioned penalties for over- and under-provisioning need to be specified as well. We assume an under-provisioning penalty of 10 cents for every second in under-provisioned state. This reflects six times the costs of an additional node. As Weinman et al. emphasize, the benefit of using resources should clearly outweigh the costs of it. We do not penalize over-provisioning state because too many provisioned resources create avoidable costs which we thereby treat as a penalty. So only the pure costs of the provisioned resources are taken into account.

To summarize, it is essential to keep track of the following KPIs:

- time while under-provisioned
- number and runtime of utilized RAMCloud nodes
- maximum allowed runtime of benchmark run

First we need to determine the time the system is in an under-provisioned state. This can be done by subtracting the maximum allowed runtime for a SSB run, the upper boundary, from the actual SSB runtime as demonstrated in Formula 1.

$$f(t) = \mathrm{SSB\_runtime}_t - \mathrm{upper\_threshold}_t \tag{1}$$

To get the time when the SSB runtime was above the upper limit, the function $f_{\mathrm{cutoff}}(t)$ needs to be ascertain. Therefore, only values above the limit as represented in Formula 2 are taken into account.

$$f_{\mathrm{cutoff}}(t) = \begin{cases} f(t) & \text{if } f(t) > 0 \\ 0 \end{cases} \tag{2}$$

Now, that only the amount of time, when the runtime of the SSB has taken longer than the upper bound per node has been determined, the accumulation over all recorded instances can be done as seen in Formula 3

$$P(t_0, t_{\mathrm{end}}) = \sum_{0}^{\#instances} \int_{t_0}^{t_{end}} f_{\mathrm{cutoff}}(t)\mathrm{d}t \tag{3}$$

The next step is to ascertain that the penalty amount for all instances will be multiplied with the defined penalty amount for the runtime of the experiment. This results in an overall penalty as seen in Formula 4.

$$P = P(t_0, t_{\mathrm{end}}) \times p_{\mathrm{underprovisioned}} \tag{4}$$

Finally, the penalty and the runtime costs will be aggregated. To calculate the runtime costs, the number of running nodes for a certain time frame needs to be multiplied by the costs of it. Formula 5 shows the calculation of it.

$$C_{\text{nodes}} = (\sum_{n=1}^{\#_{nodes}} runtime(n)) \times c_{node} \tag{5}$$

Formula 6 shows the calculation of the final elasticity by adding up the penalty and the runtime costs for the provisioned resources C_{nodes} and dividing it by the runtime of the experiment. This gives us a comparable value in cents per seconds.

$$E = \frac{P + \#\text{nodes}_{\text{charged}} \times c_{\text{nodes}}}{t_{end} - t_0} \tag{6}$$

3.1.2 Workload

For our experiment we decided to use the Star Schema Benchmark (SSB) [O'N]. The benchmark has been designed to measure performance of analytical databases and applications. It is based on TPC-H which is widely accepted in literature. The underlying data model is formed as a star schema. It consists of a large fact table and four dimensional tables. The fact table holds a join of the TPC-H LINEITEM and ORDER table. The dimensional tables store information like customer, supplier, part and date.

The SSB benchmark comes with a data generator to create easy to use data sets. The data set size can be configured in scaling factors. The SSB benchmark offers a set of 13 analytical queries classified in four flights with different properties. We implemented the queries in AnalyticsDB. The query suite executes the queries consecutively in an endless loop repeating the SSB after every run. This is used to generate a constant load and to touch all tables within a query suite run. The runtime of every SSB run is stored in a RAMCloud table to provide performance data for the framework.

In Section 4 we will present the results of a sinus-shaped, a plateau-shaped and a exponential workload. The load to the RAMCloud cluster will be generated by multiple AnalyticsDB instances executing constantly recurring the SSB query suite. The required number of AnalyticsDB nodes is managed by the framework controller.

4 High Availability and Elasticity Experiments

This section shows the conducted high availability and elasticity experiments. For each experiment we describe in detail the setup, the sizing of the data set, and the derived insights.

4.1 High Availability of AnalyticsDB on RAMCloud

AnalyticsDB can make use of the high availability features of RAMCloud. As presented by [ORS+11] RAMCloud allows fast-crash recovery. To prove its applicability in the

context of a database application, we conducted the following experiments.

4.1.1 Setup & Sizing

This experiment utilizes 20 nodes. Each node is connected to a switched Infiniband network and equipped with a 2.4GHz CPU with 4 cores and 24GB RAM. One node is running a RAMCloud coordinator. There are 10 cluster nodes running RAMCloud master services with enabled backup service. The remaining 9 servers run an AnalyticsDB which consecutively executes the SSB benchmark suite to create a base load on the cluster. The used SSB data set sizing factor is 10.

RAMCloud allows to configure the number of replicas. That means, that data segments which are stored in RAMCloud will be copied to the configured number of backup nodes. In this setup, three replicas are used.

4.1.2 Results

In this experiment, AnalyticsDB nodes executes permanently the SSB query suite. With sizing factor 10, the average runtime is about 8.2 seconds. As shown in Figure 2, after about 60 seconds, one node is getting manually removed from the cluster, decreasing the total RAMCloud node count to 9 running nodes within the cluster. When a RAMCloud node is being removed, the data it was holding needs to be restored from its backups. The data will be restored on the remaining servers.

Just the data recovery process including relay of the log from the backup takes usually about 100ms. Until the data is restored, AnalyticsDB can not perform any operations on the data while being restored. Noteworthy, the graph in Figure 2 shows a much higher peak than a few milliseconds. This is due to a RPC timeout which has been set to 1000 milliseconds for stability reasons. Nevertheless, the graph shows, that the SSB runtime goes almost back to a normal runtime after the node has been recovered.

AnalyticsDB does not know, that one of its persistency nodes got vanished. It just sees that the execution of a query is not getting completed. Best practice is to restart the operation which timed-out and continue with query execution. Over the whole benchmark, the runtime of Star Schema Benchmark execution goes slightly up because the size of cluster gets smaller.

4.2 Migration time

RAMCloud persists the data in tablets. A tablet is a key range of hash keys which can be assigned to a server. During Scaling-out of a RAMCloud cluster, tablets are being moved from an existing nodes to a recently added new one. The time the system needs for data migration depends on two major factors. The first one is the load on the system. Since migrating data from one server to another adds additional load to the cluster, it is

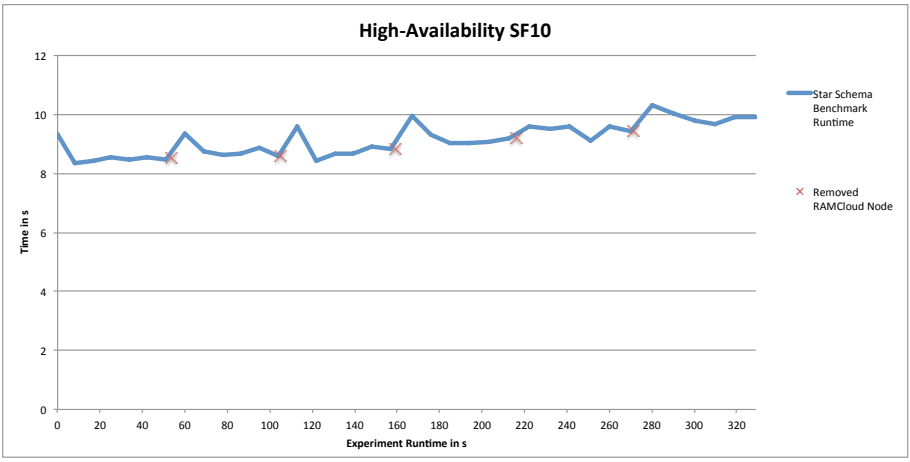

Figure 2: High-Availability experiment

obvious that the higher the load from operational work on the cluster is, the longer it takes to migrate tablets to a new server.

The second factor is the size of the cluster. Scaling out a RAMCloud installation with one node to a two node cluster means that 50 percent of all data needs to be moved. The diagram in Figure 3 shows how both factors influence the total time of migration.

The experiment has been conducted using the SF10 data set and three different load scenarios. The first load scenario is a low load experiment where one AnalyticsDB node is running the SSB in an endless loop on one RAMCloud node. The x-axis represents the total number of RAMCloud master services in the cluster and the y-axis represents the time it took to distribute the data among the nodes on scaling-out. We conducted the same experiment with five and ten AnalyticsDB nodes running the SSB infinitely creating different levels of load to the cluster.

As one can see the factor of time for migrating is about the same as the load factor. E.g it takes a little less than 5 times the time to migrate from one node to two nodes as well as about 8 times longer with a ten times higher base load. It is also very visible that with a higher number of total nodes, the migration times are approaching the same level no matter how high the actual load is. This is due to the fact that there is much less data to be moved between the servers, the more servers are already in the cluster.

4.3 Elasticity

This subsection presents the elasticity experiments which have been performed. The three different workloads presented in the following sections show how elastic RAMCloud reacts on different load patterns executed by AnalyticsDB.

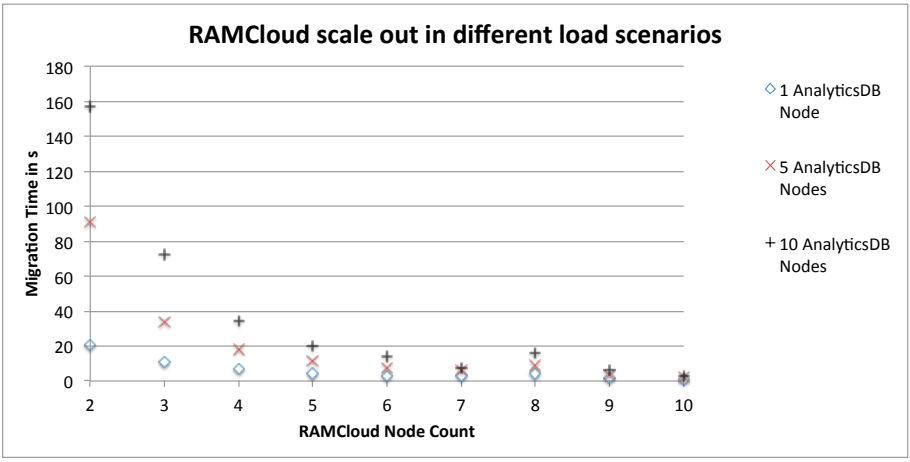

Figure 3: Migration time in different load scenarios

4.3.1 Setup and Sizing

For the elasticity experiment, we use an Infiniband cluster with 22 servers. Each server has a 2.4GHz CPUs housing 4 cores and 24GB RAM. 10 servers are dedicated AnalyticsDB servers and 10 are used to run RAMCloud master servers. One node is used as the RAMCloud Coordinator and one node as management unit running our presented benchmark framework to control load, scale-in, scale-out, and redistribution of data.

The AnalyticsDB nodes can be seen as load generators to the cluster. Each AnalyticsDB node runs the Star-Schema-Benchmark (SSB) as presented in 3.1.1 in an endless-loop. The SSB benchmark is executed on a data set of about 6GB in size. On experiment start, the coordinator and one RAMCloud node are started. Once the RAMCloud cluster is up and running, an initial AnalyticsDB node is launched to seed the data. When everything is ready the framework initializes the experiment for the selected workload.

On every finishing of a SSB benchmark run, AnalyticsDB saves the total used runtime in RAMCloud. The load manager checks every 300ms the runtime of the last finished benchmark run. The upper bound is set to 30s and the lower bound to 20s.

4.3.2 Sinus-shaped Workload

The first experiment that we present executes a sinus-shaped workload. In the beginning only one AnalyticsDB node executes the SSB benchmark, after a while a second node, then three more nodes and finally five nodes are added. The experiments highest load counts ten AnalyticsDB nodes in total. After a while, five nodes, then three nodes and finally one node are removed to lower the load back to the initial load.

Once the load manager detects a breach of the upper bound, it will check another 90 times before acting. The delay of the lower bound is set to 30. The delay for the upper bound

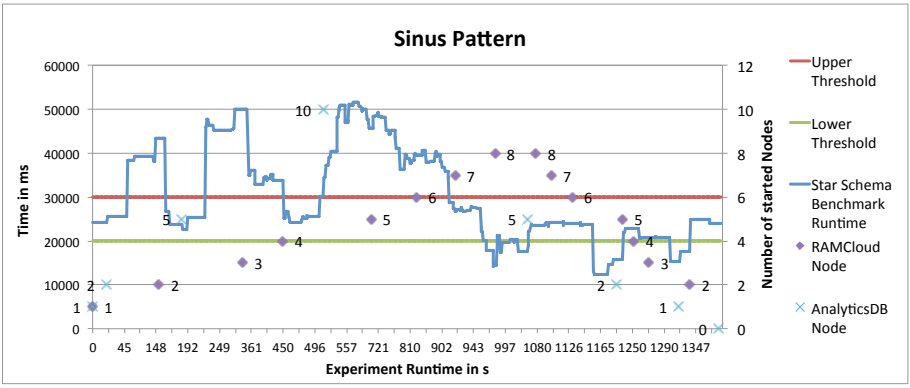

Figure 4: Sinus-shaped Workload

breach is 3 times higher then for the lower bound. This can be useful to avoid provisioning resources in case of runaways. On the other hand, a shorter delay helps the load manager to provision new resources faster and therefore react more elastically. For future work it can be interesting to investigate what the best delay setting constellation for the load manager is.

Figure 4 shows a sinus-shaped workload pattern in which RAMCloud elastically scales out and later back in as load is decreasing.

The delay between breaching the upper bound and starting the second RAMCloud node is obvious. Since the load manager is a simple reactive manager, it only starts one RAMCloud node at the time leading to a period of slow mitigation from the moment of high load at about 550 seconds from the experiment's beginning. At about 950 seconds, the SSB runtime is back in its normal boundaries leading the load manager to not start more than the eight already running RAMCloud nodes. With lowering the load, the system reacts faster with de-provisioning of nodes. This is because of the lower delay on breach of the lower bound.

4.3.3 Plateau Workload

The second experiment is depicted in Figure 5. The upper and lower boundary for the SSB runtime is set to the same value (30 seconds upper and 20 seconds lower) as in the first experiment. One RAMCloud node is started at the beginning of the experiment. The cluster load gets ramped up by starting an AnalyticsDB node every 60 seconds. In this experiment, the load manager's delay on acting to a boundary violation is configured to 60 checks every 300 milliseconds before acting. Because of the instantaneous under-provisioning situation, the load manager starts seven more RAMCloud nodes to bring the SSB execution time into the specified boundaries.

The time period the cluster runs on high load is called plateau time. In this experiment,

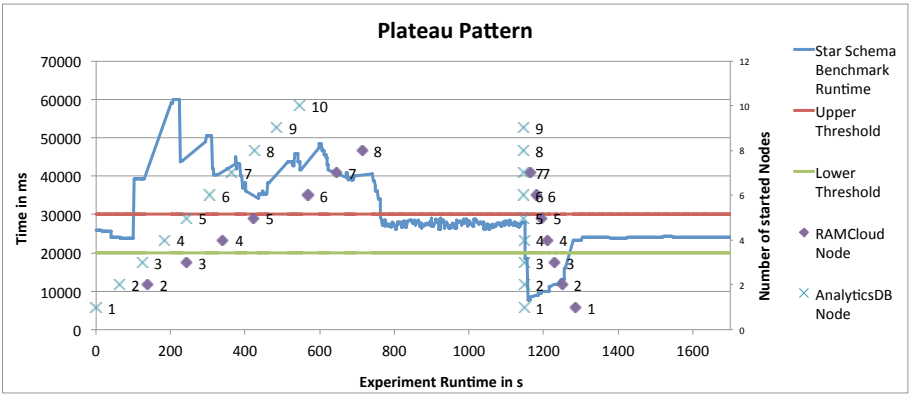

Figure 5: Plateau Workload

it is 600 seconds long. At about 1100 seconds, the plateau time is over. The benchmark framework stops 9 out of 10 AnalyticsDB nodes, reducing the cluster load to one tenth of the plateau load. This leads to a immense over-provisioning situation. The runtime of the SSB benchmark drops to under 10 seconds per SSB run. The load manager acts immediately by stopping 7 of the 8 running nodes to bring the runtime back into its boundaries. The SSB runtime runs within boundaries until the end of the experiment.

4.3.4 Exponential Workload

In the third experiment, we probed a exponential workload. As shown in Figure 6 the experiment starts with one AnalyticsDB node. After the second node has been added, first 2 then 4 nodes are added with a delay of 120 seconds. The load manager delay on acting to boundary violations is again 60 checks long by checks every 300ms. It is clearly visible, that the system needs much more time to normalize the SSB runtime when adding 4 nodes to the system than with less. This has two reasons. The first one is that the load manager only adds one node at a time. With a more efficient load manager, it could be possible to start more RAMCloud nodes the more the load has increased. This would lead to a faster recovery of the SSB runtime. The second reason lies in the cluster load itself. The higher the load in the system the longer it takes to migrate data between the nodes to distribute the load among more nodes.

When the load decreases, the system corrects the over-provisioning situation by stopping RAMCloud nodes.

4.3.5 Evaluation

The three presented and measured workload scenarios to experiment with elasticity properties can be seen in Table 4.3.5. We can see that the costs for nodes as well as the penalties for the three workloads vary from each other. For the given system, framework and load

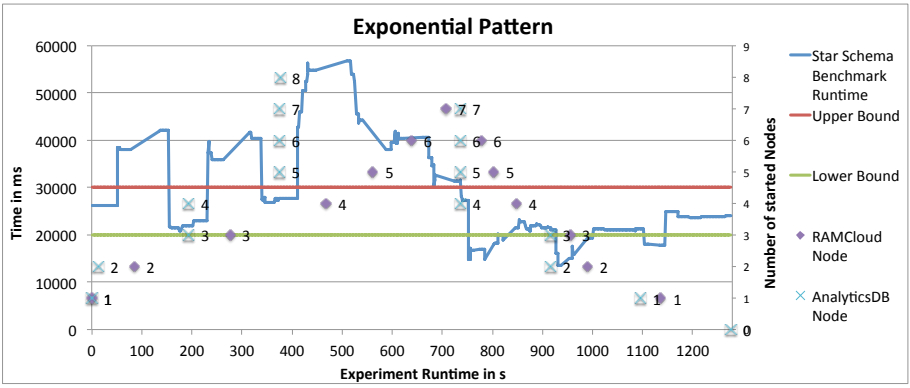

Figure 6: Exponential Workload

Workload	C_{nodes} in cents	$P_{\text{underprovisioned}}$ in cents	E in cents/second
sinus	9,099.83	31,619.61	29.59
plateau	11,918.79	28,447.91	23.74
exponential	6,768.51	22,612.04	22.97

Table 1: Elasticity of Experiment Workloads

manager this gives us a quantification of elasticity of AnalyticsDB on RAMCloud.

5 Related Work

Minhas et al. present in [MLA+12] how a partionied, shared nothing parallel database system can be exploited to scale-out elastically. The authors demonstrate a scale-out solution using the in-memory database system VoltDB. Database systems nowadays are not able to automatically scale-in and scale-out to react on load changes to it. The authors present a mechanism for scale-out by dynamically adding more resources and moving data from existing nodes to recently added nodes.

In the work of Konstantinou et al. in [KAB+11], a monitoring system for NoSQL systems is being presented. With the help of this monitoring system, the elasticity features of popular NoSQL databases on a cloud platform are quantified. Furthermore, the authors present a decision making system for automatic elastic operations for any NoSQL engine.

In [TWR+], Mühlbauer et al. describe a hybrid OLTP and OLAP DBMS called ScyPer. It deals with high availability by multicasting a redo log of the processed OLTP data from the primary database server to multiple secondary nodes. Clients send transactional and analytical queries to the primary server, where the transactional data is being processed. Analytical queries, however, are load-balanced to secondary nodes by a coordinator process. On node failure of the primary node, the secondary nodes elect by common consent

a new primary node using a PAXOS-based protocol based on the last acknowledged log sequence number. Secondary nodes can fail at any time because the authors assume that the client will resend the analytical query after a timeout. OLAP queries can also be replicated throughout the system, allowing a continuation of OLAP processing after a node failed.

Some fundamental aspects about keeping data in in-memory databases resistant against failure is presented in [GMS92]. The authors state that if data is kept in volatile memory it is indispensable to have a backup on a stale medium. One possibility to achieve this is by maintaining a transaction log. This log can be used to replay transactions on a system failure with data loss. In case of a failover, replaying a whole database back to memory can take some time. The authors point out that checkpoints can help to reduce the time for recovery quite a lot.

6 Conclusions

In this paper, we present a shared storage architecture where an in-memory database system (AnalyticsDB) uses a DRAM-based storage system (RAMCloud). Throughout the various experiments in the paper we demonstrated the following aspects: a) this architecture allows an in-memory DBMS to utilize RAMCloud's fast-crash recovery mechanism where the data from a crashed storage node could be restored within a few seconds. b) During the recovery phase, the query processing was able to continue. c) The performance penalty on the query processing was between 25% and 40% as some resources from the storage system were used for the recovery. d) This architecture allows for an adaption to changing database workloads within seconds by e.g. adding more resources to the storage system. e) Again, this adaptation did not result in an interruption of the query processing.

References

[Ama13] Amazon. Amazon EC2 Prizing List, 2013.

[BKM08] Peter A. Boncz, Martin L. Kersten, and Stefan Manegold. Breaking the memory wall in MonetDB. *Commun. ACM*, 51(12):77–85, 2008.

[CJMB11] Carlo Curino, Evan P.C. Jones, Samuel Madden, and Hari Balakrishnan. Workload-aware database monitoring and consolidation. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 313–324, New York, NY, USA, 2011. ACM.

[FML+12] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[GMS92] H. Garcia-Molina and K. Salem. Main memory database systems: an overview. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):509–516, 1992.

[ILFL12] Sadeka Islam, Kevin Lee, Alan Fekete, and Anna Liu. How a consumer can measure elasticity for cloud platforms. *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering - ICPE '12*, page 85, 2012.

[Jif13] Jiffybox. Jiffybox, 2013.

[KAB+11] Ioannis Konstantinou, Evangelos Angelou, Christina Boumpouka, Dimitrios Tsoumakos, and Nectarios Koziris. On the elasticity of NoSQL databases over cloud management platforms. *Proceedings of the 20th ACM international conference on Information and knowledge management - CIKM '11*, page 2385, 2011.

[MG] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing (Draft) Recommendations of the National Institute of Standards and Technology. 145.

[MLA+12] Umar Farooq Minhas, Rui Liu, Ashraf Aboulnaga, Kenneth Salem, Jonathan Ng, and Sean Robertson. Elastic Scale-Out for Partition-Based Database Systems. *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 281–288, April 2012.

[OAE+11] John K. Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Diego Ongaro, Guru M. Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman. The case for RAMCloud. *Commun. ACM*, 54(7):121–130, 2011.

[O'N] O'Neil, P. E. and O'Neil, E. J. and Chen, X. The Star Schema Benchmark (SSB).

[OOCR09] Patrick E. O'Neil, Elizabeth J. O'Neil, Xuedong Chen, and Stephen Revilak. The Star Schema Benchmark and Augmented Fact Table Indexing. In Raghunath Othayoth Nambiar and Meikel Poess, editors, *TPCTC*, volume 5895 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2009.

[ORS+11] Diego Ongaro, Stephen M. Rumble, Ryan Stutsman, John K. Ousterhout, and Mendel Rosenblum. Fast crash recovery in RAMCloud. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 29–41. ACM, 2011.

[Pap11] Working Paper. Time is Money : The Value of " On-Demand " Time is Money : The Value of On-Demand . pages 1–29, 2011.

[ROS+11] Stephen M. Rumble, Diego Ongaro, Ryan Stutsman, Mendel Rosenblum, and John K. Ousterhout. It's time for low latency. In *Proceedings of the 13th USENIX conference on Hot topics in operating systems*, HotOS'13, pages 11–11, Berkeley, CA, USA, 2011. USENIX Association.

[TKG+13] Christian Tinnefeld, Donald Kossmann, Martin Grund, Joos-Hendrik Boese, Frank Renkes, Vishal Sikka, and Hasso Plattner. Elastic online analytical processing on RAMCloud. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 454–464, New York, NY, USA, 2013. ACM.

[TWR+] M Tobias, R Wolf, Angelika Reiser, Alfons Kemper, and Thomas Neumann. ScyPer : A Hybrid OLTP & OLAP Distributed Main Memory Database System for Scalable Real-Time Analytics. pages 499–502.