State-Space Feedback Control for Elastic Distributed Storage in a Cloud Environment

M. Amir Moulavi, Ahmad Al-Shishtawy, and Vladimir Vlassov
KTH Royal Institute of Technology
Stockholm, Sweden
Email: {moulavi,ahmadas,vladv}@kth.se

***Abstract*—Elasticity in Cloud computing is an ability of a system to scale up and down (request and release resources) in response to changes in its environment and workload. Elasticity can be achieved manually or automatically. Efforts are being made to automate elasticity in order to improve system performance under dynamic workloads. In this paper, we report our experience in designing an elasticity controller for a key-value storage service deployed in a Cloud environment. To design our controller, we have adopted a control theoretic approach. Automation of elasticity is achieved by providing a feedback controller that automatically increases and decreases the number of nodes in order to meet service level objectives under high load and to reduce costs under low load. Every step in the building of a controller for elastic storage, including system identification and controller design, is discussed. We have evaluated our approach by using simulation. We have developed a simulation framework EStoreSim in order to simulate an elastic key-value store in a Cloud environment and be able to experiment with different controllers. We have examined the implemented controller against specific service level objectives and evaluated the controller behavior in different scenarios. Our simulation experiments have shown the feasibility of our approach to automate elasticity of storage services using state-space feedback control.**

Keywords-elasticity; key-value store; Cloud; state-space feedback control

I. INTRODUCTION

Web-based services frequently experience high workloads during their lifetime. A service can become popular in just an hour, and the occurrence of such high workloads has been observed more and more recently. Cloud computing has brought a great solution to the problem by requesting and releasing VM (Virtual Machine) instances that provide the service on-the-fly. This helps to distribute the loads among more instances. However, the high level load typically does not last for long and keeping resources in the Cloud costs money. This solution has led to Elastic Computing where a system running in the Cloud can scale up and down based on a dynamic property that is changing from time to time.

In 2001, P. Horn from IBM [1] marked the new era of computing as Autonomic Computing. He pointed out that the software complexity would be the next challenge of Information Technology. Growing complexity of IT infrastructures can undermine the benefits IT aims to provide. One traditional approach to manage the complexity is to rely on human intervention. However, considering the expansion rate of software, there would not be enough skilled IT staff to tackle the complexity of its management. Moreover, most of the real-time applications require immediate administrative decision-making and actions. Another drawback of the growing complexity is that it forces us to focus on management issues rather than improving the system itself.

Elastic Computing requires automatic management that can be provided using results achieved in the field of Autonomic Computing. Systems that exploit Autonomic Computing methods to enable automated management are called self-managing systems. In particular, such systems can adjust themselves according to the changes of the environment and workload. One common and proven way to apply automation to computing systems is to use elements of control theory. In this way a complex system, such as a Cloud service, can be automated and can operate without the need of human supervision.

In this paper, we report our experience in designing an elasticity controller for a key-value storage service deployed in a Cloud environment. To design our controller, we have adopted a control theoretic approach. Automation of elasticity is achieved by providing a feedback controller that continuously monitors the system and automatically changes (increases or decreases) the number of nodes in order to meet Service Level Objectives (SLOs) under high load and to reduce costs under low load. We believe that this approach to automate elasticity has a considerable potential for practical use in many Cloud-based services and Web 2.0 applications including services for social networks, data stores, online storage, live streaming services.

Our second contribution presented in this paper is an open-source simulation framework called EStoreSim (Elastic key-value Store Simulator) that allows developers to simulate an elastic key-value store in a Cloud environment and be able to experiment with different controllers.

The rest of the paper is organized as follows. In Section II, we define the problem of automated elasticity and describe the architecture of an elastic Cloud-based key-value store with feedback control. Section III presents different approaches to system identification. In Section IV, we show how we construct a state-space model of our elastic key-value store. We continue in Section V by presenting the controller designing for our storage. Section VI summarises steps of controller design including system identification. In Section VII, we describe the implementation of our simulation framework

 ISBN: 978-1-61208-187-8

EStoreSim. Experimental results are presented in Section VIII followed by a discussion of related work in Section IX. Finally, our conclusion and our future work are given in Section X.

II. Problem Definition and System Description

Our research reported here aims at automation of elasticity of a key-value store deployed in a Cloud environment. We want to automate the management of elastic storage instances depending on workload. a Cloud environment allows the system that is running in the Cloud to scale up and down in few minutes in response to load changes. In-time and proper decisions regarding the size of the system in response to the changes in the workload is very critical when it comes to enterprise and scalable applications.

In order to achieve elasticity of a key-value store in the Cloud, we adopt a control theoretic approach to designing a feedback controller that automatically increases and decreases the number of storage instances in response to changes in workload in order to meet SLOs under high load and to reduce costs under low load. The overall architecture of the key-value store with the feedback controller is depicted in Fig. 1.

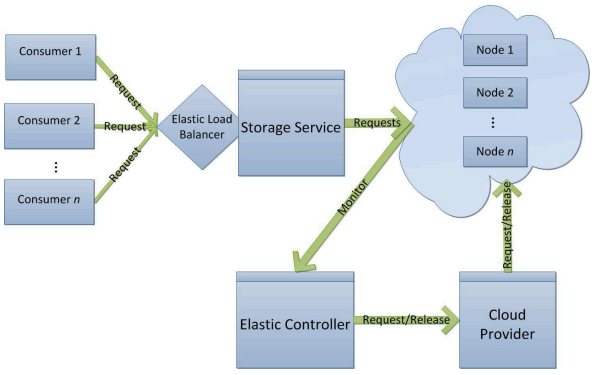

Fig. 1. Architecture of the Elastic Storage with feedback control of elasticity

End-users request files that are located in the storage Cloud nodes (instances). All the requests arrive at the Elastic Load Balancer (ELB) that sits in front of all storage instances. The Elastic Load Balancer decides to which instance the request should be dispatched. In order to do this, the Elastic Load Balancer tracks the CPU load and the number of requests sent previously to each instance and based on that it determines the next node that can serve the incoming request. In addition to the performance metrics that it tracks, ELB has the file tables with information about file replica locations since more than one instance can have a replica of the same file in order to satisfy the replication degree.

The Cloud Provider (Fig. 1) is an entity that is responsible for launching a new storage instance or terminating the existing one on requests of the Elasticity Controller.

Our system contains the Elasticity Controller, which is responsible for controlling the number of storage instances in the Cloud in order to achieve the desired SLO (e.g., download time). The Controller monitors the performance of the storage instances (and indirectly the quality of service) and issues requests to scale the number of instances up and down in response to changes in the measured quality of service (compared to the desired SLO). These changes are caused by changes in the workload, which is not controllable and is considered to be a disturbance in terms of control theory.

In the following two sections, we provide the relevant background and present steps of the *design of the controller* including *system identification* [2].

III. Approaches to System Identification

In this section, we present methods of system identification, which is the most important step in the design of a controller. It deals with how to construct a model to identify a system. System identification allows us to build a mathematical model of a dynamic system based on measured data. The constructed model contains a number of transfer functions, which define how the output depends on past/present inputs and outputs. Based on the transfer functions and desired properties and objectives, a control law is chosen. System identification can be performed using one of the following approaches.

First principle approach is one of the de facto approaches to identification of computer systems [3]. It can be considered as a consequence of the queue relationship. The first principle approach is developed based on knowledge of how a system operates. For example, this approach has been used in some studies and systems like [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. However, there are some shortcomings with this approach that have been stated in [2]. It is very difficult to construct a first principle model for a complex system. Since this approach considers detailed information about the target systems, it requires an ongoing maintenance by experts. Furthermore, this approach does not address model validation.

Empirical approach starts by identifying the input and output parameters like the first principle approach. But rather than using a transfer function, an *autoregressive moving average* (ARMA) model is built and common statistical techniques are employed to estimate the ARMA parameters [2]. This approach is also known as *Black Box* [3]; and it requires minimal knowledge of the system. Most of the systems in our studies have employed a black-box approach rather than a first-principle approach for system identification, e.g., [2], [15], [16], [17], [18], [19]. This is mainly because the relationship between inputs and outputs of the system is complex enough so that the first-principle system identification cannot be done easily. One of the empirical approaches is to build a State-Space Model, which requires more knowledge of the internals of the system. We use the state-space model approach for system identification as described in the next section.

IV. State-Space Model of the Elastic Key-Value Store

A state-space model provides a scalable approach to model systems with a large number of inputs and outputs [3]. The state-space model allows dealing with higher order target systems without a first-order approximation. Since the studied system executes in a Cloud environment, which is complex and

 ISBN: 978-1-61208-187-8

dynamic in a sense of dynamic set of VMs and applications, we have chosen state-space modeling as the system identification approach. Another benefit of using the state-space model is that it can be extended easily. Suppose that after the model is built, we find more parameters to control the system. This can be accommodated by the state-space model without affecting the characteristic equations as shown later in Section VI where we summarize a generic approach for system identification and controller design

The main idea of the state-space approach is to characterize how the system operates in terms of one or more variables. These variables may not be directly measurable. However, they can be sufficient in expressing the dynamics of the system. These variables are called *state variables*.

A. State Variables and the State-Space Model

In order to define the state variables for our system, first we need to define the inputs and measured outputs since the state variables are related to them. In particular, state variables can be used to obtain the measured output. It is possible for a state variable to be a measured output like it is in our case.

In our case, the system input is the number of nodes (instances) denoted by $\texttt{NN}(k)$ at time k. The measured system outputs (and hence state variables) are the following:

- *average CPU load* $\texttt{CPU}(k)$: the average CPU load of all instances currently running in the Cloud during the time interval $[k-1, k]$;
- *interval total cost* $\texttt{TC}(k)$: the total cost of all instances during the time interval $[k-1, k]$;
- *average response time* $\texttt{RT}(k)$: the average time required to start a download during the time interval $[k-1, k]$.

The value of each state variable at time k is denoted by $x_1(k)$, $x_2(k)$ and $x_3(k)$. The offset value for input is $\bar{u}_1(k) = \texttt{NN}(k) - \widehat{\texttt{NN}}$, where $\widehat{\texttt{NN}}$ is the operating point for the input. The offset values for outputs are

$$\bar{y}_1(k) = \texttt{CPU}(k) - \widehat{\texttt{CPU}} \quad (1)$$

$$\bar{y}_2(k) = \texttt{TC}(k) - \widehat{\texttt{TC}} \quad (2)$$

$$\bar{y}_3(k) = \texttt{RT}(k) - \widehat{\texttt{RT}} \quad (3)$$

where $\widehat{\texttt{CPU}}$, $\widehat{\texttt{TC}}$ and $\widehat{\texttt{RT}}$ are operating points for corresponding outputs.

The state-space model uses state variables in two ways [3]. First, it uses state variables to describe the dynamics of the system and how $\mathbf{x}(k+1)$ can be obtained from $\mathbf{x}(k)$. Second, it obtains the measured output $\mathbf{y}(k)$ from state $\mathbf{x}(k)$.

State-space dynamics for a system with n states is described as follows

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \quad (4)$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) \quad (5)$$

where $\mathbf{x}(k)$ is a $n \times 1$ vector of state variables, $\mathbf{A}$ is a $n \times n$ matrix, $\mathbf{B}$ is a $n \times m_I$ matrix, $\mathbf{u}(k)$ is a $m_I \times 1$ vector of inputs, $\mathbf{y}$ is a $m_O \times 1$ vector of outputs and $\mathbf{C}$ is a $m_O \times n$ matrix.

According to equations 4 and 5, we can describe dynamics of our system as follows:

- Average CPU Load ($\texttt{CPU}$) is dependant on the number of nodes in the system and previous CPU load, thus it becomes

$$\begin{aligned} x_1(k+1) = \texttt{CPU}(k+1) = \\ a_{11}\texttt{CPU}(k) + \\ b_{11}\texttt{NN}(k) + \\ 0 \times \texttt{TC}(k) + 0 \times \texttt{RT}(k) \end{aligned} \quad (6)$$

- Total Cost ($\texttt{TC}$) is dependant on the number of nodes in the system (the more nodes we have, the more money we should pay) and the previous $\texttt{TC}$, hence it becomes

$$\begin{aligned} x_2(k+1) = \texttt{TC}(k+1) = \\ a_{21}\texttt{TC}(k) + \\ b_{21}\texttt{NN}(k) + \\ 0 \times \texttt{RT}(k) + 0 \times \texttt{CPU}(k) \end{aligned} \quad (7)$$

- Average Response Time ($\texttt{RT}$) is dependant on the number of nodes in the system and the CPU load, so it is

$$\begin{aligned} x_3(k+1) = \texttt{RT}(k+1) = \\ a_{31}\texttt{CPU}(k) + a_{33}\texttt{RT}(k) + \\ b_{31}\texttt{NN}(k) + \\ 0 \times \texttt{TC}(k) \end{aligned} \quad (8)$$

In each equation (6, 7, 8) terms with zero factor include those state variables that do not affect the corresponding state variable definition. Thus their coefficient is zero. This is to ensure that there is no relation between those state variables or the relation is negligible and can be ignored. Their presence in the equations is for the sake of clarity and completeness. In order to prove that there is no relation or that it is negligible one should do a sensitivity analysis to investigate this, but it is out of the scope of this paper.

The output for the system at each time point k is equivalent to the corresponding state variable:

$$\mathbf{y}(k) = I_3\mathbf{x}(k) \quad (9)$$

The outputs are the same as the internal state of the systems at each time. That is why the matrix $\mathbf{C}$ is an identity matrix, i.e., a diagonal matrix of 1's. The matrices of coefficients are:

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ a_{31} & 0 & a_{33} \end{bmatrix} \quad (10)$$

$$\mathbf{B} = \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} \quad (11)$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (12)$$

 ISBN: 978-1-61208-187-8

B. Parameter Estimation

In Section IV-A, we have derived the State-Space model (Equations 6-12) that describes the dynamics of an elastic key-value store. There are two matrices **A** and **B** that contain the unknown coefficients for the equations 6-8. In order to use the model to design the controller we need to estimate the coefficient matrices **A** and **B**.

Parameter estimation is done using experimental data. In this research, we use data obtained from the simulation framework EStoreSim that we have built, rather than from a real system, because the major focus is on controller design and the simulation framework allows us to experiment with different controllers. We have implemented a simulation framework EStoreSim (described in Section VII) of a Cloud system. Using the framework we can obtain experimental data for system identification.

To get the data, we have designed and run an experiment, in which we feed the system with an input signal and observe the output and internal state variable periodically. We change the input (which is the number of nodes in the system) by increasing it from a small number of nodes a to a large number of nodes b and then back from b to a in a fixed period of time, and measure outputs (CPU load, cost, and response time). In this way, we ensure the complete coverage of the output signals in their operating regions by the input signal (the number of nodes). Load should be generated according to an arbitrary periodic function to issue a number of downloads per seconds. The period of the function should be chosen such that at least one period is observed during the time of changing the input between $[a, b]$.

For example, using the modeler component of our framework EStoreSim (Section VII), we scale up the number of nodes from 2 to 10 and then scale down from 10 to 2. Every 225 seconds a new node is either added or removed (depending on whether we scale up or down); sampling of training data (measuring outputs) is performed every 10 seconds.

When identifying the system, the workload is modeled as a stream of requests issued by the request generator component where the time interval between two consecutive requests forms a triangle signal in the range $[1, 10]$ seconds as follows: the first request is issued after 10 seconds, the second after 9 seconds, etc. The requests are received by the load balancer component in the Cloud provider component. After each scaling up/down the system will experience 2 triangle loads of requests between 1 to 10 seconds. The time needed to experience 2 triangles is $4\sum_{i=1}^{10} i$, which is 220 seconds. That is why we have selected 225 seconds as the action time.

Once training data are collected, they can be used to compute the matrices **A** and **B** using the *multiple linear regression* method. We use the `regress(y,X)` function of Matlab to calculate matrices:

$$\mathbf{A} = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.724 & 0 \\ 5.927 & 0 & 0.295 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 2.3003 \\ 0.0147 \\ 77.8759 \end{bmatrix}$$

V. Controller Design

In this section, we describe how the feedback controller for the elastic storage deployed in a Cloud environment is designed. The controller design starts by choosing an appropriate controller architecture according to system properties. There are three common architectures for state-space feedback control, namely, *Static State Feedback*, *Precompensated Static Control* and *Dynamic State Feedback*. A good comparison between these architectures can be found in [3]. A close investigation in this comparison reveals that dynamic state feedback control is more suitable for a Cloud system since it has disturbance rejection that the other two architectures lack. Disturbance (in terms of control theory) is observed in a Cloud in the form of changes in the set of virtual machines and workload of Cloud applications. Thus we choose dynamic state feedback control as our controller architecture for autonomic management of elasticity.

A. Dynamic State Feedback

Dynamic state feedback can be viewed as a State-Space analogous to PI (Proportional Integral) control that has good disturbance rejection properties. It both tracks the reference input and rejects disturbances. We need to augment the state vector with the control error $e(k) = r - y(k)$ where r is the reference input. We use integrated control error, which describes the accumulated control error. The integrated control error is denoted by $x_I(k)$ and computed as

$$x_I(k+1) = x_I(k) + e(k)$$

The augmented state vector is $\begin{bmatrix} \mathrm{x}(k) \\ x_I(k) \end{bmatrix}$. The control law is

$$u(k) = -\begin{bmatrix} \mathrm{K}_p & K_I \end{bmatrix} \begin{bmatrix} \mathrm{x}(k) \\ x_I(k) \end{bmatrix} \tag{13}$$

where K_p is the feedback gain for $\mathrm{x}(k)$ and K_I is the gain associated with $x_I(k)$.

B. LQR Controller Design

An approach to controller design is to focus on the trade-off between control effort and control errors. The *control error* is determined by the squared values of state variables, which are normally the difference from their operating points. The *control effort* is quantified by the square of $u(k)$, which is the offset of the control input from the operating point. By minimizing control errors we improve accuracy and reduce both settling times and overshoot and by minimizing control effort, system sensitivity to noise is reduced.

Least Quadratic Regulation (LQR) design problem is parametrized in terms of relative cost of control effort (defined by matrix **R**) and control errors (defined by matrix **Q**). The quadratic cost function to minimize is the following [3]:

 ISBN: 978-1-61208-187-8

$$J = \frac{1}{2}\sum_{k=0}^{\infty}\left[\mathbf{x}^{\top}(k)\mathbf{Q}\mathbf{x}(k) + \mathbf{u}^{\top}(k)\mathbf{R}\mathbf{u}(k)\right] \qquad (14)$$

where **Q** must be positive semidefinite (eigenvalues of **Q** must be nonnegative) and **R** must be positive definite (eigenvalues of **R** must be positive) in order for J to be nonnegative.

After selecting the weighting matrices **Q** and **R**, the controller gains **K** can be computed using the Matlab `dlqr` function that takes as parameters the matrices **A**, **B**, **Q**, and **R**. The performance of the system with the designed controller can be evaluated by simulation. If the performance is not appropriate, the designer can select new **Q** and **R** and recompute the vector gain **K**.

In our example, the matrices **Q** and **R** are defined as follows:

$$\mathbf{Q} = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} 1 \end{bmatrix}$$

We have given 100 to the element that corresponds to CPU Load to emphasize that this state variable is more important compared to the others. One can give a high weight to total cost TC to trade off cost for performance. Using the Matlab `dlqr` function we compute the controller gains `K = dlqr(A, B, Q, R)`. For example, using the results of system identification in the example in Section IV-B, the controller gains (corresponding to the measured outputs of the elastic storage, CPU, TC, and RT) are:

$$\mathbf{K} = \begin{bmatrix} 0.134 & 1.470162e-06 & 0.00318 \end{bmatrix}$$

C. Fuzzy Controller

The main purpose in using an additional fuzzy controller is to optimize the control input produced by the Dynamic State Feedback Controller that we have designed in Section V-B. A fuzzy controller uses heuristic rules that define when and what actions the controller should take. The output of the Dynamic State Feedback Controller (control input) is redirected together with measured outputs to the fuzzy controller, which decides if the control input should affect the system or not. The overall architecture for controllers is demonstrated in Fig. 2.

There is one important case that the dynamic state feedback controller cannot act accordingly. Let us assume that there are some instances with high CPU load. Since the average is high, the controller will issue a control request to add a number of new instances. The new instances will be launched and will start to serve requests. But at the beginning of their life cycle they have low CPU load, thus the average CPU load that is reported back to the controller can be low. The controller then assumes that the CPU load has dropped, and it requests to remove some nodes.

A closer look at the CPU loads reveals that we can not judge the system state by only the average CPU load. Hence the fuzzy controller also takes into account the standard deviation

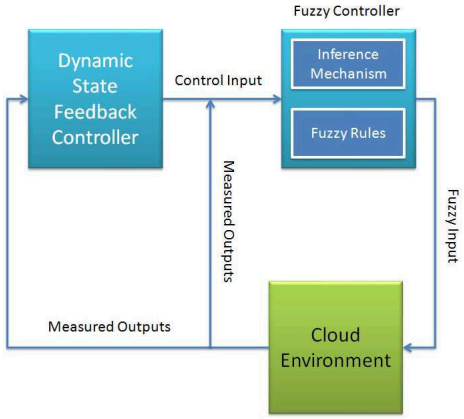

Fig. 2. Controllers Architecture

of CPU load. In this way, if the feedback controller gives an order to reduce the number of nodes when there is high standard deviation for CPU loads, the fuzzy controller will not allow this control input to affect the system, thus reducing the risk of unexpected results and confusions for the controller that may cause oscillations. This will lead to a more stable environment without so many unnecessary fluctuations.

D. Stability Analysis of Controller

A system is called stable if all bounded inputs produce bounded outputs. The BIBO theorem [3] states that for a system to be stable, its poles must lie within the unit circle (have magnitude less than 1). In order to calculate the poles for the controller we need to get the eigenvalues of matrix `A` that are 0.2951, 0.9 and 0.7247. As it is obvious from the values, all of the poles reside within the unit circle thus the controller is stable.

VI. Summary of Steps of Controller Design

This section summarizes the steps needed to design a controller for an elastic storage in a Cloud environment. The steps described below are general enough to be used to design a controller for an elastic service in a Cloud. The design process consists of two stages: system identification and controller design. The design steps are as follows: the system identification stage includes steps 1-9; and the remaining steps (10-12) belong to the stage of the controller design.

1) Study system behavior in order to identify the inputs and outputs of the system.
2) Place inputs and outputs in $\mathbf{u}(k)$ and $\mathbf{y}(k)$ vectors respectively.
3) Select n system outputs that you want to control and place them in state variable vector **x**. The outputs should be related to SLOs and performance metrics.
4) Select m system inputs that you will use to control. These system inputs will be the outputs of your controller. The system outputs should depend on the system inputs These inputs should have the highest impact in your system. In some systems there might be only one

 ISBN: 978-1-61208-187-8

input that has high impact whereas in other systems there might be several inputs that together have high impact. To assess the impact you might need to do sensitivity analysis.

5) Define state variables that describe the dynamics of the system. State variables can be equivalent to system outputs selected in step 3. Each state variable can depend on one or more other state variables and system inputs. Find the relation between the next value for a state variable to other state variables and system inputs and construct the characteristic equations as follows (see also Equation 4).

$$\begin{aligned} x_1(k+1) &= a_{11}x_1(k) + \ldots + a_{1n}x_n(k) \\ &\quad + b_{11}u_1(k) + \ldots + b_{1m}u_m(k) \\ x_2(k+1) &= a_{21}x_1(k) + \ldots + a_{2n}x_n(k) \\ &\quad + b_{21}u_1(k) + \ldots + b_{2m}u_m(k) \\ &\vdots \\ x_n(k+1) &= a_{n1}x_1(k) + \ldots + a_{nn}x_n(k) \\ &\quad + b_{n1}u_1(k) + \ldots + b_{nm}u_m(k) \end{aligned}$$

6) Place coefficients from the previous equations into two matrices **A** and **B**. Some of the coefficients can be zero:

$$\mathbf{A}_{n\times n} = \begin{bmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \ldots & a_{nn} \end{bmatrix}$$

$$\mathbf{B}_{n\times m} = \begin{bmatrix} b_{11} & \ldots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & \ldots & b_{nm} \end{bmatrix}$$

7) In order to simplify the design of controller, one can assume that outputs of the systems are equal to state variables, thus matrix **C** is an identity matrix:

$$\mathbf{C}_{n\times n} = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix}$$

8) Design an experiment, in which the system is fed with its inputs. Inputs in the experiment should be changed in such a way that they cover their ranges at least one time. A range for an input is the interval that the values of the input will most likely belong to when the system operates. The selection of ranges can be based on industry's best practices. All inputs and outputs should be measured periodically with a fixed time interval T. Store collected data for each equation in a separate file called x_i.

9) In Matlab, for each file x_i, load the file and extract each column of data in a separate matrix. Use the function `regress` to calculate the coefficients. Repeat this for every file. At the end you will have all the coefficients that are required for matrices **A** and **B**.

10) Choose a controller architecture for feedback control: such as dynamic state feedback control, which is, in our opinion, more appropriate for a Cloud based elastic service (as discussed in Section V).

11) Construct matrices **Q** and **R** as described in Section V-B. Remember to put high weights in matrix **Q** for those state variables that are of more importance.

12) Use the Matlab function `dlqr` with matrices **A**, **B**, **Q** and **R** as parameters to calculate the vector **K** of controller gains. Perform stability analysis of the controller checking whether its poles reside within the unit circle (Section V-D).

VII. EStoreSim: Elastic Key-Value Store Simulator

We have implemented a simulation framework, which we call EStoreSim, that allows developers to simulate an elastic key-value store in a Cloud environment and to experiment with different controllers. We have selected Kompics as the implementation tool. Kompics [20] is a message-passing component model for building distributed systems using event-driven programming. Kompics components are reactive state machines that execute concurrently and communicate by passing data-carrying typed events through typed bidirectional ports connected by channels. For further information please refer to the Kompics programming manual and the tutorial on its web site [20].

Implementation is done in Java and Scala languages [21] and the source is publicly available at [22]. The overall architecture of EStoreSim is shown in Fig. 3. The simulator includes the following components.

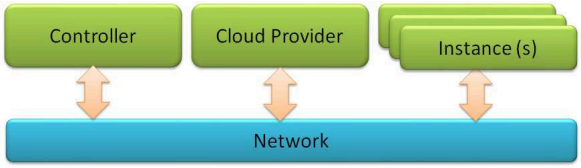

Fig. 3. Overall Architecture of the EStoreSim Simulation Framework

Cloud Instance Component represents an entire storage instance within a Cloud. The component architecture for instance is shown in Fig. 4.

Cloud Provider Component represents an important unit in the implementation. It is the heart of a simulated Cloud computing infrastructure and provides vital services to manage and administer the nodes (VM instances) within the Cloud. The Cloud provider component architecture is shown in Fig. 5.

Elasticity Controller represents the controller that can connect to the Cloud provider and retrieve information about the current nodes in the system. The main responsibility of the controller component is to manage the number of nodes currently running in the Cloud. In other words, it attempts to optimize the cost and satisfy some SLO parameters. The overall component architecture is shown in Fig. 6.

For further information on EStoreSim please refer to [22].

 ISBN: 978-1-61208-187-8

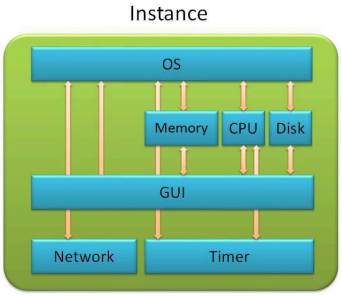

Fig. 4. Cloud Instance Component

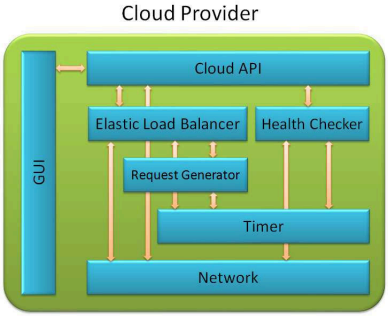

Fig. 5. Cloud Provider Component

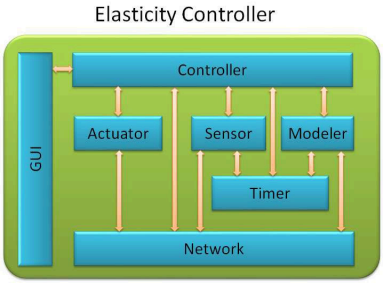

Fig. 6. Elasticity Controller Component

VIII. Experiments

We have conducted a number of simulation experiments using EStoreSim in order to evaluate how the use of an elasticity controller in a Cloud-based key-value store improves the operation of the store by reducing the cost of Cloud resources and the number of SLO violations. The baseline in our experiments is a non-elastic key-value store, i.e., a key-value store without the elasticity controller.

For evaluation experiments, we have implemented a dynamic state feedback controller with the parameters (controller gains) calculated according to the controller design steps (Section V-B). The controller is given reference values of the system outputs that correspond to SLO requirements. Values of system outputs (average CPU load `CPU`, Total Cost `TC`, and average Response Time `RT`) are fed back into the controller periodically. When the controller gets the values, it calculates and places the next value of the number of nodes `NN` on its output. The controller output is a real number that should be rounded to a natural integer. We round it down in order to save the total cost the Cloud generates. One can assume two boundaries, which are defined as follows:

- L (Lower boundary): the minimum number of instances that should exist in the Cloud at all times;
- U (Upper boundary): the maximum number of instances that is allowed to exist in the Cloud at all times.

Hence if the value of controller output is smaller than L or greater than U, then the value should be discarded. If the calculated output of the controller is Θ, the number of nodes is defined as follows:

$$\mathtt{NN} = \begin{cases} L & \text{if} \quad \Theta \leqslant L \\ \Theta & \text{if} \quad L < \Theta < U \\ U & \text{if} \quad U \leqslant \Theta \end{cases} \tag{15}$$

If the number of current nodes in the system is $\mathtt{NN}'$ and the control input (output of the controller) is `NN`, then the next control action is determined as follows:

$$\text{Next action} = \begin{cases} \text{scale up with } \mathtt{NN} - \mathtt{NN}' \text{ nodes} & \text{if} \quad \mathtt{NN}' < \mathtt{NN} \\ \text{scale down with } \mathtt{NN}' - \mathtt{NN} \text{ nodes} & \text{if} \quad \mathtt{NN} < \mathtt{NN}' \\ \text{no action} & \text{otherwise} \end{cases} \tag{16}$$

We have conducted two series of experiments to prove our approach to elasticity control. By these experiments we check whether the elasticity feedback controller operates as expected. In the first series (which we call SLO Experiment), the load is increased to a higher level. This increase is expected to cause SLO violation that is detected by the feedback controller, which adds nodes in order to meet SLO under high load. In the second series (which we call Cost Experiment), the load decreases to a lower level. This causes the controller to release nodes in order to save cost under low load. The instance configuration for these experiments are as follows:

- CPU frequency: 2 GHz;
- Memory: 8 GB;
- Bandwidth: 2 MB/s;
- Number of simultaneous downloads: 70.

There are 10 data blocks in the experiments with sizes between 1 to 5 MB. Note that the same configuration is used in the system identification experiments.

A. SLO Experiment: Increasing Load

In this series we conducted two experiments: one with controller and another without controller. In the results and figures presented below, they are denoted by `w/controller` and `w/o controller`, respectively. Each experiment starts with three warmed up instances. By a warmed up instance we mean that in this instance each data block is requested at least once thus it resides in the memory of this instance.

Workload that is used for this experiment is of two levels: normal and high. Under the normal load the time interval between consecutive requests is selected from a uniform random distribution in the range [10, 15] seconds that corresponds to an average request rate of 4.8 requests per minute. Under the high load the time interval between consecutive requests is selected from a uniform random distribution in the range [1, 5] seconds that corresponds to an average request rate of 20 requests per minute. The experiment starts with normal load and after 500 seconds the workload increases to the high level. This is shown in Fig. 7.

Sensing of instance output is done every 25 seconds. In the case of controller, actuation is performed every 100 seconds. Thus there are 4 sets of measured data at each actuation time that the controller should consider. In order to calculate values of the system output, the controller computes averages of data

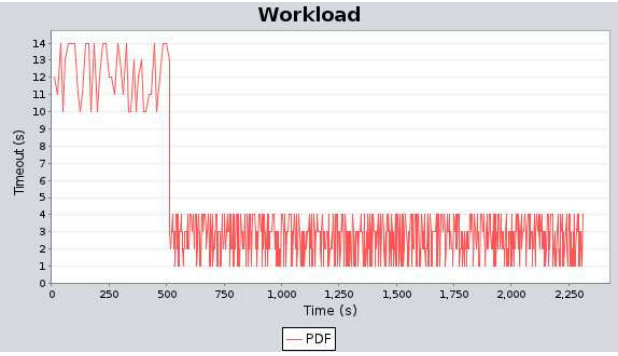

Fig. 7. SLO Experiment Workload

TABLE I
SLO VIOLATIONS

SLO Parameter Violation (%)	w/ Controller	w/o Controller
CPU Load	17.94	72.28
Response Time	2.12	7.073
Bandwidth	35.89	74.69

sets. The duration of each experiment is 2000 seconds with warm up of 100 seconds. SLO requirements are as follows:

- Average CPU Load: $\leqslant 55\%$
- Response Time: $\leqslant 1,5$ seconds
- Average Bandwidth per download: $>$ 200000 B/s

For each experiment the percentages of SLO violations are calculated for each aforementioned SLO requirement based on Equation 17. The result is shown in Table I.

$$SLO\ Violations = 100\% \times \frac{Number\ of\ SLO\ Violations}{Total\ Number\ of\ SLO\ Checks} \quad (17)$$

Checking of SLO is done at each estimate (sensing) of the *Average CPU Load* and *Average Bandwidth per download* and each estimate of *Response Time*.

This experiment gives us interesting results that are discussed in this section. `NL` and `HL` in figures 8-12 indicate periods of *Normal Load* and *High Load* respectively.

Fig. 8 depicts the Average CPU Load for the aforementioned experiments. The Average CPU Load is the average of all nodes' CPU Loads at each time the sensing is performed. As one can see in Fig. 8, CPU loads for the experiment with the controller is generally lower than the same experiment without the controller. This is due to the controller that launches new instances under high workloads causing a huge drop in average CPU Load.

Fig. 9 depicts the Average Response Time for the experiments. By response time we mean the time that it takes for an instance to respond to a request that download is started and not the actual download time. As it is seen from the diagram, the average response time for the experiment with the controller is generally lower than the experiment without controller. This is because in case of having a fixed number of instances (3 in this experiment), there would be congestion by

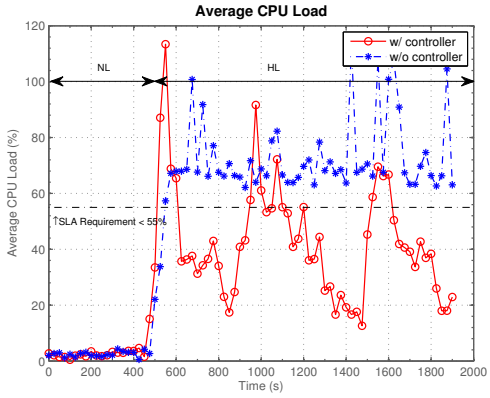

Fig. 8. SLO Experiment - Average CPU Load

	w/ controller	w/o controller
Total Cost ($)	14.4528	8.6779

TABLE II
TOTAL COST FOR EACH SLO EXPERIMENT

the number of requests an instance can process. This increases the responsivity of an instance. However, in the case that the controller launches new instances, no instance will actually go under high number of requests.

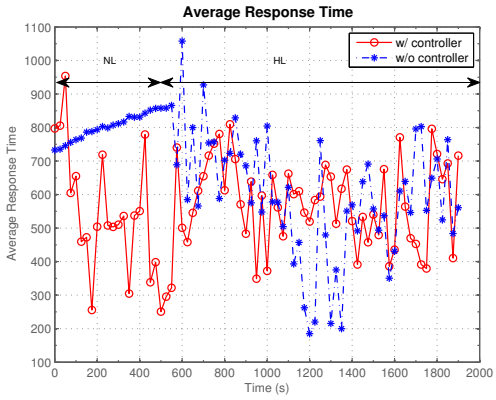

Fig. 9. SLO Experiment - Average Response Time

Fig 10 shows the total cost for the experiments. Interval total cost means that total cost is calculated for each interval in which the senses are done. As can be observed from the diagram, the interval total cost for the experiment with the controller is much higher than the experiment without the controller. This is because launching new instances will cost more money than having a fixed number of instances available in the Cloud. This experiment has high load of requests for the system in which the controller is more likely to scale up and resides in that mood than to scale down. It should be noted that costs are computed according to Amazon EC2 price list.

Calculated total cost for each experiment is given in Table II.

Fig. 11 depicts the Average bandwidth per download. If an instance has a bandwidth of 4 Mb/s and has two current downloads running, the bandwidth per download is 2 Mb/s. As

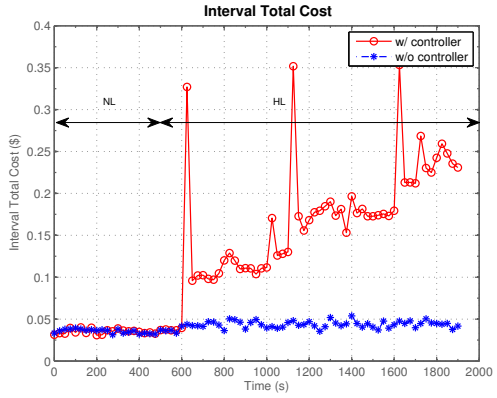

Fig. 10. SLO Experiment - Interval Total Cost

can be seen from the diagram, the experiment with controller shows significantly higher bandwidth per download. This is mainly because the instances receive less number of requests and bandwidth is divided among less requests also. This will end up having higher bandwidth available on each instance.

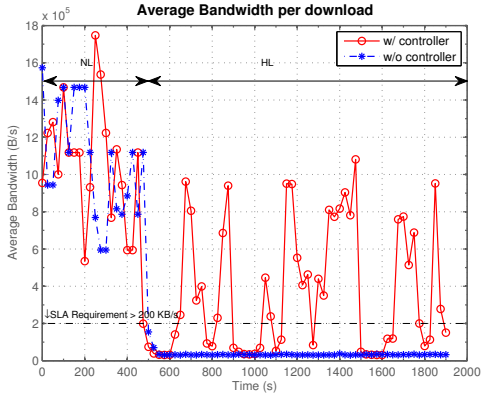

Fig. 11. SLO Experiment - Average Bandwidth per download (B/s)

Fig 12 shows the number of nodes for each experiment. As we discussed earlier the number of nodes is constant for experiment without controller. However, for the experiment with the controller the number of nodes is changed over time hence the SLO requirements can be met.

B. Cost Experiment: Decreasing Load

The purpose of this series of experiments is to show that the controller can save the total cost by releasing instances when the load is low. Each experiment in this series starts with 7 instances. The duration of the experiment is 2000 seconds.

In this series we use different workloads of two levels: high and low. In the high load the time interval between consecutive requests is selected from a uniform random distribution in the range [1, 3] seconds that corresponds to a request rate of 30 requests per minute. In the low load the time interval between consecutive requests is selected from a uniform random distribution in the range [15, 20] seconds that corresponds to a

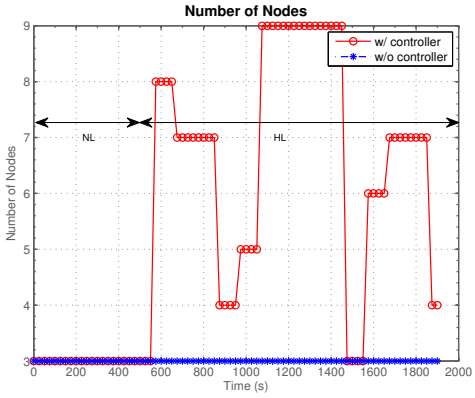

Fig. 12. SLO Experiment - Number of Nodes

	w/ controller	w/o controller
Total Cost ($)	10.509	16.5001

TABLE III
TOTAL COST FOR COST EXPERIMENT

request rate of about 3.4 requests per minute. Unlike the SLO experiment, the cost experiment starts with a high load, which changes to a low load after 500 seconds as shown in Fig. 13.

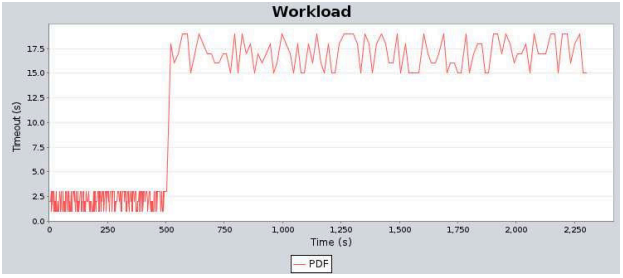

Fig. 13. Cost Experiment workload

The result of the cost experiment shown in Table III is interesting. It is observed that the total cost in the experiment with the controller is actually lower than the total cost in the experiment without the controller unlike in the SLO experiment. This is because the controller removes instances under low load and that results in cost savings. The reason that this experiment has lower cost than the previous one is that L (lower bound on number of nodes) is not equal to the initial number of nodes and it is smaller. Hence controller can scale down the number of nodes to L.

IX. RELATED WORK

There are many projects that use elements of control theory for providing automated control of computing systems including Cloud-based services [2], [7], [8], [9], [12], [15], [16], [17], [18], [19], [23]. Here we consider two related pieces of work [17], [23], which are the closest to our research aiming at automation of elasticity of storage services.

The SCADS Director proposed in [23] is a control framework that reconfigures a storage system at run time in response to workload fluctuations. Reconfiguration includes

 ISBN: 978-1-61208-187-8

adding/removing servers, redistributing and replicating data between servers. The SCADS Director employs the Model-Predictive Control technique to predict system performance for the given workload using a performance model of the system and make control decisions based on prediction. Performance modeling is performed by statistical machine learning.

Lim et al. [17] have proposed a feedback controller for elastic storage in Cloud environment. The controller consists of three components: Horizontal Scale Controller responsible for scaling the storage; Data Rebalancer Controller that controls data transfer for rebalancing after scaling up/down; and the State Machine that coordinates the actions of the controllers in order to avoid wrong control decisions caused by interference of rebalancing with applications and sensor measurements.

To our knowledge both aforementioned projects do not explicitly use cost as a controller input (state variable, system output) in the controller design. In contrast, we use state-space feedback control and explicitly include the total cost of Cloud instances as a state (system output) variable in the state-space model (when identifying the system) and as a controller input in the controller design (when determining controller gains). This allows us to use a desired value of cost in addition to the SLO requirements to automatically control the scale of the storage by trading off performance for cost.

X. Conclusion and Future Work

Elasticity in Cloud computing is an ability of a system to scale up and down (request and release resources) in response to changes in its environment and workload. Elasticity provides an opportunity to scale up under high workload and to scale down under low workload to reduce the total cost for the system while meeting SLOs. We have presented our experience in designing an elasticity controller for a key-value store in a Cloud environment and described the steps in designing it including system identification and controller design. The controller allows the system to automatically scale the amount of resources while meeting performance SLO, in order to reduce SLO violations and the total cost for the provided service. We also introduced our open source simulation framework (EStoreSim) for Cloud systems that allows to experiment with different controllers and workloads. We have conducted two series of experiments using EStoreSim. Experiments have shown the feasibility of our approach to automate elasticity control of a key-value store in a Cloud using state-space feedback control. We believe that this approach can be used to automate elasticity of other Cloud-based services.

In our future work, we will study other controller architectures such as model predictive control, and conduct experiments using real-world traces. We will also research on using feedback control for other elastic Cloud based services.

Acknowledgments

This research is supported by the End-to-End Clouds project funded by the Swedish Foundation for Strategic Research, the Complex Service Systems focus project, a part of the ICT-TNG Strategic Research Area initiative at the KTH Royal Institute of Technology, and by the Testbed for E2E Clouds RCLD-project funded by EIT ICT Labs.

References

[1] P. Horn, "Autonomic computing: IBM's perspective on the state of information technology," IBM, Tech. Rep., October 2001.

[2] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus, "Using control theory to achieve service level objectives in performance management," *Real-Time Syst.*, vol. 23, pp. 127–141, July 2002.

[3] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Wiley-IEEE Press, September 2004.

[4] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN*, vol. 17, no. 1, pp. 1–14, 1989.

[5] S. Keshav, "A control-theoretic approach to flow control," *SIGCOMM Comput. Commun. Rev.*, vol. 21, pp. 3–15, August 1991.

[6] B. Li and K. Nahrstedt, "A control-based middleware framework for quality-of-service adaptations," *Selected Areas in Communications, IEEE Journal on*, vol. 17, no. 9, pp. 1632 –1650, sep 1999.

[7] A. Kamra, V. Misra, and E. M. Nahum, "Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites," *In International Workshop on Quality of Service (IWQoS*, pp. 47–56, 2004.

[8] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: a control-theoretical approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 1, pp. 80–96, August 2002.

[9] A. Robertson, B. Wittenmark, and M. Kihl, "Analysis and design of admission control in web-server systems," in *American Control Conference. Proceedings of the 2003*, vol. 1, june 2003, pp. 254–259.

[10] T. Abdelzaher and N. Bhatti, "Web content adaptation to improve server overload behavior," in *WWW8 / Computer Networks*, 1999, pp. 1563–1577.

[11] B. Li and K. Nahrstedt, "A control theoretical model for quality of service adaptations," in *In Proceedings of Sixth International Workshop on Quality of Service*, 1998, pp. 145–153.

[12] H. D. Lee, Y. J. Nam, and C. Park, "Regulating i/o performance of shared storage with a control theoretical approach," *NASA/IEEE conference on Mass Storage Systems and Technologies (MSST)*, April 2004.

[13] S. Mascolo, "Classical control theory for congestion avoidance in high-speed internet," in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, vol. 3, 1999, pp. 2709 –2714 vol.3.

[14] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *Proceedings of the third symposium on Operating systems design and implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 145–158.

[15] M. Karlsson, C. Karamanolis, and X. Zhu, "Triage: Performance isolation and differentiation for storage systems," in *In International Workshop on Quality of Service (IWQoS)*, 2004, pp. 67–74.

[16] N. Gandhi, D. Tilbury, Y. Diao, J. Hellerstein, and S. Parekh, "Mimo control of an apache web server: modeling and controller design," in *American Control Conference, 2002. Proceedings of the 2002*, vol. 6, 2002, pp. 4922 – 4927 vol.6.

[17] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," *International Conf. on Autonomic Computing*, pp. 1–10, 2010.

[18] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *4th ACM European conf. on Computer systems*, 2009, pp. 13–26.

[19] C. Lu, T. Abdelzaber, J. Stankovic, and S. Son, "A feedback control approach for guaranteeing relative delays in web servers," in *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*, 2001, pp. 51–62.

[20] "Kompics," http://kompics.sics.se/, accessed Oct 2011.

[21] "Scala language," http://www.scala-lang.org/, accessed Oct 2011.

[22] "EStoreSim: Elastic storage simulation framework," https://github.com/amir343/ElasticStorage, accessed Oct 2011.

[23] 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.

 ISBN: 978-1-61208-187-8