Elasticity Controller for Cloud-Based Key-Value Stores

Conference Paper · December 2012

DOI: 10.1109/ICPADS.2012.45

CITATIONS
4

READS
21

3 authors:

Ala Arman
University of Florence
6 PUBLICATIONS **16** CITATIONS
SEE PROFILE

Ahmad Al-Shishtawy
Swedish Institute of Computer Science
33 PUBLICATIONS **167** CITATIONS
SEE PROFILE

Vladimir Vlassov
KTH Royal Institute of Technology
122 PUBLICATIONS **772** CITATIONS
SEE PROFILE

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
†Swedish Institute of Computer Science, Stockholm, Sweden
ahmad@sics.se

***Abstract*—Clouds provide an illusion of the 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 automation of elasticity of Cloud-based key-value stores. We describe all 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, Zoho and Salesforce, 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. 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 SLO; whereas, if the load decreases, a number of instances are removed from the system in order to reduce the cost.

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 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, used as a case study. In particular, the steps include defining touchpoints, and system identification. 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. For example, in [3], one of the system variables is the response time, i.e., the time that it takes to send a request from a client to an application, to

process the request, and to return the result to the client. As known, the round trip time depends on the several metrics such as physical medium, physical distance of resource and destination, the existence of interference in the system etc. These metrics do not reflect the amount of the load in the system. As a result the round-trip time is not a good option as a system variable to be monitored and used for control. In addition, only scalability has been considered. However, in an elastic system, the resources can be removed in case of low workload. In [4], the system variable monitored and used in feedback control is the CPU utilization, because as shown by the authors, the CPU utilization is highly correlated with the response time. In 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 case of low CPU utilization. However, in a Cloud environment the aforementioned correlation might not hold due to the variable performance of Cloud Virtual Machines.

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 [5], 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, based on a predefined algorithm. 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 SLOs in case of increasing workload. 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 the elastic key-value store 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 (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 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.

A. Touchpoints

According to [6], a touchpoint is defined as "an interface to an instance of a managed resource such an operating system or a server. A touchpoint implements sensor and the effector behavior for the managed resource."

In our paper, we defined a touchpoint as an interface to Voldemort in a way that we could measure the service time in each node of Voldemort and actuate by adding or removing Voldemort servers. We defined our SLO as "99th percentile of read operation latency".

B. 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. Figure 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 operations that it is compared with measured output by 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. Figure 2 shows the architecture of the framework.

Now we consider our framework in more details. Figure 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 creates various load scenarios [7].
- **Sensor:** Monitors the load by measuring the 99th percentile of read operation over a fixed period of time and then gives it to the Filter.
- **Actuator (Rebalance tool):** Gets a target cluster file from the PID controller and updates the cluster.

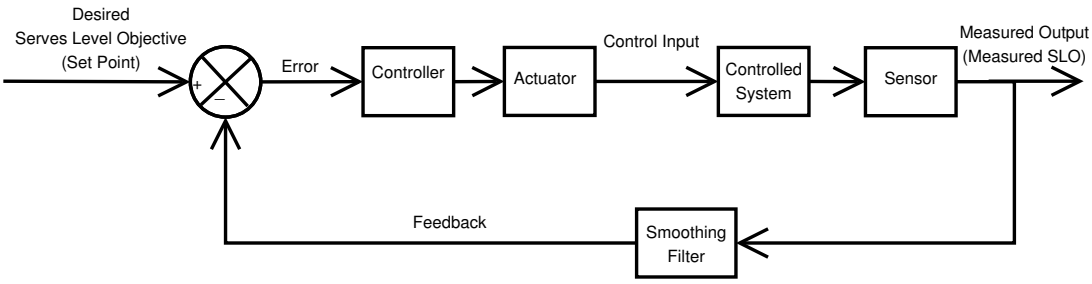

Fig. 1. Generic Control Framework

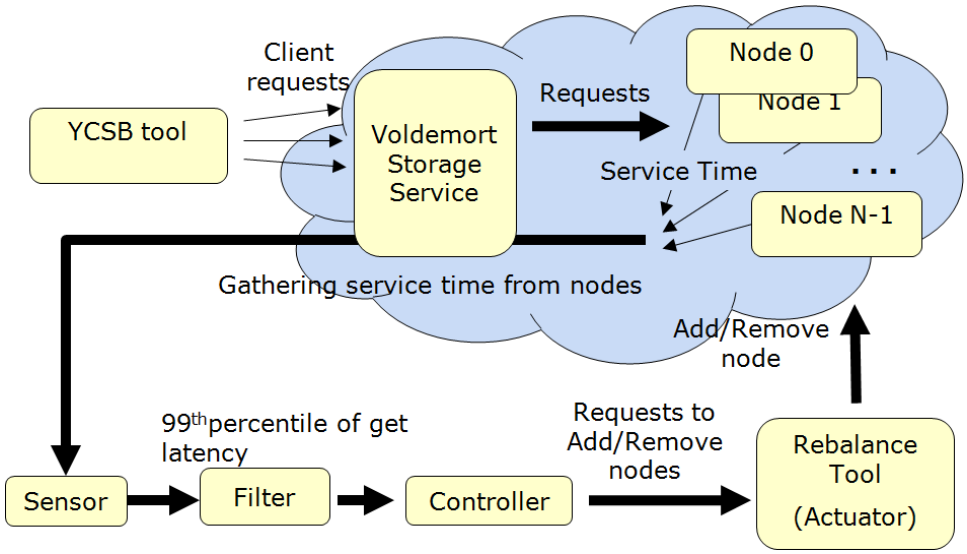

Fig. 2. Framework Architecture

Component	Implementation / tool
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

- **Filter:** It smooths the service time signal that is given to the controller by avoiding spikes in output values resulting from noise.
- **PID controller:** 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, shows how we designed and implemented or used each component of the controlling framework:

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

C. Voldemort

Voldemort is a distributed, persistent fault-tolerant non-relational key-value hash table. It is used in LinkedIn for certain high-scalability storage services where simple functional partitioning is not sufficient. The main characteristics of Voldemort are as follows.

- **Data Replication:** Data is replicated over multiple servers automatically.
- **Data Partitioning:** Data is automatically partitioned in a way that each server contains only a portion of the total data.
- **Data Versioning:** Data items are versioned to maximize data integrity in failure scenarios without compromising availability of the system
- **Node independency:** Each node is independent of other nodes with no central point of failure or coordination.
- **(Good) single node performance:** 10-20k operations/second depending on the machines, the network, the disk system, and the data replication factor.
- **Horizontal Scalability:** It provides a rebalance tool that adds/removes nodes from the cluster of nodes.

D. Yahoo! Cloud Serving Benchmark (YCSB)

In order to generate client requests, we use an open source benchmark tool called Yahoo! Cloud Serving Benchmark (YCSB) [7] has been used. 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.

E. Actuator

In the context of our paper, an actuator is a component that can make some change 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 removes 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.

F. Sensor

We used the touchpoints in order to monitor the load in the cluster of Voldemort nodes by measuring service time (which is the get 99th percentile time). In other word, the sensor in Figure (20) uses them to sense the load and give it to the Filter.

G. Filter

There are some times that measured values by the Sensor, are not smooth enough because of noise or other special circumstances. We used a filter to reduce the influence of these undesirable situations. Therefore they can decrease the fluctuations in the measured output values. We designed Filter component in a way that we gave more weight to the previous Filter output and less weight to the new filter input. We can show this concept in a mathematical way as following:

$$\text{Filter output} = 0.9*(\text{Previous Filter Output})+0.1*(\text{new Filter Input})$$

H. 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. In the following sections, first we will discuss about the system identification process and then the steps of designing the controller are presented in detail.

III. System Identification

Intuitively, 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. More formally, it can be considered as a link between application in real world and model abstractions. In this paper, we used Black-box approach [8]. In this approach instead of knowing all properties of the system, a mathematical approach is taken and a model is proposed based on different input and output. We used this approach in our work as the studied system (Voldemort) is a complex system with unknown parameters. In any system identification with black-box approach, there are some steps that should be followed:

- Determining the input/output of the system.
- Experiment and collect the input and output of the system which will be discussed in the next chapter.
- preprocess data and select the useful part of data
- Design a model based on the data which has 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 the Figure 3. As we can see in this Figure:

- 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 get 99th percentile time 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.

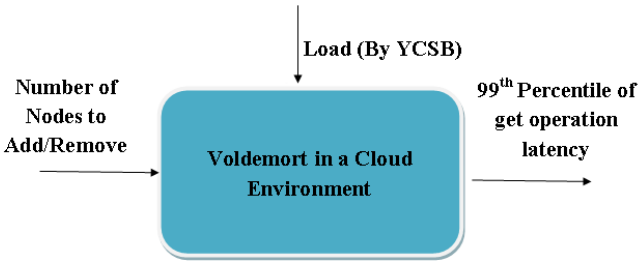

Fig. 3. System Input/output

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 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.

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

The dynamics of basic state space model is as following:

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

$x(k)$ is the vector of state variables, $u(k)$ is output Matrix, $Y(k)$ the Input Matrix , $D(k)$ is the Delay Scalar. Using `PEM` method to model a system based on, we should we should specify two parameters, delay and the order of the system. Delay is a vector of the number of input delays which is zero as is considered in discrete models. The order of the system is estimated by following command in Matlab. This command estimates the parameters for the state-space model:

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

This command specifies the best order for the system. `dat` is an object which is created by the `idata` command which takes the output vector and the input vector as its input. By using `pem` command, we found out 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 work space, now it is time to estimate initial steps of the model. It is specified by 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 build the model using the system identification tool, we can determine A, B, C and D (used in equation 1 and 2) using the `ss` function in Matlab. It creates some state-space object from `PEM` model that we have built.

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

`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 building a transfer function of the system that models a linear system. It is calculated by the `tf` command in Matlab which takes the state space model object as input (which is taken from `ss` command). It converts the state space model 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 Figure 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 SLO (reference point) as 0.036. Then we can set the gain parameters Kp, Ki and Kd of the PID controller by tuning the controller. After tuning the gain parameters using PID Tuner, 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 sections, the parameters selected empirically by running various experiments led us to the most efficient parameters:

B. Node Setup

Our cluster consisted from eight nodes running on eight machines. In the following table, the node setup of our experiment is presented:

C. Benchmark Setup

For effective benchmarking, we used two powerful machines to load the Voldemort nodes as much as benchmark parameters were set. Table (12), shows the setup used in our benchmark:

D. Benchmark Experiment

We started with three active nodes and run the controller and two YCSB instances with a specific throughput. Then with the delay D=30 (min) between each rebalancing, node 4th, 5th, 6th, 7th and 8th were added. Afterwards, with the same delay, node 8th, 7th, 6th, 5th, 4th are removed to cover the all range of input.

Figures 5 and 6 show the results of experimental design and data acquisition. The X-axis shows the sampling time.

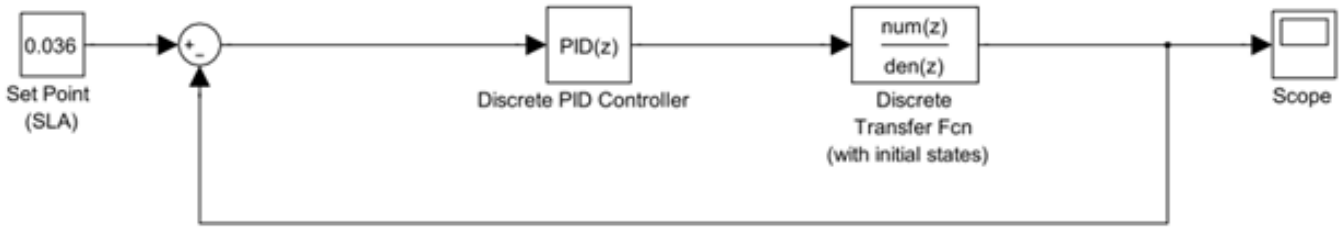

Fig. 4. Graphical design of the PID controller using Simulink

Machine Setup	
Parameter	**Value**
Processor	4
CPU Cores	4
model name	Intel Core2 Quad Q9400 @ 2.66GHz
Cache size (MB)	3
Memory (MB)	3887
Swap (MB)	8001
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
Swap	20002
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

Figure 5, Y-axis shows the number of nodes and In Figure 6 shows the average get 99th percentile time. As we can see, by increasing the number of nodes, the get 99th percentile time decreases. Similarly, by removing some nodes, get 99th percentile time increases.

As was mentioned in the previous chapter, the model that

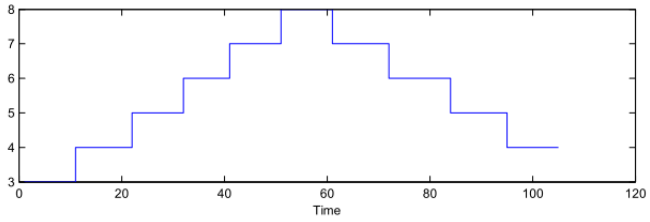

Fig. 5. The changes in the number of nodes in the experimental design and data acquisition

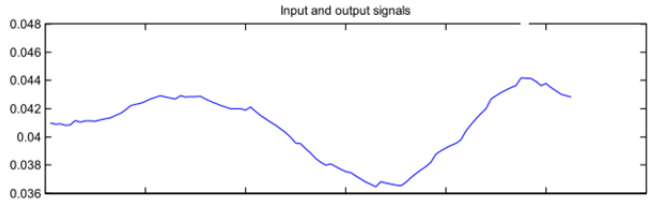

Fig. 6. The changes in get 99th percentile time changes in the experimental design and data acquisition

is created by system identification tool is a PEM model in State Space structure. Figure 7, shows the output model which compares the measured outputs and simulated model. Y-Axis shows the get 99th percentile time. As we can see that there is a good consistency between measured and simulated model.

E. First Experiment (Low Workload)

The experiment was as following. We ran the sensor for half an hour and then the controller started at point C. After about 40 minutes we decreased the load. Therefore, the controller removes some appropriate nodes not to waste resources and meet pay-as-you-go property. Table IV shows the configuration of YCSB instances for this experiment. Other parameters were like previous experiments. Figures 8 and 9 show the results of the first experiment.

Another important point in our experiments is that, as we applied a filter in our framework, the controller sees the filtered values and uses them. That is why that the changes in the number of loads were done with a little delay; but we could see smoother outputs with less spikes instead.

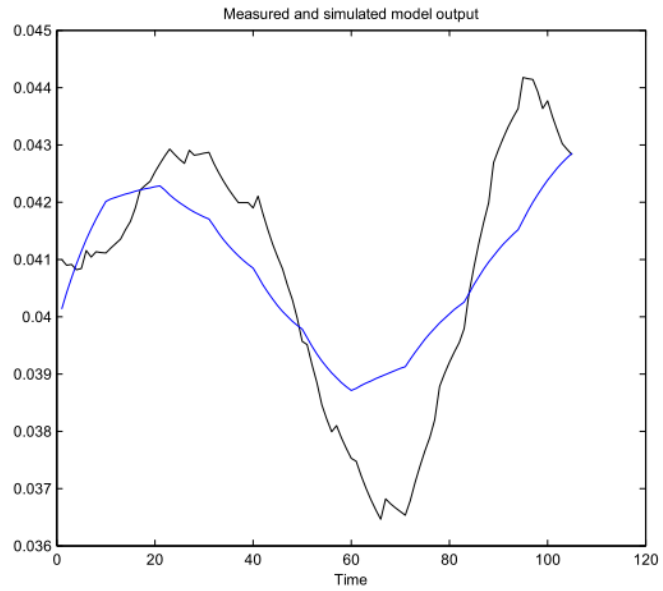

Fig. 7. Model output. The black curve shows the measured output values and the blue one shows the output of simulated model by Matlab

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

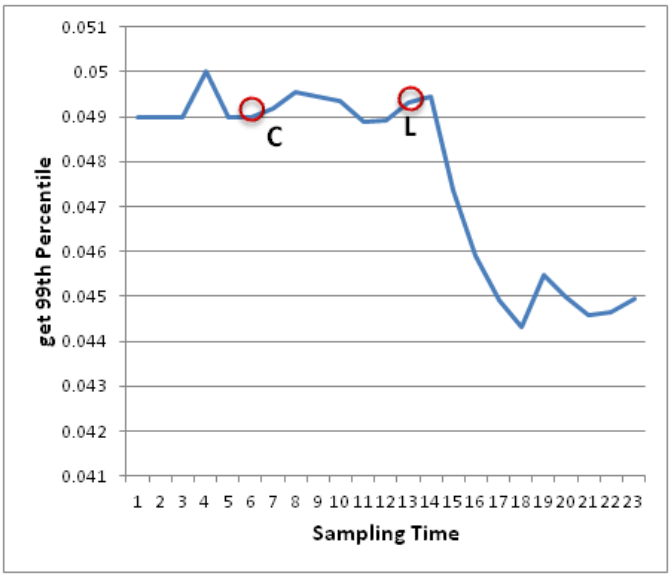

Fig. 8. The changes in get 99th percentile time(the first experiment)

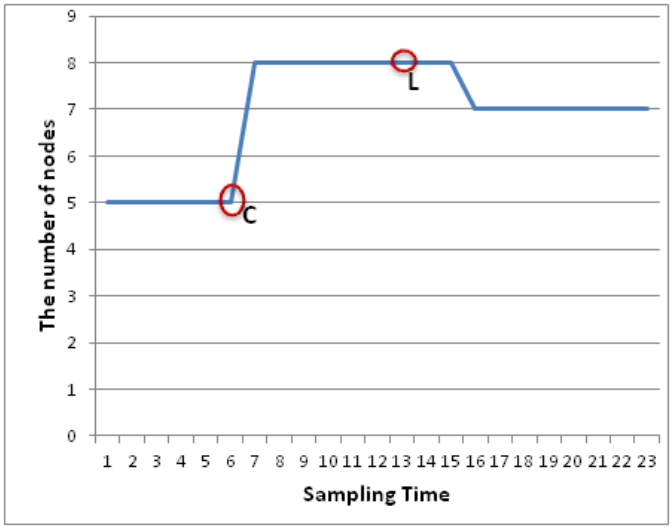

Fig. 9. The changes in the number of nodes (the first experiment)

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 increasing Load	
Parameter	**Value**
Throughput(Ops/Sec)	500
Value size (bytes)	6144

TABLE V
THE CONFIGURATION OF YCSB INSTANCES FOR THE SECOND EXPERIMENT

F. Second Experiment (High Workload)

The experiment was as following. We ran the sensor for half an hour and then the controller started at point C. After about 110 minutes we increased the load. Therefore, the controller added some appropriate nodes to reach a better performance as a result. Table V shows the configuration of YCSB instances for this experiment. Other parameters were like previous experiments. Figures 10 and 11 show the results of the second experiment.

VI. CONCLUSIONS AND FUTURE WORK

Cloud providers are currently offering "pay-as-you-go" access to their resources and services while guaranteeing meeting SLO metrics, e.g. performance. 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

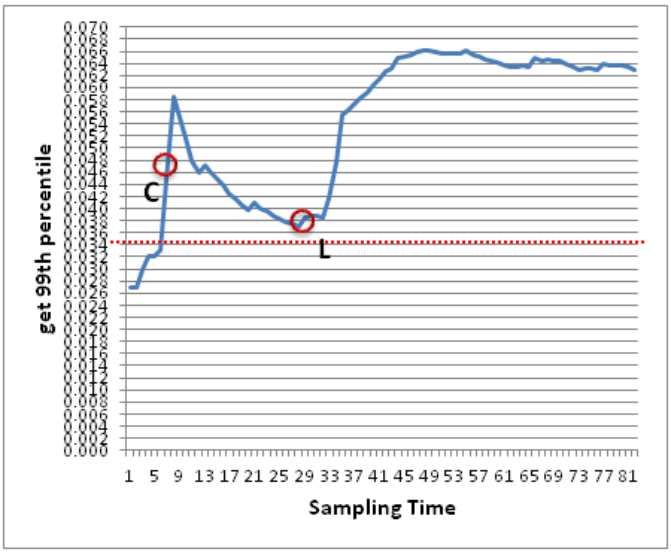

Fig. 10. The changes in get 99th percentile time(the second experiment)

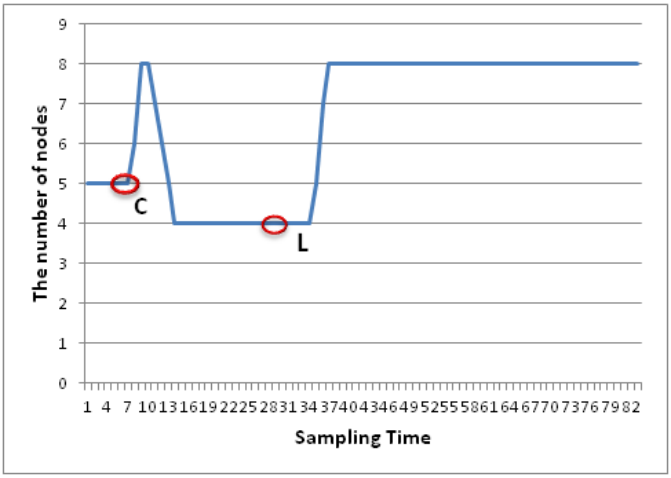

Fig. 11. The changes in the number of nodes (the second experiment)

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 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 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).

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] 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

[4] 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

[5] 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.

[6] 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.

[7] 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

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