Elasticity Controller for Cloud-Based Key-Value Stores

Ala Arman, Ahmad Al-Shishtawy, and Vladimir Vlassov

KTH Royal Institute of Technology, Stockholm, Sweden
{aarman, ahmadas, vladv}@kth.se

***Abstract*—Clouds provide an illusion of an infinite amount of resources and enable elastic services and applications that are capable to scale up and down (grow and shrink by requesting and releasing resources) in response to changes in its environment, workload, and Quality of Service (QoS) requirements. Elasticity allows to achieve required QoS at a minimal cost in a Cloud environment with its pay-as-you-go pricing model.**

In this paper, we present our experience in designing a feedback elastically controller for a key-value store. The goal of our research is to investigate the feasibility of the control theoretical approach to the automation of elasticity of Cloud-based key-value stores. We describe design steps necessary to build a feedback controller for a real system, namely Voldemort, which we use as a case study in this work. The design steps include defining touchpoints (sensors and actuators), system identification, and controller design. We have designed, developed, and implemented a prototype of the feedback elasticity controller for Voldemort. Our initial evaluation results show the feasibility of using feedback control to automate elasticity of distributed key-value stores.

***Keywords*-Cloud Computing; Elasticity; Feedback Control; Key-Value Store; Voldemort.**

I. Introduction

Cloud computing is leveraged by various IT companies and organizations. A Cloud is an infrastructure that provides resources, information and services as a utility over the Internet [1]. In recent years, Cloud storage, such as Amazon S3, has become rather popular and widely used in many different application domains, including Web 2.0 and mobile applications.

There are several features defined for Cloud computing such as scalability and elasticity that can be leveraged when developing Cloud-based applications. There are two main advantages of Cloud computing over other large scale computing alternatives. The first one is that the end-user does not need to be involved in the configuration and maintenance of the Cloud. A developer does not have to buy servers, security solutions, etc, and set them up; rather, she builds her applications on Cloud resources [2] of a Cloud provider. The second advantage and probably the most important one, is that the end-user only pays for resources she requests and uses. That is why the Cloud computing approach is less expensive than its alternatives. However, this property, called "pay-as-you-go" has an important drawback. If allocated resources exceed the required amount, it will incur a waste of money. On the other hand, if the obtained resources are not adequate, the system might not meet the Service Level Objectives (SLOs), resulting in a negative impact on its performance and, as a consequence, negative impact on user experience with the system.

Considering these issues, the concept of elasticity comes to the attention of many Cloud users. There is a difference between elasticity and scalability. In the context of this work, scalability means the expansion capacity of the system, which is expected to be reached in the future. This approach has a very important drawback, which is that the ultimate size of the system should be specified in advance. There is a contradiction between this concept and the "pay-as-you-go" property of Cloud computing, because in this approach the end-user should pay for the ultimate size of the system which might never be fully utilized. Another disadvantage is, as scaling up means adding more physical resources; it might be not easy to scale down by removing them.

To deal with these problems, a new approach, called Elasticity, has become favorite in recent years. In this approach, the final size of the system is not predefined. But an elastic system is capable of scaling up and down (growing and shrinking by requesting and releasing resources) at runtime in response to changes in its environment, workload, and Quality of Service (QoS) requirements. In the case of increasing the load, a new instance is added to meet SLOs; whereas, if the load decreases, a number of instances are removed from the system in order to reduce the cost. In order to effectively and efficiently utilize the elasticity property of the system, the elasticity control should be automated.

In this paper, we present our experience in designing an elasticity controller for a key-value store. The goal of our research is to investigate the feasibility of a control theoretical approach to automation of elasticity of a Cloud-based storage by performing all design steps necessary to build a feedback controller for a real system, namely Voldemort [3], [4], used as a case study. The design steps include defining touchpoints (sensors and actuators), system identification, and controller design. We have designed and developed a prototype of the feedback elasticity controller for Voldemort.

There has been several related work in the area of the automation of elasticity Cloud-based storage services such as [5]–[7]. For example, in [5], the SLO is specified as a requirement on the average response time. The system variable monitored and used in feedback control is the CPU utilization, because, as shown by the authors, the CPU utilization (which

is relatively easy to monitor) is highly correlated with the response time. In the case of high CPU utilization, the number of active nodes is increased by adding new nodes to the system. Similarly, the number of active nodes is decreased in the case of low CPU utilization. However, in a Cloud environment the aforementioned correlation might not hold due to the variable performance of Cloud VMs [8], [9].

In [10], one of the system variables is the response time, i.e., the time it takes to send a request from a client to an application, to process the request, and to return the result to the client. The round trip time depends on several metrics such as physical medium and distance between the source and the destination, the existence of interference in the system etc. These metrics do not reflect the amount of the load in the system. Thus, the round-trip time is not a good option as a system variable to be monitored and used for control. In [10], only scalability has been considered; however, in an elastic system, resources can be removed in the case of low workload.

In this paper, we consider service time as a system variable monitored and used in feedback control. It is the time needed for an operation to be served in the system (a distributed storage, namely Voldemort [3], in our case). In other words, it does not include the network round-trip time. This time reflects changes in the workload fashion.

We design, implement, and evaluate an automatic controller, which is built based on control theory considering the elasticity property in a distributed storage. We use the Voldemort distributed key-value store as a case study. In other words, the controller would be an extension to Voldemort in a way that it scales a Voldemort cluster up by allocating more nodes in the case of a high load and scales it down by removing a number of nodes in the case of a decreasing load. The goal here is that a system uses the resources in an efficient way, so that it does not waste resources in the case of a low load. On the other hand, it adds a number of nodes to meet the SLO in case of increasing workload. In this work, we define the SLO as the 99th percentile of read operation latency over 1 minute period. Moreover, the automatic controller eliminates the need for the administrator of the system to configure the system manually to leverage the main advantage of Cloud computing (as a utility), "pay-as-you-go".

The rest of paper is organized as follows. In Section II we present the architecture of our elasticity controller integrated with the Voldemort key-value store. Section III describes the system identification process followed by the controller design described in Section IV. Evaluation of our elasticity controller is discussed in Section V. Finally, we present conclusions and future work in Section VI.

II. Elasticity Controller Framework

We have designed and implemented a controlling framework to automate elasticity in distributed key-value stores. Elasticity control can be manual in a way that adding or releasing resources would be done by the administrator of the system. However, our framework has been designed to monitor the load in the system and allocate or release resources based on a feedback controller as described in this section. When the load increases, the nodes probably cannot handle the requests in appropriate time (the SLO). Therefore, the controller would detect this and handle this issue by adding a number of nodes according to its parameters. In other word, the role of the controller is deciding about the time of adding the nodes to the system and the number of nodes that are going to be added. Similarly, when the load decreases and the service time becomes less than the SLO, the nodes are less busy and the storage can handle the requests with less number of nodes. Therefore by removing some nodes, we can save more resources and reduce the cost of using resources as a result. In our work, we define the SLO as the 99th percentile of read operation latency over 1 minute period.

A. System Architecture

We mentioned that in a Cloud environment which is dynamic, the management of resources becomes very important. They should be managed in such way that they would keep their efficiency. We design and implement a controller that monitors the performance of the storage system and requests to allocate or release the nodes based on the deviation from the desired performance caused by changes in the workload. Fig. 1 shows a generic control framework.

We can see that system has a reference input (Set point) which is the desired value of the service parameter which is set by administrators. In our case, it is the desired 99th percentile of read latency that is compared with measured output from the Sensor. This is done by the calculating the error between the measured and the desired value. The result is error signal which is used by the Controller to make decisions. Control decisions are passed to the actuator, which makes change in the controlled system that typically result in a change in the measured output bringing it closer to the Set point.

Now we consider our framework in more details. Fig. 2 shows the architecture of the framework which consists of the following six major components.

- **Voldemort:** A distributed key-value store that consists of a cluster of nodes.
- **YCSB:** A benchmark tool which is an open source framework that allows creating various load scenarios [11].
- **Sensor:** It is a component that monitors the load by measuring the 99th percentile of read operation latency over a fixed period of time (1 minute in our case) and then gives it to the Filter.
- **Actuator (rebalance tool):** It gets a target cluster file from the elasticity controller and updates the cluster.
- **Filter:** It smooths the service time signal that is given to the controller by avoiding spikes in output values resulting from noise.
- **Elasticity controller:** It is a PID (Proportional Integral Derivative) controller that gets the average service time in each interval from the Filter and decides on the number of nodes that should be added or removed based on gain parameters that has been specified before.

Table I summarizes components of the controlling framework.

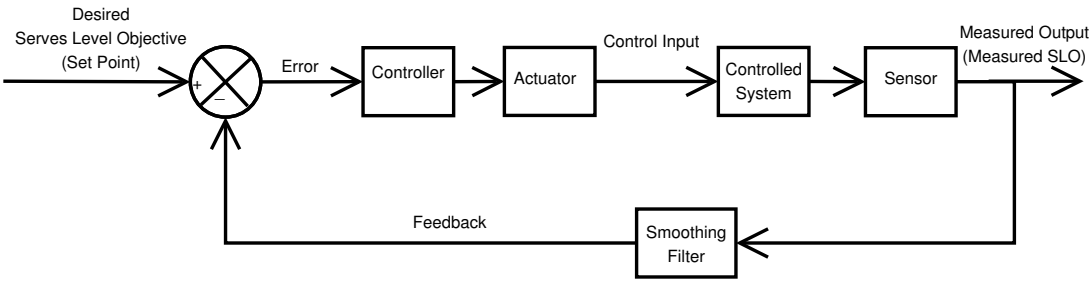

Fig. 1. Generic Control Framework

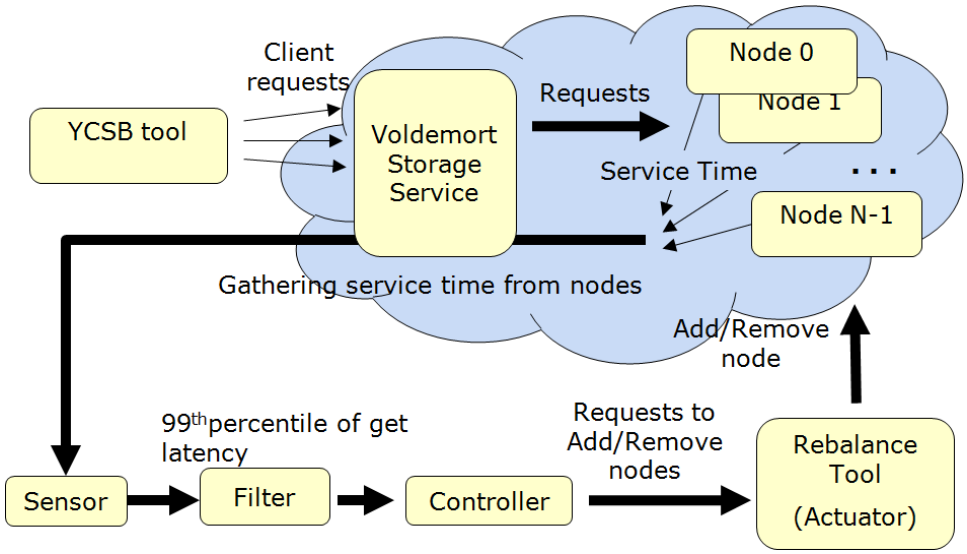

Fig. 2. Framework Architecture

Component	Tool used / Implemented in
YCSB	Embedded benchmark tool in Voldemort
Actuator	Embedded rebalance tool in Voldemort
Voldemort Storage	Java
PID Controller	Matlab / Java
Sensor	Java
Filter	Java

TABLE I
COMPONENTS IN THE CONTROLLING FRAMEWORK

In the following we present in more details the components of the framework.

B. Voldemort

Voldemort is an open-source, distributed, fault-tolerant non-relational key-value hash table. It is used, in particular, in LinkedIn for highly-scalable storage services. Voldemort supports automatic data replication and data partitioning among multiple servers, as well as data versioning to improve data integrity in the case of failures without compromising availability. Voldemort is decentralized as the Voldemort nodes are independent from each other, and hence there is no single point of failure. Voldemort includes a rebalance tool that allows adding and removing nodes. For more details on the Voldemort key-value store, the reader is referred to [3], [4].

C. Yahoo! Cloud Serving Benchmark (YCSB)

In order to generate client requests, we have used an open source benchmark tool called Yahoo! Cloud Serving Benchmark (YCSB) [11]. The most important characteristic of this tool is its extensibility which means that it can be used to benchmark Cloud storage systems and also to generate new types of workload.

D. Touchpoints

According to [12], a touchpoint is an interface to the managed resource that implements the sensor and actuator behaviors for the resource. In our work, we define a touchpoint as an interface to Voldemort that include sensors and actuators, which allow measuring the service time in each Voldemort server, aggregate these measurements, and actuate by adding or removing Voldemort nodes.

Sensor: A sensor is a software component that is able to monitor a Voldemort server (e.g., measure the read operation latency). We use server sensors to monitor the server load in the Voldemort cluster by measuring service time (which is the

99th percentile of get operation latency). The system sensor in Fig. 2 use the server sensors to collect and to aggregate measurements and give the stream of aggregated values to the Filter.

Actuator: In our system, an actuator is an API provided by the Voldemort rebalance tool that allows adding and removing Voldemort nodes and activate data rebalancing among nodes. An actuator is used to make some changes in the storage in order to move system performance to a desired region. From our point of view, a desired system is the one that uses the resources based on the load changes. This is possible by adding or removing nodes in Voldemort. If the load increases, it will add some more resources to handle the increasing load and if the load decreases, it will remove some nodes not to waste the resources and save more money. Adding and removing node is done during rebalancing process. Therefore we used rebalancing tool as an actuator.

E. Filter

Sometimes values measured by the system sensor are not smooth enough because of noise. In order to reduce the influence of noise, we use a smoothing filter that can decrease the fluctuations in the measured output values. We have designed a filter component in a way that we give more weight to the previous filter output and less weight to the new filter input that is described mathematically as follows.

$$\text{Filter Output} = 0.9(\text{Previous Filter Output}) + 0.1(\text{New Filter Input})$$

F. PID Controller

One of the most important components in our controller framework is the PID controller. It gets the filtered values form Filter component and decides how many nodes should be added or removed based on gains that have been calculated during controller design.

The classical steps in system identification (building a system model) and controller design are as follows. First, the designer performs a number of experiments in order to identify the system. In these experiments, she collects the data measured by sensors. Second, using the data collected, she obtains an identified model using PEM (Prediction Error Minimization) method. Third, based on the obtained model, she creates a feedback model with the transfer function of the system. Finally, she designs the controller to be inserted in the system. Controller design phase includes selecting the controller type and determining its gains.

In the following sections, first we describe the system identification process and then present the steps of the elasticity controller design.

III. System Identification

System identification is the process of recognizing relation between the control input and monitored output of the system and how output depends on the input. It can be considered as a link between application in real world and model abstractions. Identification is about building a formal model of the system. Starting from a set of measured input and corresponding output values, by using one of system identification approaches, it is possible to create a mathematical model, which estimates the real model.

In this work, we used the black-box approach [13], which allows building a model without knowing properties of the system. This is a mathematical approach in which a model is proposed based on measured input and output values by means of identification experiments. We used this approach in our work because the studied system (Voldemort) is a complex system with many parameters. The black-box system identification process typically passes the following steps.

- Determining inputs and outputs of the system to be used in the model;
- Experiment and collect the input and corresponding output values;
- Preprocess data and select the useful part of data;
- Design a model based on the data which have been collected;
- Observe the system behavior. If the model does not reflect the system behavior, go to the first step.

A. System Input/Output

Input and output of the system are shown in Fig. 3. System input is the number of nodes that are going to be added or removed. System output is the 99th[1] percentile of get (read) operation latency. We measured several possible outputs for the system. After running various experiments, we found out that the 99th percentile of read latency is the best parameter to represent the output of the system. Because it replied more reasonable results to the different load scenarios that we applied to Voldemort.

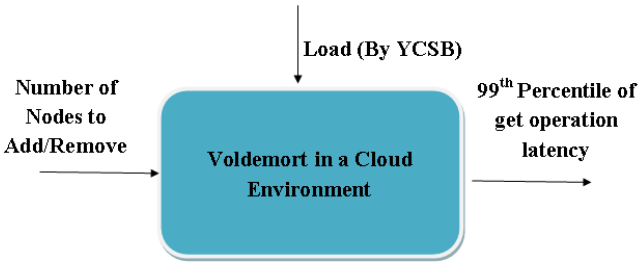

Fig. 3. System Input/output

B. State-Space Model

We used System Identification Tool in Matlab to model the system. We import input and output data that we gained in the data acquisition step in this tool and choose linear parametric models to estimate the model. Command `ident` opens this tool for us. We exploited state-space approach to model the system that models a system based on the input, output and state variables within the system. The most important

[1]The 99th percentile of get latency equals x means that 99% of get operation latencies are below x ms.

advantage of this approach is its extensibility in a way that we can add input and output to the system easily. We chose this approach to model the Voldemort key-value store used as a case study.

The dynamics of basic state-space model is described by the following equations.

$$x(k+1) = Ax(k) + Bu(k) \quad (1)$$
$$y(k) = Cx(k) + Du(k) \quad (2)$$

where $x(k)$ is the vector of state variables, $u(k)$ is output matrix, $y(k)$ the input matrix, D is the delay scalar. We use the PEM method to find the identified model of the system. To use the method, we need to specify two parameters, delay and the order of the system. Delay is a vector of the number of input delays which is zero in our case. The order of the system can be estimated by the following command in Matlab, which is used to estimates the parameters for the state-space model:

```
pem(dat,'best')
```

The above command specifies the best order for the system and `dat` which is an object created by the `idata` command that takes the output vector and the input vector as its parameters. By using the `pem` command, we have found that the best order for the system is 2. Now we have all parameters to model the system. After adding the model object to the workspace, we estimate the initial state of the model by executing the following command:

```
pem(dat,'best','InitialState','estimate')
```

and we have the initial states as a scalar:

$$X_0 = [0.32689 - 0.96019]$$

After we have built the model using the system identification tool, we can determine A, B, C and D (equation 1 and 2) using the `ss` function in Matlab. It creates a state-space object from `PEM` model that we have built.

```
sys=ss(pss)
```

where `pss` is the `PEM` model object that was created by System Identification Tool.

$$A = \begin{bmatrix} 0.88577 & -0.035944 \\ 0.11396 & 1.0356 \end{bmatrix} \quad (3)$$
$$B = \begin{bmatrix} -0.00069035 & 0.00059463 \end{bmatrix} \quad (4)$$
$$C = \begin{bmatrix} 0.142 & 0.0054066 \end{bmatrix} \quad (5)$$
$$D = \begin{bmatrix} -0.000091668 \end{bmatrix} \quad (6)$$

C. Transfer Function

The next step in the system identification process is obtaining a transfer function of the system, which can be calculated by the `tf` command in Matlab. The command takes the state-space model object as input parameter (computed by `ss`) and converts it to the transfer function form:

```
Transfer Function=tf(sys)
```

In our case, the transfer function of the Voldemort system is as follows.

$$\frac{0.0001565z^2 - 0.00009465z + 0.000007484}{z^2 - 1.921z + 0.9215}$$

IV. Controller Design

We have used Simulink environment to design the controller. The graphical designed controller in Simulink is shown in Fig. 4.

To configure the transfer function block, we insert the dominator and numerator coefficients of the transfer function that was calculated to the block as well as initial state scalar. We set the SLO (reference point) as 0.036 seconds. By tuning the controller, we set the gain parameters K_p, K_i, and K_d of the PID controller that are the proportional, integral, and derivative gains of the controller, respectively. After tuning the gain parameters using PID Tuner (in Matlab), finally we reach the block response and Tuned response time (using PID controller):

Now we have the gain parameters of the PID controller:

$$K_p = 1.19785394231464$$
$$K_i = 0.0256625849637579$$
$$K_d = -288.920114195685$$

V. Evaluation and Experimental Results

In this section we present the evaluation of the Elasticity controller for the Voldemort key-value store. We mentioned that the second step in the system identification is data acquisition. In the next section we show how we gathered data to identify the system.

A. Setup

We discuss the experimental setup in two parts, node setup and benchmark setup. In both parts, the parameters selected empirically by running various experiments that led us to the most efficient parameters.

Node Setup: Our cluster consists of 8 Voldemort nodes running on 8 machines. The node setup of our experiment is described in Table II.

Benchmark Setup: For effective benchmarking, we used two powerful machines that are capable of simulating multiple clients requests in order to load the Voldemort nodes as much as benchmark parameters were set. Table III, shows the setup used in our benchmark.

B. Benchmark Experiment for System Identification

We started with three active nodes and run the controller and two YCSB instances with a specific throughput. Then with a delay of 30 minutes between each rebalancing, nodes 4th to 8th are added. Afterwards, with the same delay, nodes 8th to 4th are removed to cover the range of input.

Fig. 5 and Fig. 6 show the results of experimental design and data acquisition. The X-axis shows the sampling time. Fig. 5, Y-axis shows the number of nodes and In Fig. 6 shows

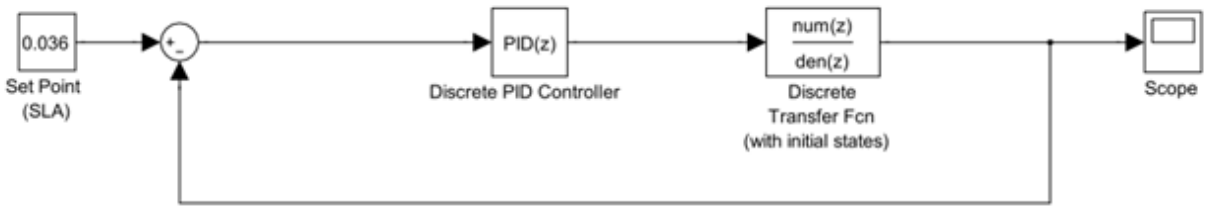

Fig. 4. Graphical design of the PID controller using Simulink

Machine Setup	
Parameter	**Value**
Processor	4
CPU Cores	4
model name	Intel Core2 Quad Q9400 @ 2.66GHz
Cache size (MB)	3
Memory (MB)	3887
Node Setup	
Parameter	**Value**
Voldemort Version	0.90.1
Database Server	Berkely DB
Socket Timeout (ms)	90000
Routing Timeout (ms)	100000
Bdb cache size	1 G
JVM_SIZE (Min and Max)	4096 MB
Replication Factor	3
Required Writes	2
Required Reads	2
Key Serializer's Type	String
Value Serializer's Type	String

TABLE II
NODE SETUP FOR DATA ACQUISITION

Machine Setup	
Parameter	**Value**
Processor	24
CPU Cores	6
Model Name	Intel Xeon X5660 @ 2.80GHz
Cache Size (MB)	12
Memory (MB)	44255
Benchmark Setup	
Parameter	**Value**
Number of records inserted in warm-up	10000
Write Percentage (%)	5
Read (%)	95
Showing Result Interval (Sec)	60
Throughput (Ops/Sec)	4000
Sampling Time (min)	5

TABLE III
BENCHMARK SETUP

the average 99th percentile of read operation latency. As we can see, by increasing the number of nodes, the 99th percentile of read latency decreases. Similarly, by removing some nodes, the 99th percentile of read latency increases.

As was mentioned in the previous chapter, the model that is created by system identification tool is a `PEM` model in State-Space structure. Fig. 7 compares the measured output with the simulated model. The y-axis shows the 99th percentile of read operation latency. As we can see, the measured and simulated

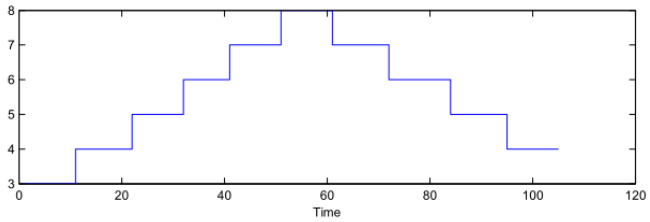

Fig. 5. The changes in the number of nodes in the experimental design and data acquisition

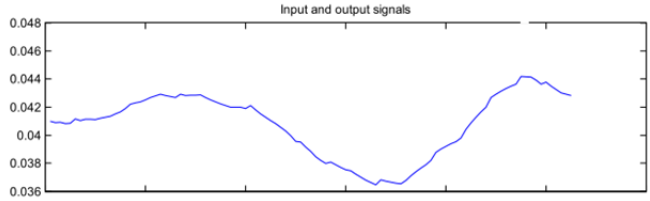

Fig. 6. The changes in the 99th percentile of read operation latency changes in the experimental design and data acquisition

models coincide rather well.

C. First Experiment: Low Workload

Fig. 8 and Fig. 9 show the results of the first experiment. The experiment is as follows. We run the sensor for half an hour and then, at point C, we starte the controller. After about 40 minutes (at point L) we decrease the load. The decrease in the load causes the controller to remove a number of nodes in order not to waste resources as motivated by the "pay-as-you-go" pricing model. Table IV shows the configuration of YCSB instances for this experiment. Other parameters were the same as in the benchmark experiments described above.

Another important point in our experiments is that, as we use a filter in our framework so that the controller sees the filtered values. That is why that the changes in the number of nodes were done with a short delay; but the controller sees smoother outputs with less spikes.

D. Second Experiment: High Workload

Fig. 10 and Fig. 11 show the results of the second experiment. In this experiment, we ran the sensor for half an hour and then the controller started at point C. After about 110 minutes

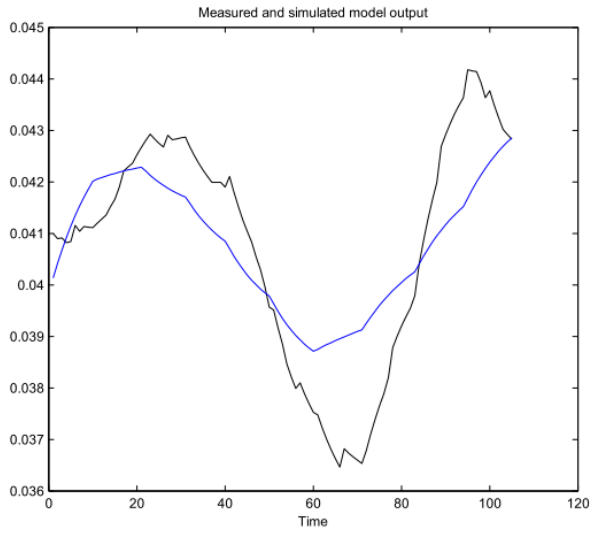

Fig. 7. Model output. The black curve shows the measured output values and the blue one shows the output of simulated model by Matlab

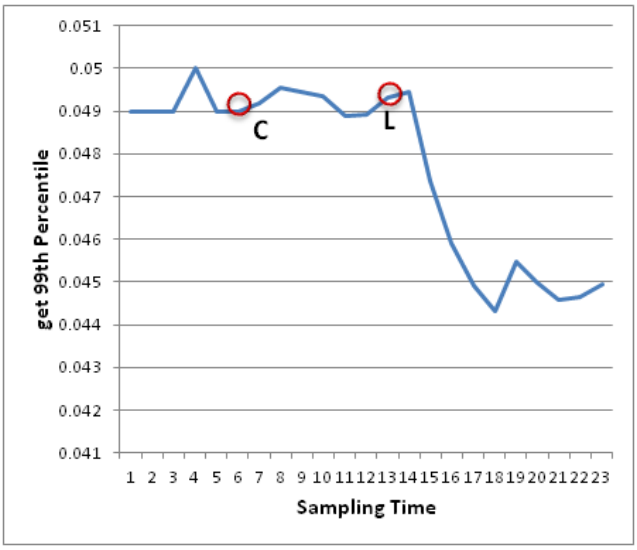

Fig. 8. The changes in the 99th percentile of read operation latency (the first experiment)

Warm-up Period Configuration	
Parameter	**Value**
Warm-up Period (min)	30
Throughput during warm-up (Ops/Sec)	4000
Value Size during Throughput (bytes)	1024
Benchmark (YCSB) Setup for decreasing Load	
Parameter	**Value**
Throughput(Ops/Sec)	500
Value size (bytes)	512

TABLE IV
THE CONFIGURATION OF YCSB INSTANCES FOR THE FIRST EXPERIMENT

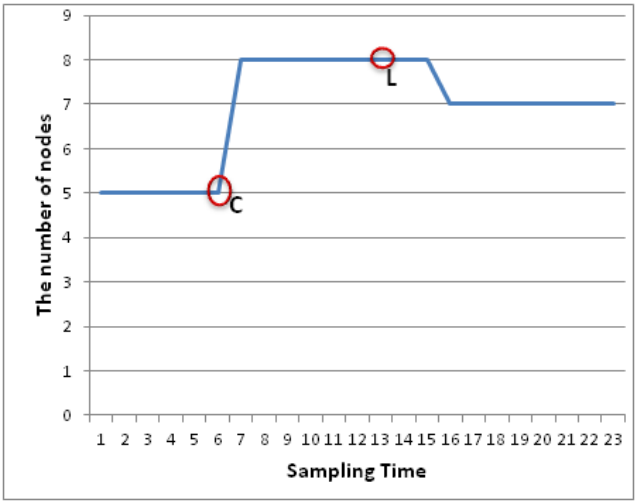

Fig. 9. The changes in the number of nodes (the first experiment)

(at point L) we increased the load. Upon the load increase, the controller added a number of nodes in order to reach a better performance and meet SLOs. Table V shows the configuration of YCSB instances for this experiment. Other parameters were the same as in the benchmark experiments described above.

VI. CONCLUSIONS AND FUTURE WORK

Cloud providers are currently offering "pay-as-you-go" access to their resources and services. In order to meet SLOs (e.g., desired service time) and to leverage this pricing model, Cloud-based applications should be elastic. In this paper, we have presented the design, implementation, and evaluation of a feedback controller in order to automate elasticity of a distributed key-value store called Voldemort. The design of controller addresses several issues to be considered when developing a feedback controller such as defining touchpoints (sensors and actuators), system identification, and controller implementation. We have chosen service time as a system variable to be monitored and used for control. In contrast to other approaches, in our approach the service time does not include the round-trip time that might introduce noise in the control system and cause inadequate control. We used the built-in rebalance tool of Voldemort as an actuator for our controller.

We have evaluated our feedback elasticity controller integrated with a Voldemort Cluster. Our evaluation shows that

Warm-up Period Configuration	
Parameter	**Value**
Warm-up Period (min)	30
Throughput during warm-up (Ops/Sec)	4000
Value Size during Throughput (bytes)	1024
Benchmark (YCSB) Setup for increasing Load	
Parameter	**Value**
Throughput(Ops/Sec)	500
Value size (bytes)	6144

TABLE V
THE CONFIGURATION OF YCSB INSTANCES FOR THE SECOND EXPERIMENT

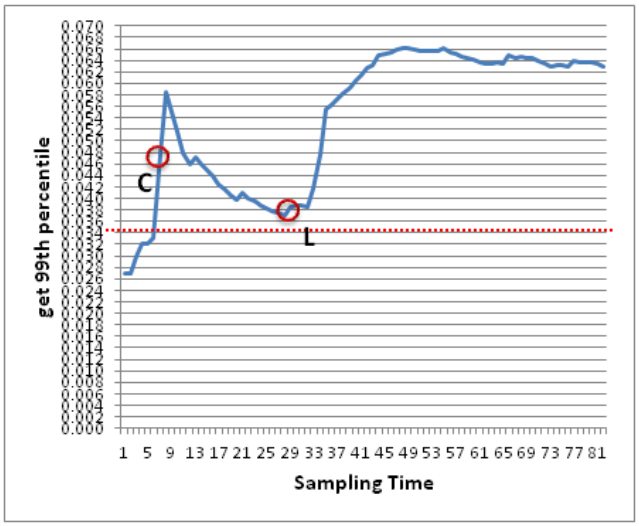

Fig. 10. The changes in the 99th percentile of read operation latency (the second experiment)

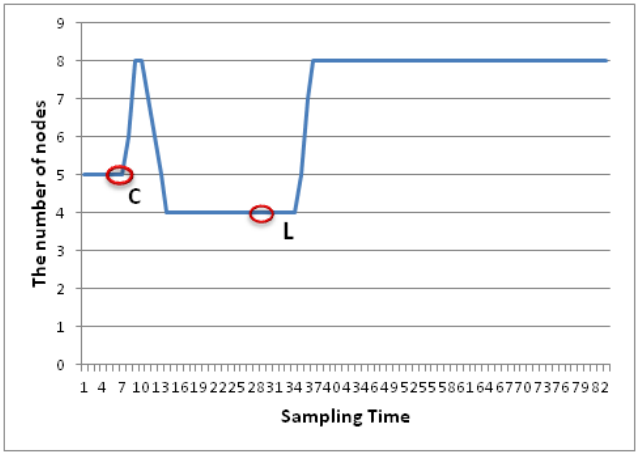

Fig. 11. The changes in the number of nodes (the second experiment)

Voldemort extended with our controller is elastic to varying workloads and reduces its cost compared to approaches based on fixed resource allocation. Evaluation results also show that our controller is also effective for reducing service time. In other words, Voldemort with our elasticity controller is able to meet the SLO, while being resource efficient.

In our future work, we intend to study in more detail the requirements for the system to be elastic and problems that one might face when automating elasticity by designing a feedback controller. One of the major problem is nonlinearities in dependency of SLO metrics (e.g., performance) on the capacity of the system (e.g., the number of servers and replicas). One of the possible solutions to this problem is to use gain scheduling, i.e., defining different gains for different operating regions (the system size).

Acknowledgements

This research has been partially funded by the Complex Service Systems project, a part of the ICT-TNG Strategic Research Areas initiative at KTH; the End-to-End Clouds project funded by the Swedish Foundation for Strategic Research; and the RMAC project funded by EIT ICT Labs. We also thank the anonymous reviewers for their constructive comments.

References

[1] R. L. Grossman, Y. Gu, M. Sabala, and W. Zhang, "Compute and storage clouds using wide area high performance networks," *Future Generation Computer Systems*, vol. 25, no. 2, pp. 179 – –183, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X08001155

[2] A. Rastogi, "A model based approach to implement cloud computing in e-governance," *International Journal of Computer Applications*, vol. 9, no. 7, pp. 15–18, 2010. [Online]. Available: http://www.doaj.org/doaj?func=abstract&id=658965

[3] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah, "Serving large-scale batch computed data with project voldemort," in *The 10th USENIX Conference on File and Storage Technologies (FAST'12)*, February 2012.

[4] Project voldemort. [Online]. Available: http://www.project-voldemort.com/

[5] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th international conference on Autonomic computing*, ser. ICAC '10. New York, NY, USA: ACM, 2010, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/1809049.1809051

[6] B. Trushkowsky, P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "The scads director: scaling a distributed storage system under stringent performance requirements," in *Proceedings of the 9th USENIX conference on File and stroage technologies*, ser. FAST'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 12–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=1960475.1960487

[7] M. A. Moulavi, A. Al-Shishtawy, and V. Vlassov, "State-space feedback control for elastic distributed storage in a cloud environment," in *The Eighth International Conference on Autonomic and Autonomous Systems ICAS 2012*, St. Maarten, Netherlands Antilles, March 2012, pp. 18–27.

[8] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell, "Modeling virtual machine performance: challenges and approaches," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 3, pp. 55–60, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1710115.1710126

[9] R. Iyer, R. Illikkal, O. Tickoo, L. Zhao, P. Apparao, and D. Newell, "VM3: Measuring, modeling and managing VM shared resources," *Computer Networks*, vol. 53, no. 17, pp. 2873–2887, December 2009. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2009.04.015

[10] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 205–216. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807162

[11] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[12] IBM, "An architectural blueprint for autonomic computing, 4th edition," http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf, June 2006.

[13] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, September 2004.