ElastMan: Autonomic Elasticity Manager for Cloud-Based Key-Value Stores

Ahmad Al-Shishtawy[1,2], and Vladimir Vlassov[1]

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

[2] Swedish Institute of Computer Science, Stockholm, Sweden
ahmad@sics.se

Abstract

The increasing spread of elastic Cloud services, together with the pay-as-you-go pricing model of Cloud computing, has led to the need of an elasticity controller. The controller automatically resizes an elastic service, in response to changes in workload, in order to meet Service Level Objectives (SLOs) at a reduced cost. However, variable performance of Cloud virtual machines and nonlinearities in Cloud services, such as the diminishing reward of adding a service instance with increasing the scale, complicates the controller design. We present the design and evaluation of ElastMan, an elasticity controller for Cloud-based elastic key-value stores. ElastMan combines feedforward and feedback control. Feedforward control is used to respond to spikes in the workload by quickly resizing the service to meet SLOs at a minimal cost. Feedback control is used to correct modeling errors and to handle diurnal workload. To address nonlinearities, our design of ElastMan leverages the near-linear scalability of elastic Cloud services in order to build a scale-independent model of the service. Our design based on combining feedforward and feedback control allows to efficiently handle both diurnal and rapid changes in workload in order to meet SLOs at a minimal cost. Our evaluation shows the feasibility of our approach to automation of Cloud service elasticity.

12.1 Introduction

The growing popularity of Web 2.0 applications, such as wikis, social networks, and blogs, has posed new challenges on the underlying provisioning infrastructure. Many large-scale Web 2.0 applications leverage elastic services, such as elastic key-value stores, that can scale horizontally by adding/removing servers. Voldemort [9], Cassandra [10], and Dynamo [11] are few examples of elastic storage services.

Cloud computing [3], with its pay-as-you-go pricing model, provides an attractive environment to provision elastic services as the running cost of such services becomes proportional to the amount of resources needed to handle the current workload. The independence of peak loads for different applications enables Cloud

providers to efficiently share the resources among the applications. However, sharing the physical resources among Virtual Machines (VMs) running different applications makes it challenging to model and predict the performance of the VMs [39,40].

Managing the resources for Web 2.0 applications, in order to guarantee acceptable performance, is challenging because of the highly dynamic workload that is composed of both gradual (diurnal) and sudden (spikes) variations [41]. It is difficult to predict the workload particularly for new applications that can become popular within few days [12,13]. Furthermore, the performance requirement is usually expressed in terms of upper percentiles which is more difficult to maintain than the average performance [11, 14].

The pay-as-you-go pricing model, elasticity, and dynamic workload of Web 2.0 applications altogether call for the need for an elasticity controller that automates the provisioning of Cloud resources. The elasticity controller leverages the horizontal scalability of elastic services by provisioning more resources under high workloads in order to meet required service level objectives (SLOs). The pay-as-you-go pricing model provides an incentive for the elasticity controller to release extra resources when they are not needed once the workload decreases.

In this paper, we present the design and evaluation of ElastMan, an *Elast*icity *Man*ager for elastic key-value stores running in Cloud VMs. ElastMan addresses the challenges of the variable performance of Cloud VMs, dynamic workload, and stringent performance requirements expressed in terms of upper percentiles by combining feedforward control and feedback control. The feedforward controller monitors the current workload and uses a logistic regression model of the service to predict whether the current workload will cause the service to violate the SLOs or not, and acts accordingly. The feedforward controller is used to quickly respond to sudden large changes (spikes) in the workload. The feedback controller directly monitors the performance of the service (e.g., response time) and reacts based on the amount of deviation from the desired performance specified in the SLO. The feedback controller is used to correct errors in the model used by the feedforward controller and to handle gradual (e.g., diurnal) changes in workload.

Due to the nonlinearities in elastic Cloud services, resulting from the diminishing reward of adding a service instance (VM) with increasing the scale, we propose a scale-independent model used to design the feedback controller. This enables the feedback controller to operate at various scales of the service without the need to use techniques such as gain scheduling. To achieve this, our design leverages the near-linear scalability of elastic service. The feedback controller controls the number of nodes indirectly by controlling the average workload per server. Thus, the controller decisions become independent of the current number of instances that compose the service.

The major contributions of the paper are as follows.

- We leverage the advantages of both feedforward and feedback control to build an elasticity controller for elastic key-value stores running in Cloud environments.

- We propose a scale-independent feedback controller suitable for horizontally scaling services running at various scales.
- We describe the complete design of ElastMan including various techniques required to automate elasticity of Cloud-based services.
- We evaluate effectiveness of the core components of ElastMan using the Voldemort [9] key-value store running in a Cloud environment against both diurnal and sudden variations in workload.
- We provide an open source implementation of ElastMan with detailed instructions on how to repeat our experiments.

The rest of this paper is organized as following. Section 12.2 summarizes key concepts necessary for the paper. In Section 12.3 we describe the basic architecture of the target system we are trying to control. We continue by describing the design of ElastMan in Section 12.4. This is followed by the evaluation in Section 12.5. Related work is discussed in Section 12.6. We discus future work in Section 12.7 followed by conclusions in Section 12.8.

12.2 Background

In this section we lay out the necessary background for the paper. This include Web 2.0 applications, Cloud computing, elastic services, feedback control, and feed-forward control.

Web 2.0 Applications

Web 2.0 applications, such as Social Networks, Wikis, and Blogs, are data-centric with frequent data access [37]. This poses new challenges on the data-layer of multi-tier application servers because the performance of the data-layer is typically governed by strict Service Level Objectives (SLOs) [14] in order to satisfy costumer expectations.

With the rapid increase of Web 2.0 users, the poor scalability of a typical data-layer with ACID [38] properties limited the scalability of Web 2.0 applications. This has led to the development of new data-stores with relaxed consistency guarantees and simpler operations such as Voldemort [9], Cassandra [10], and Dynamo [11]. These storage systems typically provide simple key-value storage with eventual consistency guarantees. The simplified data and consistency models of key-value stores enable them to efficiently scale horizontally by adding more servers and thus serve more clients.

Another problem facing Web 2.0 applications is that a certain service, feature, or topic might suddenly become popular resulting in a spike in the workload [12, 13]. The fact that storage is a stateful service complicates the problem since only a particular subset of servers host the data related to the popular item.

These challenges have led to the need for an automated approach, to manage the data-tier, that is capable of quickly and efficiently responding to changes in the workload in order to meet the required SLOs of the storage service.

Cloud Computing and Elastic Services

Cloud computing [3], with its pay-as-you-go pricing model, provides an attractive solution to host the ever-growing number of Web 2.0 applications. This is mainly because it is difficult, specially for startups, to predict the future load that is going to be imposed on the application and thus the amount of resources (e.g., servers) needed to serve that load. Another reason is the initial investment, in the form of buying the servers, that is avoided in the Cloud pay-as-you-go pricing model.

To leverage the Cloud pricing model and to efficiently handle the dynamic Web 2.0 workload, Cloud services (such as key-value stores in the data-tier of a Cloud-based multi-tier application) are designed to be elastic. An Elastic service is designed to be able to scale horizontally at runtime without disrupting the running service. An elastic service can be scaled up (e.g., by the system administrator) in the case of increasing workload by adding more resources in order to meet SLOs. In the case of decreasing load, an elastic service can be scaled down by removing extra resource and thus reducing the cost without violating the service SLOs. For stateful services, scaling is usually combined with a rebalancing step necessary to redistribute the data among the new set of servers.

Feedback versus Feedforward Control

In computing systems, a controller [21] or an autonomic manager [5] is a software component that regulates the nonfunctional properties (performance metrics) of a target system. Nonfunctional properties are properties of the system such as the response time or CPU utilization. From the controller perspective these performance metrics are the *system output*. The regulation is achieved by monitoring the target system through a monitoring interface and adapting the system's configurations, such as the number of servers, accordingly through a control interface (*control input*). Controllers can be classified into feedback or feedforward controllers depending on what is being monitored.

In feedback control, the system's output (e.g., response time) is being monitored. The controller calculates the control error by comparing the current system's output to a desired value set by the system administrators. Depending on the amount and sign of the control error, the controller changes the control input (e.g., number of servers to add or remove) in order to reduce the control error. The main advantage of feedback control is that the controller can adapt to disturbance such as changes in the behaviour of the system or its operating environment. Disadvantages include oscillation, overshoot, and possible instability if the controller is not properly designed. Due to the nonlinearity of most systems, feedback controllers are ap-

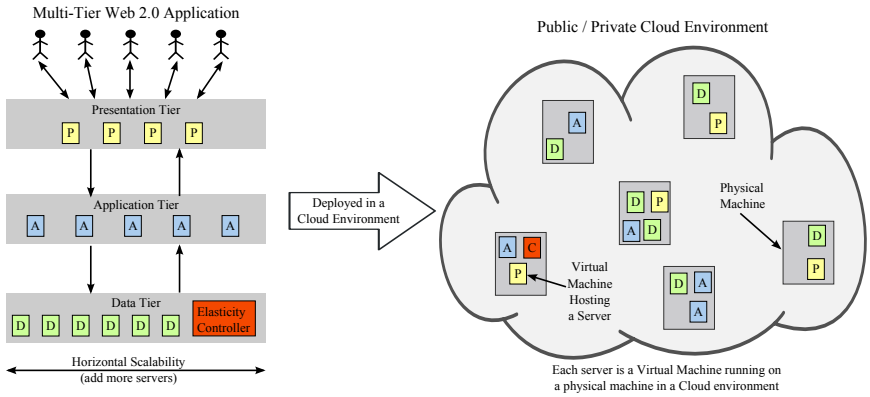

Figure 12.1: Multi-Tier Web 2.0 Application with Elasticity Controller Deployed in a Cloud Environment

proximated around linear regions called the operating region. Feedback controllers work properly only around the operating region they where designed for.

On the other hand, in feedforward control, the system's output is not being monitored. Instead the feedforward controller relies on a model of the system that is used to calculate the systems output based on the current system state. For example, given the current request rate and the number of servers, the system model is used to calculate the corresponding response time and act accordingly to meet the desired response time. The major disadvantage of feedforward control is that it is very sensitive to unexpected disturbances that are not accounted for in the system model. This usually results in a relatively complex system model compared to feedback control. The main advantages of feedforward control include being faster than feedback control and avoiding oscillations and overshoot.

12.3 Target System

We are targeting multi-tier Web 2.0 applications as depicted in the left side of Figure 12.1. We are focusing on managing the data-tier because of its major effect on the performance of Web 2.0 applications, which are mostly data centric [37]. Furthermore, the fact that storage is a stateful service makes it harder to manage as each request can be handled only by a subset of the servers that store replicas of the particular data item in the request.

For the data-tier, we assume horizontally scalable key-value stores due to their popularity in many large scale Web 2.0 applications such as Facebook and LinkedIn. A typical key-value store provides a simple put/get interface. This simplicity enables key-value stores to efficiently partition the data among multiple servers and thus to scale well to a large number of servers.

The minimum requirements to manage a key-value store using our approach (described in Section 12.4) is as follows. The store must provide a monitoring interface that enables the monitoring of both the workload and the latency of put/get operations. The store must also provide an actuation interface that enables the horizontal scalability by adding or removing service instances.

Because storage is a stateful service, actuation (adding or removing service instances) must be combined with a rebalance operation. The rebalance operation redistributes the data among the new set of servers in order to balance the load among them. Many key-value stores, such as Voldemort [9] and Cassandra [10], provide tools to rebalance the data among the service instances. In this paper, we focus on the control problem and rely on the built-in capabilities of the storage service to rebalance the load. If the storage does not provide such service, techniques such as rebalancing using fine grained workload statistics proposed by Trushkowsky et al. [14], the Aqueduct online data migration proposed by Lu et al. [42], or the data rebalance controller proposed by Lim et al. [43] can be used.

In this work we target Web 2.0 applications running in Cloud environments such as Amazon's EC2 [44] or private Clouds. The target environment is depicted on the right side of Figure 12.1. We assume that each service instance runs on its own VM; Each Physical server hosts multiple VMs. The Cloud environment hosts multiple such applications (not shown in the figure). Such environment complicates the control problem. This is mainly due to the fact that VMs compete for the shared resources. This high environmental noise makes it difficult to model and predict the performance of VMs [39, 40].

12.4 Elasticity Controller

The pay-as-you-go pricing model, elasticity, and dynamic workload of Web 2.0 applications altogether call for the need for an elasticity controller that automates the provisioning of Cloud resources depending on load. The elasticity controller leverages the horizontal scalability of elastic Cloud services by provisioning more resources under high workloads in order to meet the required SLOs. The pay-as-you-go pricing model provides an incentive for the elasticity controller to release extra resources when they are not needed once the workload starts decreasing.

In this section we describe the design of ElastMan, an elasticity controller designed to control the elasticity of key-value stores running in a Cloud environment. The objective of ElastMan is to regulate the performance of key-value stores according to a predefined SLO expressed as the 99th percentile of read operations latency over a fixed period of time.

Controlling a noisy signal, such as the 99th percentile, is challenging [14]. The high level of noise can mislead the controller into taking incorrect decisions. On the other hand, applying a smoothing filter in order to filter out noise, may also filter out a spike or, at least, delay its detection and handling. One approach to control noisy signals is to build a performance model of the system, thus avoiding

the need for measuring the noisy signal. The performance model used to predict the performance of the system given its current state (e.g., current workload). However, due to the variable performance of Cloud VMs (compared to dedicated physical servers), it is difficult to accurately model the performance of the services running in the Cloud.

To address the challenges of controlling a noisy signal and variable performance of Cloud VMs, ElastMan consists of two main components, a feedforward controller and a feedback controller. ElastMan relies on the feedforward controller to handle rapid large changes in the workload (e.g., spikes). This enables ElastMan to smooth the noisy 99th percentile signal and use feedback controller to correct errors in the feedforward system model in order to accurately bring the 99th percentile of read operations to the desired SLO value. In other words, the feedforward control is used to quickly bring the performance of the system near the desired value and then the feedback control is used to fine tune the performance.

The Feedback Controller

The first step in designing a feedback controller is to create a model of the target system (the key-value store in our case) that relates the control input to the system output (i.e., how a change in the control input affects the system output). For computing systems, a black-box approach is usually used [21], which is a statistical technique used to find the relation between the input and the output. The process of constructing a system model using the black-box approach is called *system identification*.

System identification is one of the most challenging steps in controller design. This is because a system can be modelled in many different ways and the choice of the model can dramatically affect the performance and complexity of the controller. The model of the system is usually a linear approximation of the behaviour of the system around an *operating point* (within an operating region). This makes the model valid only around the predefined point.

In order to identify a key-value store, we need to define what is the *control input* and the *system output*. In feedback control we typically monitor (measure) the system output that we want to regulate, which is, in our case, the 99th percentile of read operations latency over a fixed period of time (called R99p thereafter). The feedback controller calculates the error, which is the difference between the *setpoint*, which in our case is the required SLO value of R99p, and the measured system output as shown in equation 12.1.

$$e(t) = \text{Setpoint}_{\text{SLO\_R99p}} - \text{Measured}_{\text{R99p}}(t) \tag{12.1}$$

For the control input, an intuitive choice would be to use the number of storage servers. In other words, to try to find how changing the number of nodes affects the R99p of the key-value store. However, there are two drawback for this choice of a model. First, the model does not account for the current load on the sys-

tem. By load we mean the number of operations processed by the store per second (i.e., *throughput*). The latency is much shorter in an underloaded store than in an overloaded store. In this case, the load is treated as disturbance in the model. Controllers can be designed to reject disturbances but it might reduce the performance of the controller. Using the number of servers (which we can control) as a control input seems to be a natural choice since we can not control the load on the system as it depends on the number of clients interacting with our web application.

The second drawback is that using the number of servers as input to our model makes it nonlinear. For example, adding one server to a store having one server,doubles the capacity of the store. On the other hand, adding one server to a store with 100 servers increases the capacity by only one percent. This nonlinearity makes it difficult to design a controller because the model behaves differently depending on the size of the system. This might require having multiple controllers responsible for different operating regions corresponding to different sizes of the system. In control theory, this approach is known as gain scheduling.

In the design of the feedback controller for ElastMan, we propose to model the target store using the average throughput per server ans the control input. Although we can not control the total throughput on the system, we can indirectly control the average throughput of a server by adding/removing servers. Adding servers to the system reduces the average throughput per server, whereas removing servers reduces the average throughput per server. Our choice is motivated by the near linear scalability of elastic key-value stores (as discussed in Section 12.5). For example, when we double both the load and the number of servers, the R99p of the store remains roughly the same as well as the average throughput per server.

The major advantage of our proposed approach to model the store is that the model remains valid as we scale the store, and it does not dependent on the number of nodes. The noise in our model is the slight nonlinearity of the horizontal scalability of the elastic key-value store and the variable behaviour of the VMs in the Cloud environment. Note that this noise also exist in the previous model using the number of servers as control input.

In our proposed model, the operating point is defined as the value of the average throughput per server (input) and corresponding desired R99p (output); Whereas in the previous model, using the number of servers as a control input, the operating point is the number of servers (input) and corresponding R99p (output). Using our proposed model, the controller remains in the operating region (around operating point) as it scales the storage service. The operating region is defined around the value of the average throughput per server that produces the desired R99p regardless of the current size of the store. This eliminates the need for gain scheduling and simplifies the system identification and the controller design.

Given the current value of R99p, the controller uses the error, defined in Equation 12.1, to calculate how much the current throughput per server (called u) should be increased or decreased in order to meet the desired R99p defined in the SLO of the store. We build our controller as a classical PI controller described by Equation 12.2. The block diagram of our controller is depicted in Figure 12.2. The

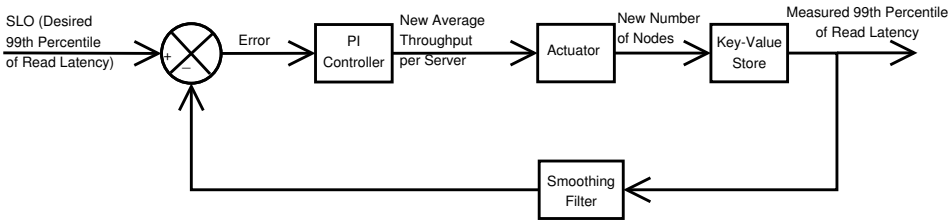

Figure 12.2: Block Diagram of the Feedback controller used in ElastMan

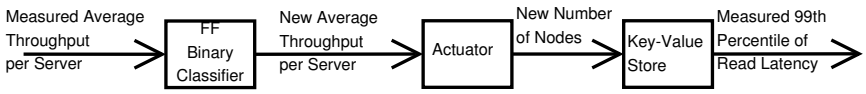

Figure 12.3: Block Diagram of the Feedforward controller used in ElastMan

controller design step involves using the system model to tune the controller gains, $\mathbf{K_p}$ and $\mathbf{K_i}$, which is out of the scope of the paper.

$$u(t+1) = u(t) + \mathbf{K_p}e(t) + \mathbf{K_i}\sum_{x=0}^{t} e(x) \tag{12.2}$$

The actuator uses the control output $u(t+1)$, which is the new average throughput per server, to calculate the new number of servers according to Equation 12.3.

$$\text{New Number of Servers} = \frac{\text{Current Total Throughput}}{\text{New Average Throughput per Server}} \tag{12.3}$$

The actuator uses the Cloud API to request/release resources and uses the elasticity API to add/remove new servers and also uses the rebalance API of the store to redistribute the data among servers.

The Feedforward Controller

ElastMan uses a feedforward model predictive controller to detect and quickly respond to spikes (rapid changes in workload). A model predictive controller uses a model of the system in order to reason about the current status of the system and make decisions. The block diagram of the feedforward controller is depicted in Figure 12.3. For our system we use a binary classifier created using logistic regression as proposed by Trushkowsky et al. [14]. The model is trained offline by varying the average intensity and the ratio of read/write operations per server as shown

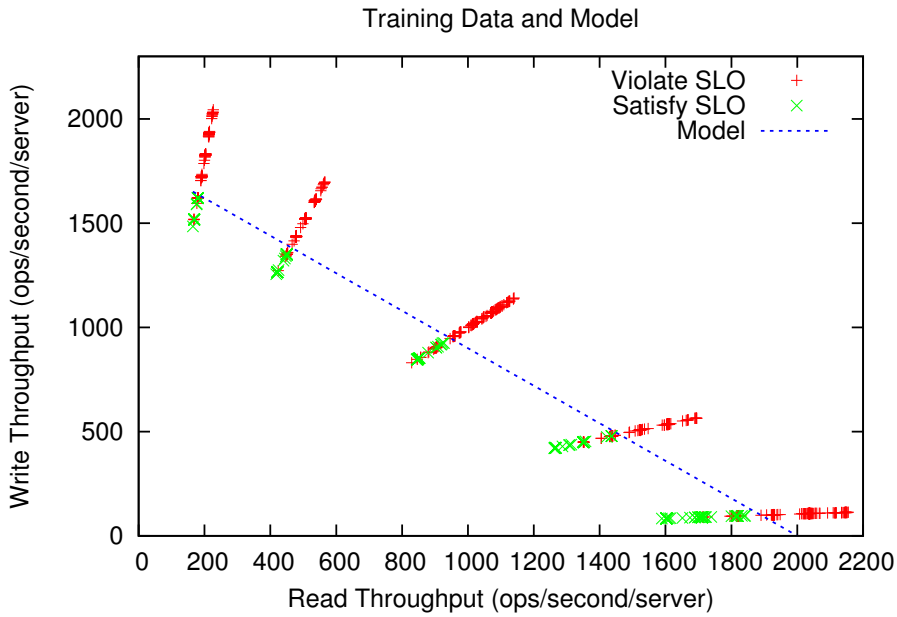

Figure 12.4: Binary Classifier for One Server

in Figure 12.4. The final model is a line that splits the plane into two regions. The region on and below the line is the region where the SLO is met whereas the region above the line is the region where the SLO is violated. Ideally, the average measured throughput should be on the line, which means that the SLO is met with the minimal number of servers.

In a very large system, averaging of throughput of servers may hide a spike that can occur on a single server or a small number of servers. In order to detect such spikes, the large system can be partitioned and each partition can be monitored separately.

The controller uses the model to reason about the current status of the system and make control decisions. If the measured throughput is far below the line, this indicates that the system is underloaded and servers (VMs) could be removed and vice versa. When a spike is detected, the feedforward controller uses the model to calculate the new average throughput per server. This is done by calculating the intersection point between the model line and the line connecting the origin with the point that corresponds to the measured throughput. The slope of the latter line is equal to the ratio of the write/read throughput of the current workload mix.

Then the calculated throughput is given to the actuator, which computes the new number of servers (using Equation 12.3) that brings the storage service close

to the desired operating point where the SLO is met with the minimal number of storage servers.

Note that the feedforward controller does not measure the R99p nor does make decisions based on error but relies on the accuracy of the model to check if the current load will cause an SLO violation. This makes the feedforward controller sensitive to noise such as changes in the behavior of the VM provided by the Cloud.

Elasticity Controller Algorithm of ElastMan

Our elasticity controller ElastMan combines the feedforward controller and feedback controller. The feedback and feedforward controllers complement each other. The feedforward controller relies on the feedback controller to correct errors in the feedforward model; whereas the feedback controller relies on the feedforward controller to quickly respond to spikes so that the noisy R99p signal that drives the feedback controller is smoothed. The flowchart of the algorithm of ElastMan is depicted in Figure 12.5.

The ElastMan elasticity controller starts by measuring the current 99th percentile of read operations latency (*R99p*) and the current average throughput (*tp*) per server. The *R99p* signal is smoothed using a smoothing filter resulting in a smoothed signal (*fR99p*). The controller then calculates the error e as in Equation 12.1. If the error is in the deadzone defined by a threshold around the desired *R99p* value, the controller takes no action. Otherwise, the controller compares the current *tp* with the value in the previous round. A significant change in the throughput (workload) indicate a spike. The elasticity controller then uses the feedforward controller to calculate the new average throughput per server needed to handle the current load. On the other hand, if the change in the workload is relatively small, the elasticity controller uses the feedback controller which calculates the new average throughput per server based on the current error. In both cases the actuator uses the current total throughput and the new average throughput per server to calculate the new number fo servers (Equation 12.3).

During the rebalance operation, which is needed in order to add or remove servers, both controllers are disabled as proposed by Lim et al. [43]. The feedback controller is disabled because the rebalance operation adds a significant amount of load on the system that causes increase in R99p. This can mislead the feedback controller causing it to wrongly add more resources. However if the storage system supports multiple rebalance instances or modifying the currently running rebalance instance, the feedforward controller can still be used. This is because the feedforward controller relies on the measured throughput of read/write operations (and it does not count rebalance operations) thus it will not be affected by the extra load added by the rebalancing operation.

Because the actuator can only add complete servers in discreet units, it will not be able to fully satisfy the controller actuation requests which are continuous values. For example, to satisfy the new average throughput per server, requested by the elasticity controller, the actuator might calculate that 1.5 servers are needed to be

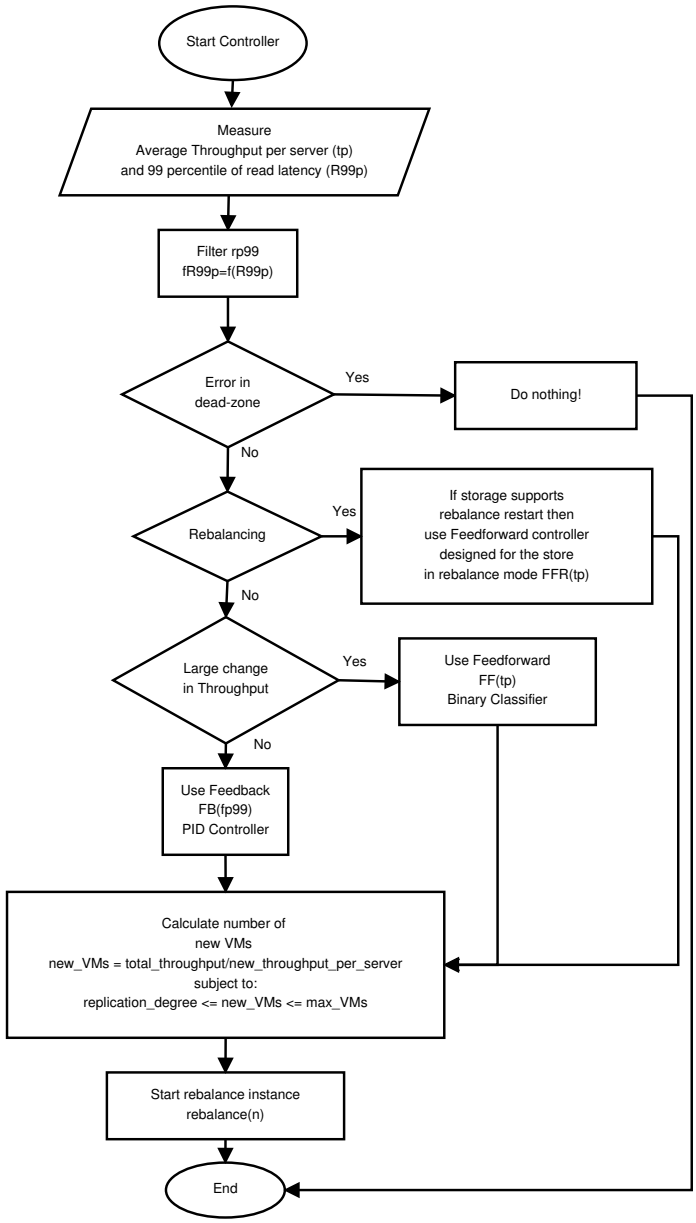

Figure 12.5: Combined Feedback and Feedforward Flow Chart

added (or removed). The actuator solves this situation by rounding the calculated value up or down to get a discrete value. This might result in oscillation, where the controller continuously adds and removes one node. Oscillations typically happen when the size of the storage cluster is small, as adding or removing a server have bigger effect on the total capacity of the storage service. Oscillations can be avoided by using the *proportional thresholding* technique as proposed by Lim et al. [43]. The basic idea is to adjust the lower threshold of the dead zone, depending on the storage cluster size, to avoid removing a server that will result in SLO violation and thus will request the server to be added back again causing oscillation.

12.5 Evaluation

We implemented ElastMan, our elasticity controller, in order to validate and evaluate the performance of our proposed solution to control an elastic key-value stores running in Cloud environments. The source code of ElastMan is publicly available[1].

Experimental Setup

In order to evaluate our ElastMan implementation, we used the Voldemort (version 0.91) Key-Value Store [9] which is used in production at many top Web 2.0 applications such as LinkedIn. We kept the core unmodified. We only extended the provided Voldemort client that is part of the Voldemort performance tool. The Voldemort performance tool is based on the YCSB benchmark [151]. Clients in our case represent the Application Tier shown in Figure 12.1. The clients connect to the ElastMan controller. Clients continuously measure the throughput and the 99th percentile of read operations latency. The controller periodically (every minute in our experiments) pulls the monitoring information from clients and then executes the control algorithm described in Section 12.4. ElastMan actuator uses the rebalance API of Voldemort to redistribute the data after adding/removing Voldemort servers.

Each client runs in its own VM in the Cloud and produces a constant workload of 3000 operations per second. The workload consists of 90% read operations and 10% read-write transactions unless otherwise stated. The total workload is increased by adding more client VMs and vice versa. This mimics the horizontal scalability of the Application Tier shown in Figure 12.1.

We run our experiments on a local cluster consisting of 11 Dell PowerEdge servers. Each server is equipped with two Intel Xeon X5660 processor (12 cores, 24 HW threads in total), and 44 GB of memory. The cluster runs Ubuntu 11.10. We setup a private Cloud using OpenStack Diablo release [92].

The Voldemort server is run in a VM with 4 cores and 6GB of memory, The Voldemort Clients run in a VM with 2 cores and 4GB of memory.

[1]The source code together with detailed instructions on how to repeat our experiments will be made publicly available after acceptance notification

Voldemort Servers	Min Client VMs	Max Client VMs	Max Cluster Total Cores	Max Cluster Total Mem (GB)	Cluster Load
9	1	12	60	102	22.7%
18	2	24	120	204	45.5%
27	3	36	180	306	68.2%
36	4	48	240	408	90.9%
45	5	60	300	510	113.6%
54	6	72	360	612	136.3%

Table 12.1: Parameters for the workload used in the scalability test.

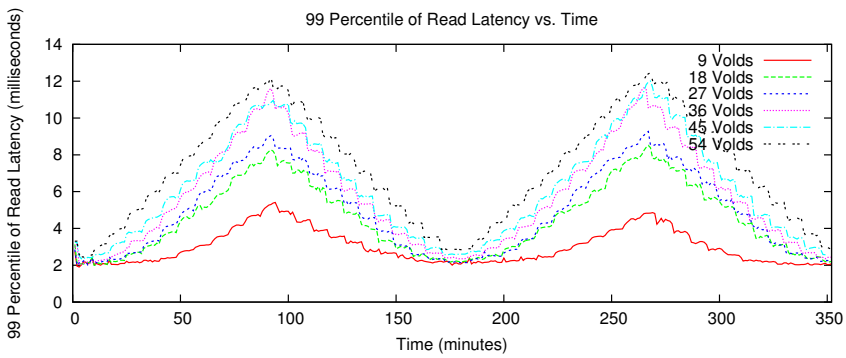

Figure 12.6: 99th percentile of read operations latency versus time under relatively similar workload

Horizontal Scalability of Key-Value stores

In this experiment we evaluated the scalability of the Voldemort key-value store in a Cloud environment. We gradually increased the cluster size and relatively scaled the workload as well. The amount of workload is described in Table 12.1.

The results, summarized in Figure 12.6 and Figure 12.7, shows the near linear scalability of Voldemort under normal load on our Cloud. That is between 45%-90% wich corresponds to 18 to 36 Voldemort servers. However, when the Cloud environment is underloaded (9 Voldemort servers at 22.7%) or when we Over-provision our Cloud environment (45 and 54 Voldemort servers at 113.6% and 136.3%), we notice a big change of the scalability of the Voldemort storage servers. This shows that the variable performance of the Cloud VMs can dramatically affect the performance and thus the ability to accurately model the system. This have motivated us to use the feedback controller in ElastMan to compensate such inaccuracies.

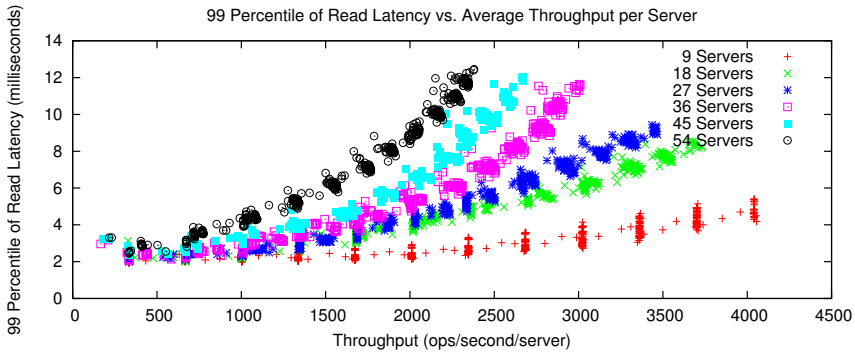

Figure 12.7: 99th percentile of read operations latency versus average throughput per server

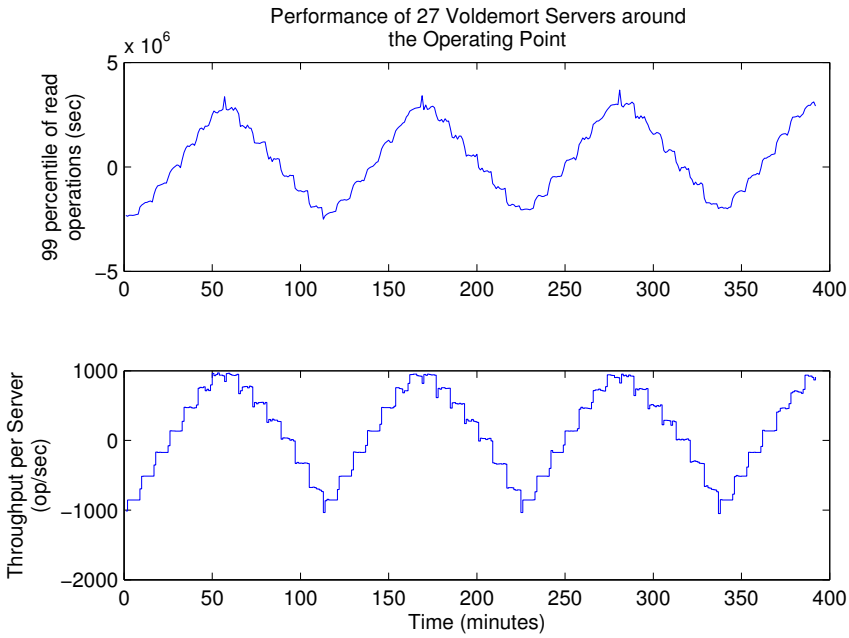

Figure 12.8: Training Data.

System Identification and Controller Design

We have used the average scenario of 27 Voldemort servers to create a model of the Voldemort storage service. The model is used to design and tune the feedback

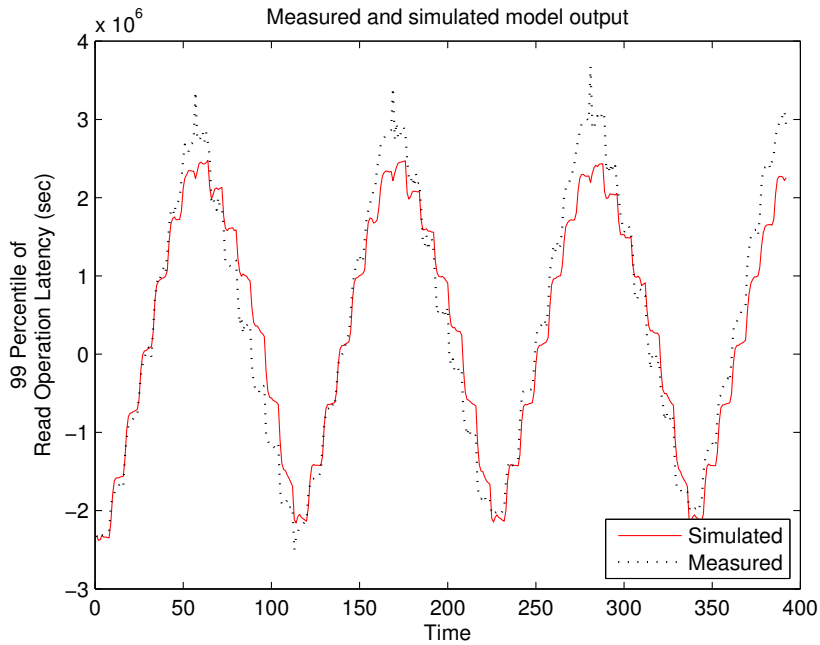

Figure 12.9: Model Performance.

controller. We used black-box system identification. The training data is shown in Figure 12.8. The performance of our model is shown in Figure 12.9 and Figure 12.10.

Varying Workload

We have tested ElastMan controller with both gradual diurnal workload and sudden increase/decrease (spikes) in workload. The goal of ElastMan controller is to keep the 99th percentile of read operation latency (R99p) at a predefined value (setpoint) as specified in the service SLO. In our experiments we choose the value to be 5 milliseconds in 1 minute period. Since it is not possible to achieve the exact specified value, we defined a 0.5 millisecond region around our setpoint with 1/3 above and 2/3 below. The controller does not react in this region which is known ad the deadzone for the controller. Note that we measure the latency from the application tier (see Figure 12.1). The overall latency observed by the clients depends on the request type that usually involves multiple read/write operations, and processing.

We start by applying gradual diurnal workload to the Voldemort cluster. The experiment starts with 9 Voldemort servers each running in its own VM. We set the maximum number of Voldemort VMs to 38. The total throughput applied on the

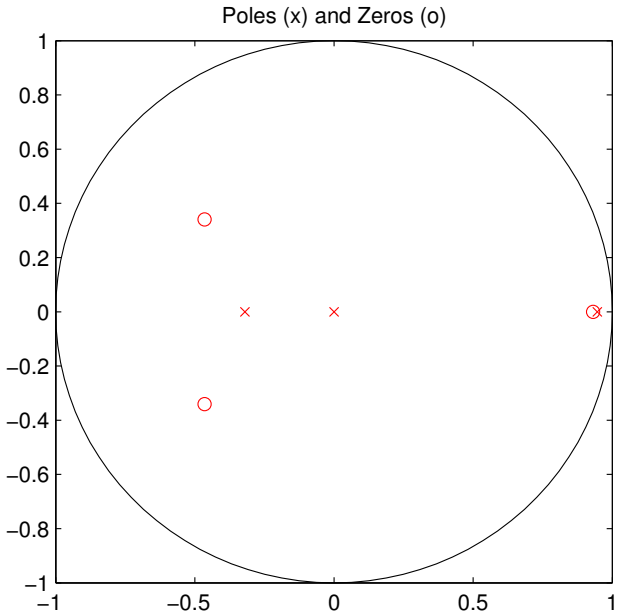

Figure 12.10: Zeros and Poles of our Model.

cluster starts with about 35000 requests per seconds and then increased to about 80000 requests per seconds. ElastMan controller is started after 30 min warm-up period. The results of our experiment is depicted in Figure 12.11. Up so far in this experiment, ElastMan relies mainly on the PI feedback controller since there are no sudden changes on the workload. ElastMan is able to keep the R99p within the desired region most of the time.

We continue the experiment by applying workload spikes with various magnitudes after 900 minutes. The results of the second part of the experiment is depicted in Figure 12.12. At the beginning of a spike, ElastMan (according to the algorithm) uses the feedforward controller since it detects large change in the workload. This is followed by using the feedback controller to fine tune the R99p at the desired value. For example, at time 924, the feedforward controller added 18 and at time 1024 added 14 nodes in order to quickly respond to the spike. Another example is at time 1336 after the spike where the feedforward controller removed 15 nodes.

Figure 12.13 depicts the performance of a Voldemort cluster with fixed number of servers (18 virtual servers).

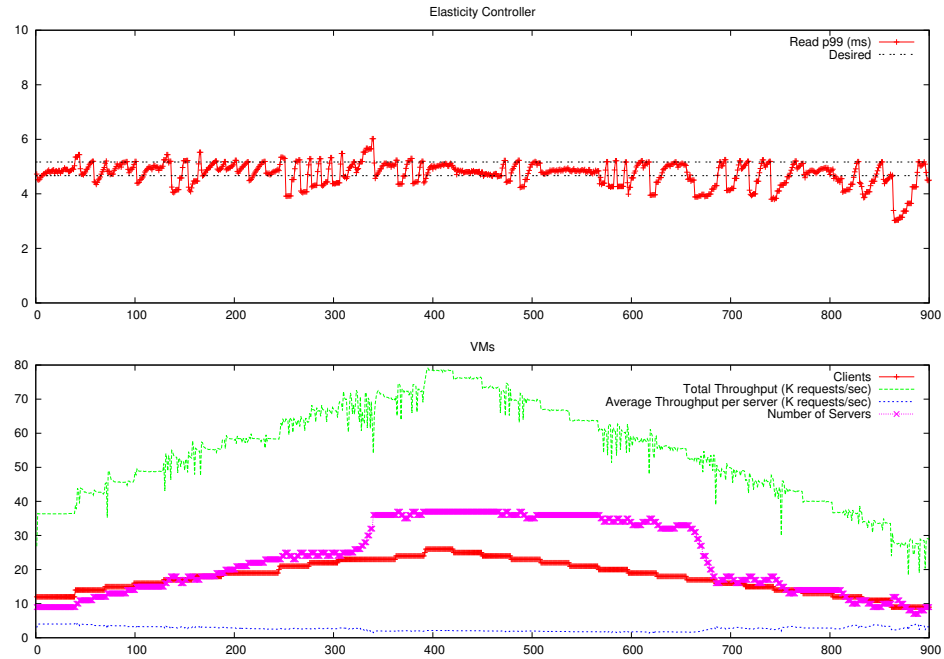

Figure 12.11: ElastMan controller performance under gradual (diurnal) workload

12.6 Related Work

There are many projects that use techniques such as control theory, machine learning, empirical modeling, or a combination of them to achieve SLOs at various levels of a multi-tier Web 2.0 application.

Lim et al. [43] proposed the use of two controllers. An integral feedback controller is used to keep the average response time at a desired level. A cost-based optimization is used to control the impact of the rebalancing operation, needed to resize the elastic storage, on the response time. The authors also propose the use of proportional thresholding, a technique necessary to avoid oscillations when dealing with discrete systems. The design of the feedback controller relies on the high correlation between CPU utilization and the average response time. Thus, the control problem is transformed into controlling the CPU utilization to indirectly control the average response time. Relying on such strong correlation might not be valid in Cloud environments with variable VM performance nor for controlling using 99th percentile instead of average. In our design, the controller uses a smoothed signal of the 99th percentile of read operations directly to avoid such problems. It is not clear how the controller proposed in [43] deals with the nonlinearity resulting

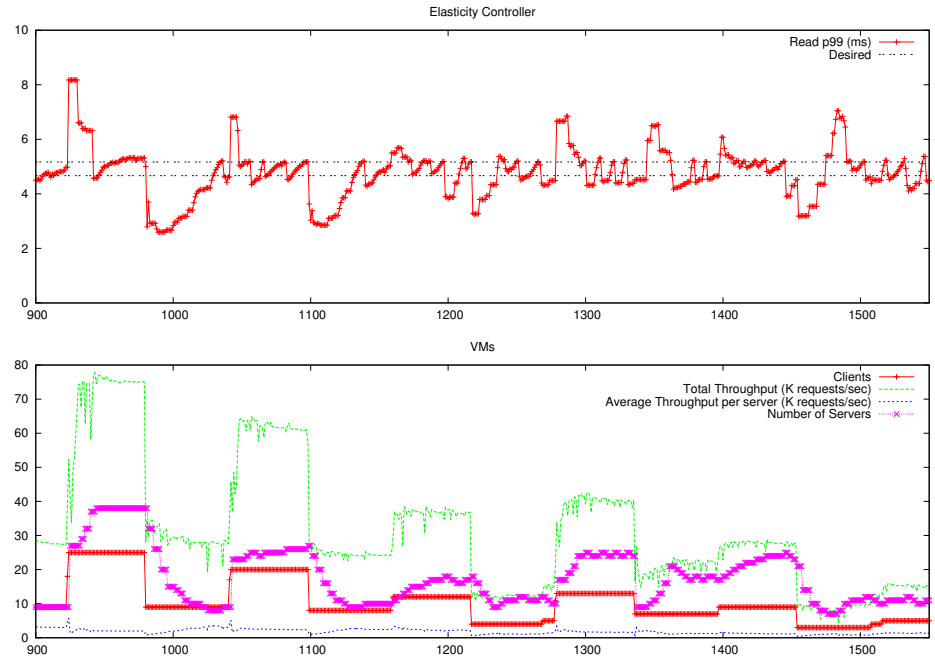

Figure 12.12: ElastMan controller performance with rapid changes (spikes) in workload

from the diminishing reward of adding a service instance with increasing the scale. Thus, it is not clear if the controller can work at different scales, a property that is needed to handle diurnal workload. In our approach we rely on the near-linear scalability of horizontally scalable stores to design a scale-independent controller that indirectly controls the number of nodes by controlling the average workload per server needed to handle the current workload. Another drawback in using only feedback controller is that it has to be switched off during rebalancing. This is because of the high disturbance resulting from the extra rebalancing overhead that can cause the feedback controller to incorrectly add more servers. We avoid switching off elasticity control during rebalancing, we use a feedforward controller tuned for rebalancing. The feedforward controller does not measure latency and thus will not be disturbed by rebalancing and can detect real increase/decrease in workload and act accordingly.

Trushkowsky et al. [14] were the first to propose a control framework for controlling upper percentiles of latency in a stateful distributed system. The authors propose the use of a feedforward model predictive controller to control the upper percentile of latency. The major motivation for using feedforward control is to

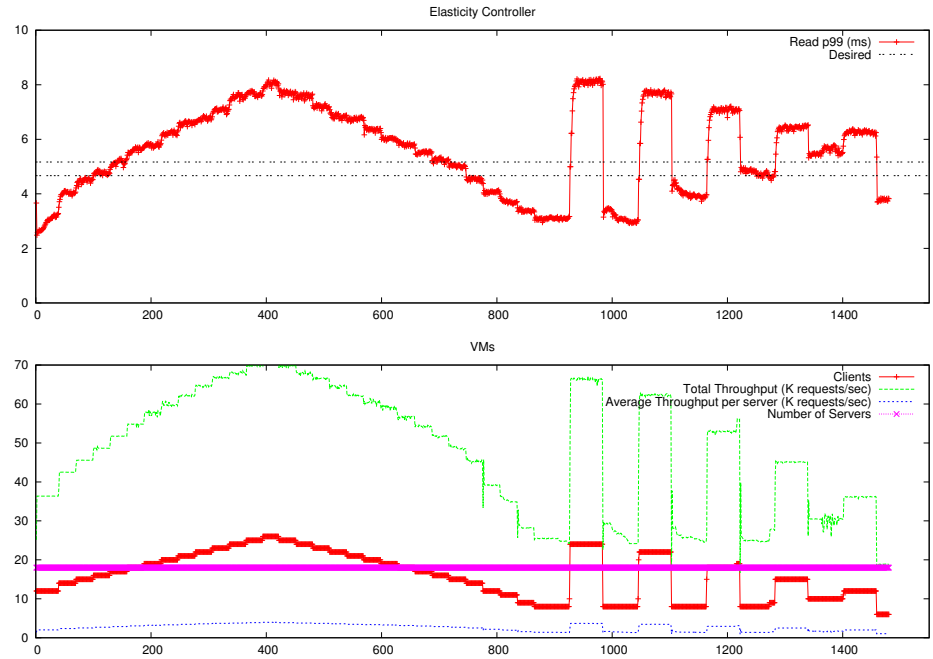

Figure 12.13: Voldemont performance with fixed number of servers (18 virtual servers)

avoid measuring the noisy upper percentile signal necessary for feedback control. Smoothing the upper percentile signal, in order to use feedback control, may filter out spikes or delay the response to them. The major drawback of using only feedforward is that it is very sensitive to noise such as the variable performance of VMs in the Cloud. The authors relies on replication to reduce the effect of variable VM performance, but in our opinion, this might not be guaranteed to work in all cases. Our approach combines both feedback and feedforward control, enabling us to leverage the advantages of both and avoid disadvantages. We rely of feedforward to quickly respond to spikes. This enables us to smooth the upper percentile signal and use feedback control to handle gradual workload and thus deal with modeling errors resulting from uncontrolled environmental noise. The authors [14] also propose the use of fine grained monitoring to reduce the amount of data transfer during rebalancing. This significantly reduces the disturbance resulting from the rebalance operation. Fine grain monitoring can be integrated with our approach to further improve the performance.

Malkowski et al. [72] focus on controlling all tiers on a multi-tier application due to the dependencies between the tiers. The authors propose the use of an empiri-

cal model of the application constructed using detailed measurements of a running application. The controller uses the model to find the best known configuration of the multi-tier application to handle the current load. If no such configuration exists, the controller falls back to another technique such as a feedback controller. Our work is different in a way that we integrate and leverage the advantages of both feedforward and feedback control. Although the empirical model will generally generate better results, it is more difficult to construct. The binary classifier proposed by Trushkowsky et al. [14] which we use together with feedback control to compensate for modeling errors is simpler to construct and might be more suitable for Cloud environments with variable VM performance. However, if needed, the empirical model can be used in our approach instead of the binary classifier. The extension of our work to control all tiers is our future work.

12.7 Future Work

A Web 2.0 application is a complex system consisting of multiple components. Controlling the entire system typically involves multiple controllers, with different management objectives, that interact directly or indirectly [47]. In our future work, we plan to investigate the controllers needed to control all tiers of a Web 2.0 application and the orchestration of the controllers in order to correctly achieve their goals.

We also plan to extend our implementation and evaluation of ElastMan to include the proportional thresholding technique as proposed by Lim et al. [43] in order to avoid possible oscillations in the feedback control. We also plan to provide the feedforward controller for the store in the rebalance mode (when performing rebalance operation). This will enable us to adapt to changes in workload that might happen during the rebalance operation.

Since ElastMan runs in the Cloud, it is necessary in real implementation to use replication in order to guarantee fault tolerance. One possible way is to use Robust Management Elements [90], that is based on replicated state machines, to replicate ElastMan and guarantee fault tolerance.

12.8 Conclusions

The strict performance requirements posed on the data-tier in a multi-tier Web 2.0 application together with the variable performance of Cloud virtual machines makes it challenging to automate the elasticity control. We presented the design and evaluation of ElastMan, an Elasticity Manager for Cloud-based key-value stores that address these challenges.

ElastMan combines and leverages the advantages of both feedback and feedforward control. The feedforward control is used to quickly respond to rapid changes in workload. This enables us to smooth the noisy signal of the 99th percentile of read operation latency and thus use feedback control. The feedback controller is

used to handle gradual (diurnal) workload and to correct errors in the feedforward control due to the noise that is caused mainly by the variable performance of Cloud VMs. The feedback controller uses a scale-independent model by indirectly controlling the number of servers (VMs) by controlling the average workload per server. This enables the controller, given the near-linear scalability of key-value stores, to operate at various scales of the store.

We have implemented and evaluated ElastMan using the Voldemort key-value store running in a Cloud environment based on OpenStack. The evaluation results show that ElastMan can handle both gradual (diurnal) workload and quickly respond to rapid changes in the workload (spikes).