ElastMan: Elasticity Manager for Elastic Key-Value Stores in the Cloud

Ahmad Al-Shishtawy
Swedish Institute of Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
ahmad@sics.se, ahmadas@kth.se

Vladimir Vlassov
KTH Royal Institute of Technology
Stockholm, Sweden
vladv@kth.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. We have implemented and evaluated ElastMan using the Voldemort key-value store running in an OpenStack Cloud environment. Our evaluation shows the feasibility and effectiveness of our approach to automation of Cloud service elasticity.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Control theory*

Keywords

Cloud Computing; Elasticity Controller; Cloud Storage; Feedback Control; Feedforward Control; SLO

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 [20], Cassandra [13], and Dynamo [9] are few examples of elastic storage services.

Cloud computing [6], 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 [21, 12].

Managing the resources for Web 2.0 applications, in order to guarantee acceptable performance, is challenging because of the dynamic workload with gradual and sudden variations [5]. It is difficult to predict the workload especially for a new application that can become popular within a few days [1, 7]. Furthermore, the performance requirement is usually expressed in terms of upper percentiles which is more difficult to maintain than the average performance [22, 9].

The pay-as-you-go pricing model, elasticity, and dynamic workload, 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 the Service Level Objectives (SLOs). The pricing model provides an incentive for the elasticity controller to release resources 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 in the Cloud. ElastMan automatically resizes an elastic service in response to changes in workload, in order to meet SLOs at a reduced cost. By combining feedforward control and feedback control, ElastMan addresses the challenges of the variable performance of Cloud VMs, dynamic workload, and stringent performance requirements expressed in terms of upper percentiles. The feedforward controller of ElastMan monitors the workload and uses a logistic regression model of the service to predict whether the workload will cause the service to violate the SLOs, and acts accordingly. The feedforward controller is used to quickly respond to sudden large changes (spikes) in the workload. The feedback controller monitors the service performance 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 services, resulting from the diminishing reward of adding a service instance as a result of 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 the elastic service. The feedback controller controls the number of servers indirectly by controlling the average workload per server. Thus, the controller decisions become independent of the number of service instances.

The major contributions of the paper are as follows. First, we leverage the advantages of both feedforward and feedback control to build an elasticity controller for elastic key-value stores running in the Cloud. Second, we propose a scale-independent feedback controller suitable for horizontally scaling services of various scales. Third, we describe the complete design of ElastMan including various techniques required to automate elasticity of Cloud-based services. Finally, we evaluate effectiveness of ElastMan using the Voldemort [20] key-value store running in a Cloud environment against both diurnal and sudden variations in workload.

The rest of this paper is organized as follows. Section 2 gives background necessary for the paper. In Section 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 4. Evaluation results are presented in Section 5. Related work is discussed in Section 6. We discuss future work in Section 7 and conclusions in Section 8.

2. BACKGROUND

This section gives the necessary background for the paper. This includes Web 2.0 applications, Cloud computing, elastic services, feedback control, and feedforward control.

2.1 Web 2.0 Applications

Web 2.0 applications, such as Social Networks, Wikis, and Blogs, are data-centric with frequent data access [18]. This poses new challenges on the data-tier of multi-tier applications because the performance of the data-tier is typically governed by strict SLOs [22]. With the rapid increase of the number of users, the poor scalability of a typical data-tier with ACID [19] properties limited the scalability of Web 2.0 applications. This has led to the development of NoSQL databases with relaxed consistency guarantees and simpler operations in order to achieve horizontal scalability and high availability. Examples of NoSQL data-stores include, among others, key-value stores such as Voldemort [20] and Dynamo [9], and wide column stores such as Cassandra [13]. In this work, we focus on key-value stores, which typically provide simple key-value pair 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 service or topic might quickly become popular resulting in a spike in the workload [1, 7]. The fact that storage is a stateful service complicates the problem since only a subset of servers host data of the popular item. For stateful services, scaling is usually combined with rebalancing necessary to redistribute the data among the new set of servers.

2.2 Cloud Computing and Elastic Services

Cloud computing [6], 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, especially for startups, to predict the future load that might be imposed on the application and thus to predict the amount of resources needed to serve that load. Another reason is the initial investment, in the form of buying the servers, is avoided in the Cloud pricing model.

To leverage the Cloud pricing model and to efficiently handle the dynamic workload, Cloud services are designed to be elastic. An elastic service is able to scale horizontally at runtime without disrupting the service. An elastic service can be scaled up in the case of increasing workload by adding resources in order to meet SLOs. In the case of decreasing load, the service can be scaled down by removing resources and thus reducing the cost without violating the SLOs.

2.3 Feedback versus Feedforward Control

In computing systems, a controller[10] or an autonomic manager[11] 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 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 approximated around linear regions called the operating region. Feedback controllers work properly only in the operating region they where designed for.

In feedforward control, the system's output is not monitored. Instead the feedforward controller relies on a model of the system that is used to calculate the system's 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 issue of feedforward control is that it is sensitive to unexpected disturbances that are not accounted for in the system model. Addressing this issue results in a relatively complex system model compared to feedback control. The advantages of feedforward control include being faster than feedback control and avoiding oscillations and overshoot.

3. TARGET SYSTEM

We are targeting multi-tier Web 2.0 applications (the left side of Fig. 1). We are focusing on managing the data-tier

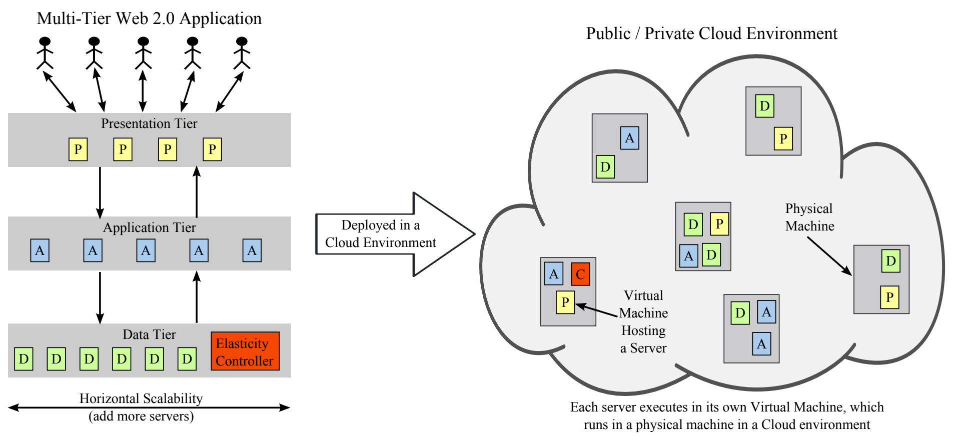
Multi-Tier Web 2.0 Application
Presentation Tier
P
Application Tier
A
Data Tier
D
Elasticity Controller
Horizontal Scalability (add more servers)
Deployed in a Cloud Environment
Public / Private Cloud Environment
Physical Machine
Virtual Machine Hosting a Server
C
Each server executes in its own Virtual Machine, which runs in a physical machine in a Cloud environment

Figure 1: Multi-Tier Web 2.0 Application with Elasticity Controller Deployed in a Cloud Environment

because of its major effect on the performance of Web 2.0 applications, which are mostly data centric [18]. 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 efficient partitioning of the data among multiple servers and thus to scale well to a large number of servers.

The minimum requirements to enable elasticity control of a key-value store using our approach (described in Section 4) are as follows. The store must provide a monitoring interface to monitor the workload and the latency of put/get operations. The store must also provide an actuation interface that allows horizontal scalability by adding or removing servers. As storage is a stateful service, actuation must be combined with a rebalance operation, which redistributes the data among the new set of servers. Many stores, such as Voldemort [20] and Cassandra [13], provide rebalancing tools. 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 have no such capabilities, existing techniques, e.g., [22, 15, 14], can be used.

We target applications running in the Cloud (right side of Fig. 1). We assume that each service instance runs on its own VM; each physical machine hosts multiple VMs. The Cloud environment hosts multiple applications (not shown in the figure). Such environment complicates the control problem mainly due to the fact that VMs compete for the shared resources. This environmental noise makes it difficult to model and predict the performance of VMs [21, 12].

4. ELASTICITY CONTROLLER

In this section we describe the design of ElastMan, an elasticity controller that allows to automate the elasticity of key-value stores running in the Cloud. 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 [22]. The high level of noise can cause the controller to make wrong decisions. Applying a smoothing filter in order to filter out noise, may also filter out a spike or delay its detection. One approach to control noisy signals is to build a performance model of the system, thus avoiding the need to measure the noisy signal [22]. The model is 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 machines), 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 VMs, ElastMan consists of two 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.

4.1 The Feedback Controller

The first step in designing a feedback controller is to build 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). A black-box approach is usually used for computing systems [10] that is a statistical technique used to find the relation between the input and the output. The process of building the model is called *system identification*.

System identification is a challenging step in controller design because a system can be modelled in different ways. The choice of the model can dramatically affect the performance and complexity of the controller. The model 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, (i.e., to build a model of the store) we need to define the *control input* and the *system output*. In feedback control we typically monitor the system output to be regulated, 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

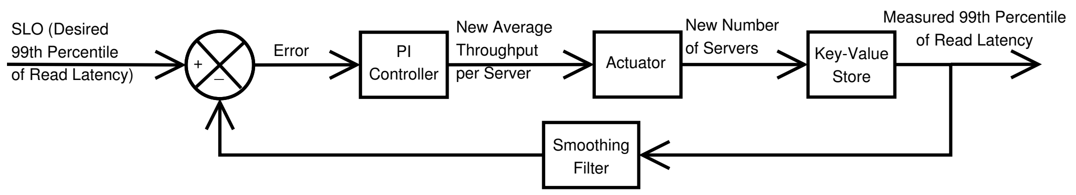
SLO (Desired 99th Percentile of Read Latency)
+
–
Error
PI Controller
New Average Throughput per Server
Actuator
New Number of Servers
Key-Value Store
Measured 99th Percentile of Read Latency
Smoothing Filter

Figure 2: ElastMan Feedback Control

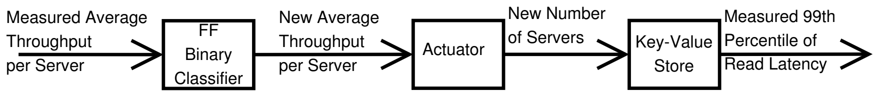
Measured Average Throughput per Server
FF Binary Classifier
New Average Throughput per Server
Actuator
New Number of Servers
Key-Value Store
Measured 99th Percentile of Read Latency

Figure 3: ElastMan Feedforward Control

the *setpoint*, which in our case is the SLO value of R99p, and the measured system output as shown in equation 1.

$$e(t) = \text{Setpoint}_{\text{SLO\_R99p}} - \text{Measured}_{\text{R99p}}(t) \qquad (1)$$

For the control input, an intuitive choice would be to use the number of storage servers. In this case, to build a model (identify the system) means to find how changing the number of servers affects the R99p of the key-value store. However, there are two drawbacks of using the number of servers as a control input in the model. First, with this control input, the model does not account for the current load on the system. 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 end-users of the web application.

The second drawback of using the number of servers as a control input is that the model becomes nonlinear. That complicates the controller design. For example, adding one server to a store having one server doubles the capacity of the store; whereas 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 system size. This might require multiple controllers responsible for different operating regions corresponding to different system sizes. 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 as the control input. Although we cannot control the total throughput on the system, we can indirectly control the average throughput of a server by adding/removing servers. Adding servers reduces the average throughput per server under the same load, whereas removing servers increases the average throughput per server. Our idea to use the average throughput per server as the control input is motivated by the near linear scalability of elastic key-value stores (as discussed in Section 5.2).

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 depend on the number of servers. 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 Cloud VMs. Note that this noise also exists in the model using the number of servers as control input.

In our 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 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 2. The block diagram of our controller is depicted in Fig. 2. The 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.

$$u(t+1) = u(t) + \mathbf{K_p} e(t) + \mathbf{K_i} \sum_{x=0}^{t} e(x) \qquad (2)$$

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 3.

$$\text{New Number of Servers} = \frac{\text{Current Total Throughput}}{\text{New Average Throughput per Server}} \qquad (3)$$

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

4.2 The Feedforward Controller

In order to detect and quickly respond to spikes in workload, ElastMan employs a feedforward model predictive controller that uses a model of the system to reason about the

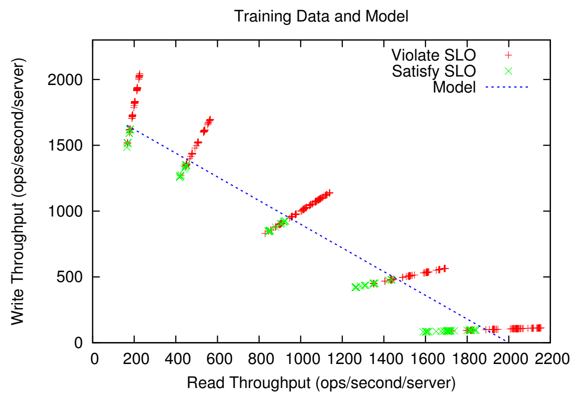

Figure 4: Binary Classifier for One Server

status of the system and make decisions. The block diagram of feedforward control is shown in Fig. 3. For the model we use a binary classifier built using logistic regression as proposed by Trushkowsky et al. [22]. The model is trained offline by varying the average intensity and the ratio of read/write operations per server as shown in Fig. 4. The final model is a line that splits the plane into two regions. In the region on and below the line, the SLO is met. In the region above the line, 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. This is caused by a possible presence of data skews, e.g., where the workload is dominated by a few popular keys. In order to deal with such bursty load toward a single (or a few) server, and to detect such spikes, the large system can be partitioned and each partition can be monitored separately. Another approach to account for skews in the workload would be to use the throughput variance in addition to the average value. Considering handling of skews in workload is a subject for our future work.

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 (and VMs where the servers run) 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.

The calculated throughput is given to the actuator, which computes the new number of servers (Equation 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 it sensitive to noise such as changes in the behavior of the Cloud VMs.

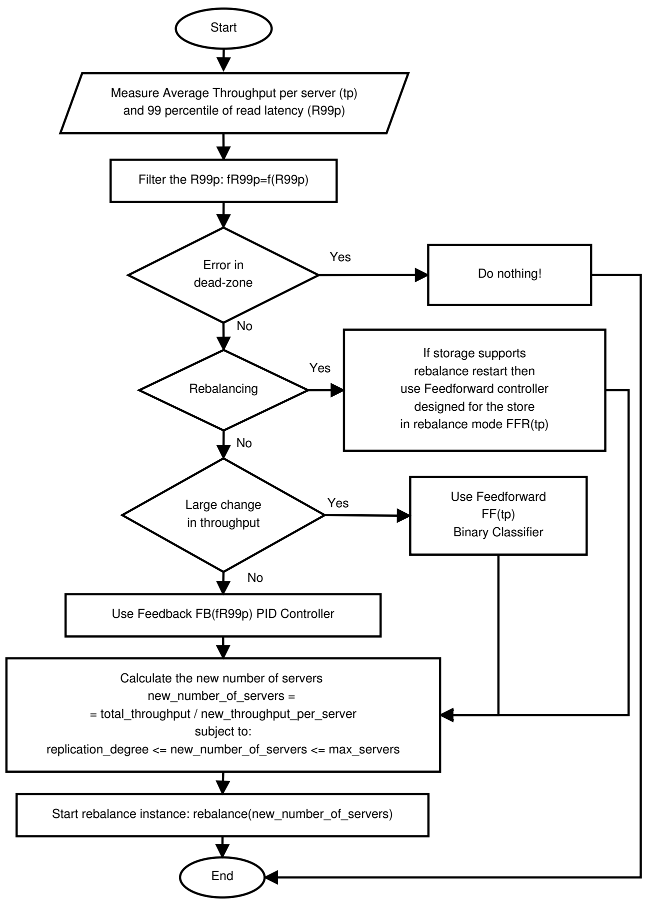
Start
Measure Average Throughput per server (tp)
and 99 percentile of read latency (R99p)
Filter the R99p: fR99p=f(R99p)
Error in
dead-zone
Yes
Do nothing!
No
Rebalancing
Yes
If storage supports
rebalance restart then
use Feedforward controller
designed for the store
in rebalance mode FFR(tp)
No
Large change
in throughput
Yes
Use Feedforward
FF(tp)
Binary Classifier
No
Use Feedback FB(fR99p) PID Controller
Calculate the new number of servers
new_number_of_servers =
= total_throughput / new_throughput_per_server
subject to:
replication_degree <= new_number_of_servers <= max_servers
Start rebalance instance: rebalance(new_number_of_servers)
End

Figure 5: ElastMan Flow Chart

4.3 Elasticity Control Algorithm of ElastMan

ElastMan combines the feedforward and feedback controllers, which complement each other. The feedforward controller relies on the feedback controller to correct errors in the feedforward model. The feedback controller relies on the feedforward controller to quickly respond to spikes so that the noisy R99p signal that drives the feedback controller can be smoothed. The flowchart of ElastMan is shown in Fig. 5.

ElastMan starts by measuring the 99th percentile of read latency (*R99p*) and the 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 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) indicates 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 of servers (Equation 3).

During the rebalance operation performed when adding or removing servers, both controllers are disabled as proposed by Lim et al. [14]. The feedback controller is disabled because the rebalance operation adds a significant amount of load on the system that causes an increase in R99p. This can mislead the feedback controller causing it to wrongly add more servers. However if the system supports multiple rebalance instances or modifying the 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 added (or removed). The actuator solves this situation by rounding the calculated value to get a discrete value. This might result in oscillation, where the controller continuously adds and removes one server. 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. [14]. The basic idea is to adjust the lower threshold of the deadzone, 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.

5. EVALUATION

We have implemented ElastMan[1] in order to evaluate our proposed approach to automation of Cloud service elasticity.

5.1 Experimental Setup

In order to evaluate ElastMan, we have chosen the Voldemort (version 0.91) Key-Value Store [20] which is used in production in many applications such as LinkedIn. We kept the core unmodified. We only extended the Voldemort client that is a part of the Voldemort performance tool, which is based on the YCSB benchmark [8]. Voldemort clients, representing the application tier shown in Fig. 1, continuously issue requests to the store and measure the throughput and the 99th percentile of read latency (playing the role of sensors). The controller periodically (every minute in our experiments) pulls the monitoring data from clients and then executes the control algorithm described in Section 4. The ElastMan actuator uses the Voldemort rebalance tool to redistribute data when adding/removing Voldemort servers.

In our evaluation experiments, the distribution of the keys in operations issued by clients is uniform. We use the rebalancing tool to keep the distribution of the keys among the servers close to uniform in order to balance the load among the servers. We run our experiments on a cluster of 11 nodes each with two Intel Xeon X5660 processors (24 HW threads), and 44 GB of memory. The cluster runs Ubuntu 11.10. We setup a private Cloud using OpenStack Diablo release [2]. Each client runs in its own VM and generates a workload of 3000 op/sec that consists of 90% read operations and 10% read-write transactions. The workload is increased by adding client VMs and decreased by removing client VMs. This mimics the horizontal scalability of the Application Tier shown in Fig. 1. Each Voldemort server runs in a VM with 4 cores and 6GB of memory. Each Voldemort Client runs in a VM with 2 cores and 4GB of memory.

The evaluation was done on a private Cloud isolated from unpredictable effects present in public Clouds; nevertheless, variations in VM performance caused by changes in scale (the increasing number of VMs competing for a fixed amount of physical resources) are present in evaluation experiments, as described in Section 5.2 and shown in Fig. 6 and Fig. 7.

[1]`https://elastman.svn.ict.kth.se/elastman/public/ElastMan/`

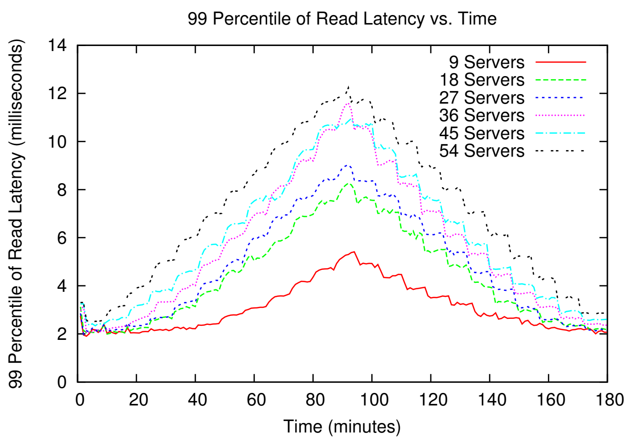

Figure 6: 99th percentile of read operations latency versus time under relatively similar workload for different number of Voldemort servers

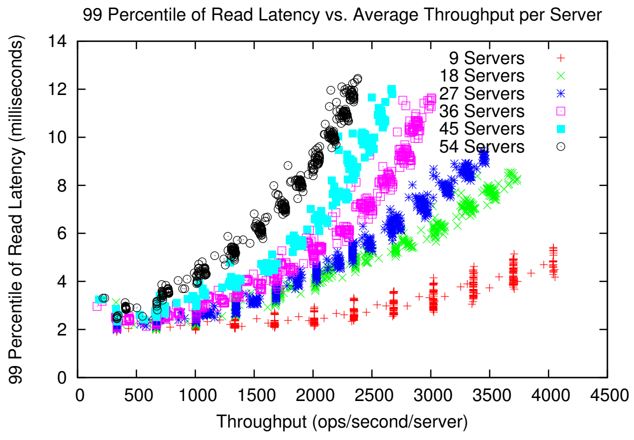

Figure 7: 99th percentile of read operations latency versus average throughput per server for different number of Voldemort servers

5.2 Horizontal Scalability of Key-Value Stores

In this experiment series, we have evaluated the scalability of the Voldemort store in a Cloud environment. We gradually increase the cluster size and relatively scale the workload. The parameters for the used resources and the

Input parameters		Calculated parameters				
Number of Voldemort servers	Number of clients	Total target throughput (K ops/sec)	Target throughput per server (K ops/sec)	Total max memory used by clients and servers (GB)	Total max number of virtual cores used by clients and servers	Cluster load
n	$[m_{min} .. m_{max}]$	$[3m_{min} .. 3m_{max}]$	$[\frac{3m_{min}}{n} .. \frac{3m_{max}}{n}]$	$6n + 4m_{max}$	$c = 4n + 2m_{max}$	$c/264$
9	[1 .. 12]	[3 .. 36]	[0.3 .. 4]	102	60	22.7%
18	[2 .. 24]	[6 .. 72]	[0.3 .. 4]	204	120	45.5%
27	[3 .. 36]	[9 .. 108]	[0.3 .. 4]	306	180	68.2%
36	[4 .. 48]	[12 .. 144]	[0.3 .. 4]	408	240	90.9%
45	[5 .. 60]	[15 .. 180]	[0.3 .. 4]	510	300	113.6%
54	[6 .. 72]	[18 .. 216]	[0.3 .. 4]	612	360	136.3%

Table 1: Parameters for resources used in the scalability test. The cluster load is computed as a ratio of the number of virtual cores used by clients and servers to the total number of physical cores (264 in our case)

cluster load are shown in Table 1. The cluster load shows how the physical resources of the cluster are loaded with virtual resources (virtual cores). The cluster load is computed as a ratio of the number of virtual cores used by clients and servers to the total number of physical cores in the cluster (264 in our case). The value of load above 100% indicates that there are more virtual cores than physical cores.

In each experiment of this series, the number of Voldemort servers is fixed (9, 18, 27, . . .) whereas the workload (the number of clients) is repeatedly increasing and decreasing over time in the corresponding range from the minimum number to the maximum number of clients according to Table 1. The parameters are chosen in such a way that target throughput per server is expected to be in the same range ([0.3 .. 4] K ops/sec) for different configurations of the second tier (clients) and the third tier (Voldemort servers) as calculated in Table 1. Our assumption about the near linear horizontal scalability of key-value stores is that the more servers are used in a configuration, the proportionally more clients can be served with about the same performance measured as the 99th percentile of read latency and throughput per server. However as clients and servers (virtual cores) share physical resources (physical cores), which can be also loaded with other applications, the above assumption is unrealistic in some cases as illustrated in our experiments (Fig. 6 and Fig. 7). The performance of a store is mostly affected by two factors: the load on the cluster caused by running VMs (contention for physical resources) and the load on the store caused by clients.

Fig. 6 depicts the 99th percentile of read latency (R99p) versus time for the different configurations specified in Table 1. Changes in the workload (the number of clients) over time cause changes in the R99p. However, these changes are different for different configurations. The R99p is about the same when both the cluster load (resource contention) and workload (the number of clients) are minimal (e.g., at time about 180 and 350 minutes). At time about 90 minutes, the difference in R99p for different configuration is rather high. The R99p for the high contended configuration (54 servers plus 72 clients) is rather high (12 ms) compared to the R99p of the uncontended configuration (9 servers plus 12 clients).

Fig. 7 depicts the R99p versus the average throughput per server for the different fixed number of servers. The throughput and the R99p are measured in 1 min intervals.

The results, depicted in Fig. 6 and Fig. 7, show the near linear scalability of Voldemort around an operating point under the normal cluster load. In this series of experiments, the operating point (desired latency) is set to 5 ms for the 99th percentile of read latency at a throughput of 2000 ops/sec per server. The normal cluster load is in the range 45%-90% (that corresponds to 18-36 servers) of the cluster's physical capacity. However, in the case when the cluster is underloaded at 22.7% (9 servers) or overloaded at 113.6%-136.3% (45-54 servers), we notice a big change in 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.

5.3 Controller Design

In our experiments, we specify the 99th percentile of read latency of 5 ms in 1 min period as the target performance (SLO). In order to build models of the system for feedforward and feedback control, we choose a normally loaded configuration of Voldemort with 27 servers that corresponds to 68.2% of the cluster load (Table 1). According to Fig. 7, the operating point for this configuration is around 2000 ops/sec per server for the specified SLO of 5 ms. In order to design a feedback controller, we have performed the black-box system identification and determined controller gains [10, 17].

5.4 Varying Workload

We have tested ElastMan with both gradual diurnal workload and sudden changes (spikes) in workload. The goal of ElastMan is to keep the 99th percentile of the 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 ms in 1 min period. Since it is not possible to achieve the exact specified value, we defined a 0.5 ms region around our setpoint with 1/3 above and 2/3 below. The controller does not react in this region which is known as the deadzone.

We start by applying a gradual diurnal workload to the Voldemort cluster (Fig. 8). The experiment starts with 9 Voldemort servers each running in its own VM. We set the maximum number of Voldemort servers to 38. The total measured throughput starts at about 35000 ops/sec and then increases to about 80000 ops/sec by adding more clients. ElastMan controller is started after 30 min warm-up period. So far in this experiment, ElastMan relies mainly on the PI feedback controller since there are no sudden changes on the workload. As results show (Fig. 8), ElastMan is able to keep the R99p within the desired region most of the time.

After 900 min (Fig. 8), we apply workload spikes with various magnitudes. At the beginning of a spike, ElastMan uses the feedforward control since it detects large change in

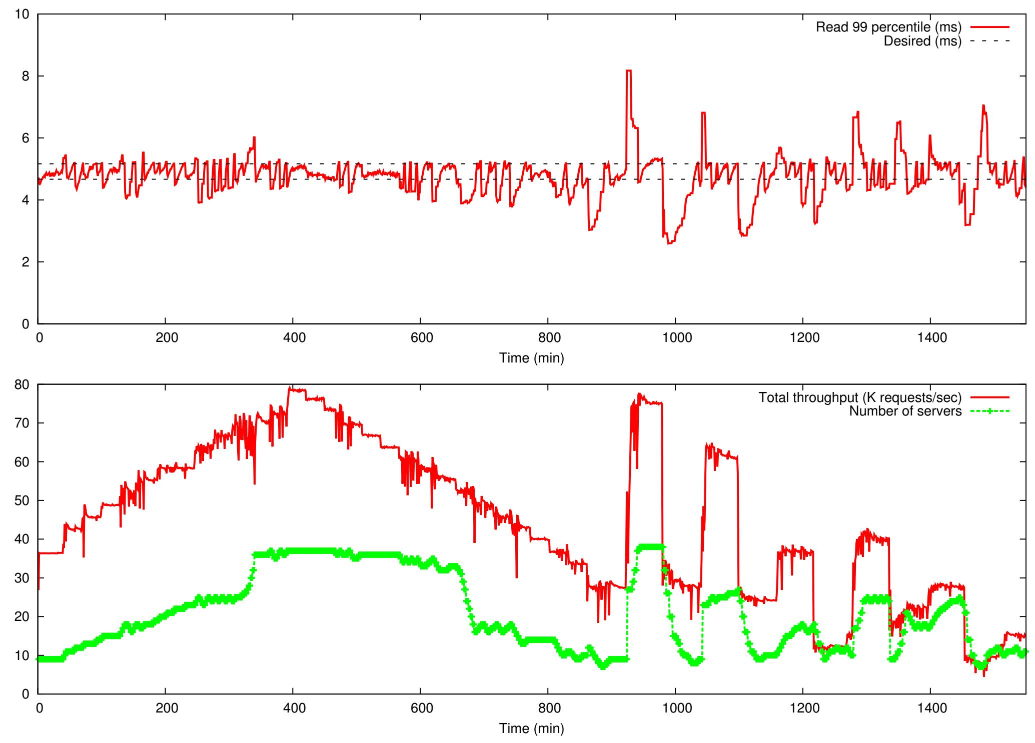

Figure 8: Performance of Voldemort with ElastMan under gradual diurnal workload (0-900 min) and under workload with spikes (900-1500 min)

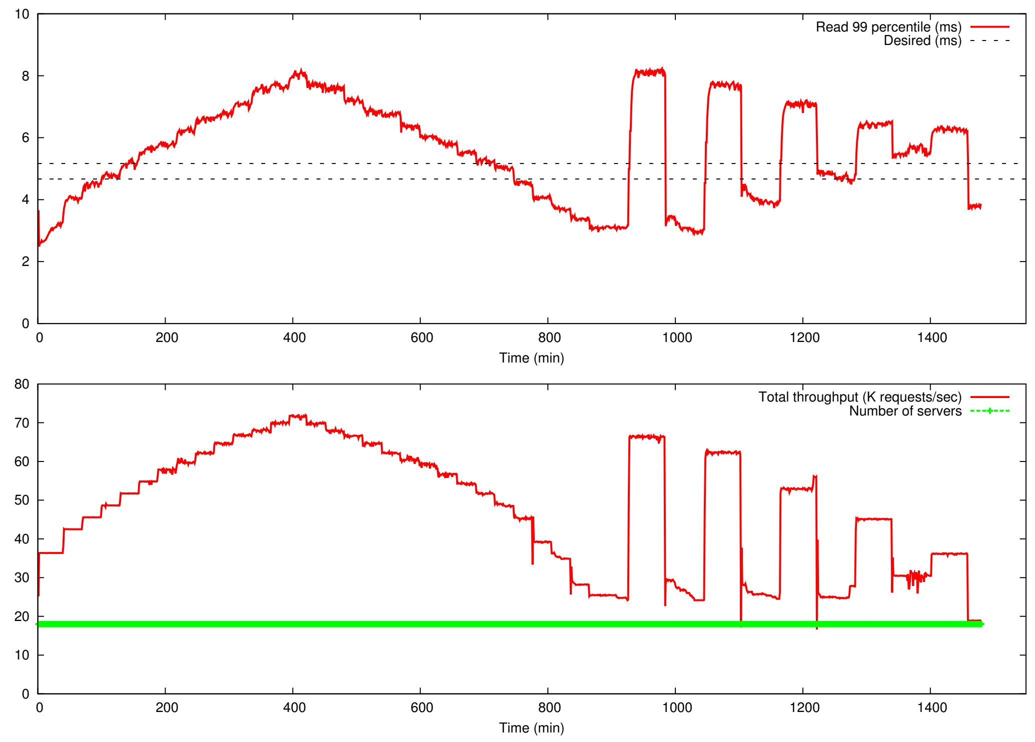

Figure 9: Performance of Voldemont without ElastMan with fixed number of servers (18 servers) under gradual diurnal workload (0-900 min) and under workload with spikes (900-1500 min)

the workload. Then it uses the feedback controller to fine tune the R99p at the desired value. For example, at time 924 min, the feedforward controller adds 18 servers whereas at time 1024 min it adds 14 servers in order to quickly respond to workload spikes. In another example, at time 1336 min, the feedforward controller removes 15 servers after a spike.

Fig. 9 depicts the performance of Voldemort without ElastMan, i.e., with a fixed number of servers under the same workload as in the previous experiment with ElastMan. Results of this experiment show that Voldemort without ElastMan can not meet required SLO most of the time as the 99th percentile read latency is strongly correlated with the workload: the higher workload is the higher is the R99p.

6. RELATED WORK

There are many projects that use different 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. [14] 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 [14] deals with the nonlinearity resulting 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 servers 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 changes in workload and act accordingly.

Trushkowsky et al. [22] 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 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 Cloud VMs. The authors rely 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 on feedforward to quickly respond to spikes. This enables us to smooth the upper percentile signal and use feedback control to handle gradual workload and deal with modeling errors resulting from uncontrolled environmental noise. The authors [22] also propose the use of fine grained monitoring to reduce the amount of data transfer during rebalancing. This significantly reduces the disturbance caused by the rebalance operation. Fine grain monitoring can be integrated with our approach to further improve the performance.

Malkowski et al. [16] focus on controlling all tiers on a multi-tier application due to the dependencies between the tiers. The authors propose the use of an empirical 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. [22] 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, the empirical model can be used in our approach instead of the binary classifier.

7. FUTURE WORK

A Web 2.0 application is a complex system consisting of multiple components. Controlling the system typically involves multiple controllers, with different management objectives, that interact directly or indirectly [4]. 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 plan to extend our implementation of ElastMan to include the proportional thresholding technique [14] in order to avoid possible oscillations in feedback control. We also plan to provide the feedforward controller that operates when the store is in rebalance mode. This will enable the store to adapt to changes in workload during rebalancing.

Replication can be used to guarantee fault tolerance of ElastMan. One possible way is to use Robust Management Elements [3] based on replicated state machines, to replicate ElastMan and guarantee fault tolerance.

8. CONCLUSIONS

The strict performance requirements posed on the datatier in a multi-tier Web 2.0 application together with the variable performance of Cloud VMs and dynamic workload make it challenging to automate elasticity.

We have presented the design and evaluation of ElastMan, an Elasticity Manager for Cloud-based key-value stores, that addresses these challenges. ElastMan automatically resizes an elastic service in response to changes in workload, in order to meet SLOs at a reduced cost. 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 allows ElastMan to smooth the noisy signal of the 99th percentile of read latency and thus to use feedback control. The feedback control is used to handle gradual workload and to correct errors in the feedforward control that occur due to the noise caused mainly by the variable performance of Cloud VMs. The feedback controller uses a scale-independent model by indirectly controlling the number of servers through 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 for the Voldemort store in an OpenStack Cloud environment. Our evaluation has shown that ElastMan can handle gradual workload and quickly respond to rapid workload spikes.

Acknowledgements

This research is supported by the End-to-End Clouds project funded by the Swedish Foundation for Strategic Research under the contract RIT10-0043; the CLOMMUNITY project funded by the European Commission under EU FP7 Grant Agreement 317879; the Complex System Engineering project in the ICT-TNG Strategic Research Areas initiative at KTH.

9. REFERENCES

[1] Animoto's Facebook scale-up (visited June 2012). `http://blog.rightscale.com/2008/04/23/animoto-facebook-scale-up/`.

[2] Openstack: Open source software for building private and public clouds. `http://openstack.org/`.

[3] A. Al-Shishtawy, M. A. Fayyaz, K. Popov, and V. Vlassov. Achieving robust self-management for large-scale distributed applications. In *Self-Adaptive and Self-Organizing Systems (SASO), 4th IEEE International Conference on*, pages 31–40, Oct 2010.

[4] A. Al-Shishtawy, V. Vlassov, P. Brand, and S. Haridi. A design methodology for self-management in distributed environments. In *Computational Science and Engineering, CSE'09. IEEE International Conference on*, volume 1, pages 430–436, Vancouver, BC, Canada, August 2009.

[5] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *Network, IEEE*, 14(3):30 –37, May/June 2000.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[7] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 241–252, USA, 2010.

[8] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

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

[11] P. Horn. Autonomic computing: IBM's perspective on the state of information technology, Oct. 15 2001.

[12] R. Iyer, R. Illikkal, O. Tickoo, L. Zhao, P. Apparao, and D. Newell. VM3: Measuring, modeling and managing VM shared resources. *Computer Networks*, 53(17):2873–2887, December 2009.

[13] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.

[14] H. C. Lim, S. Babu, and J. S. Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, pages 1–10, New York, NY, USA, 2010.

[15] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: Online data migration with performance guarantees. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[16] S. J. Malkowski, M. Hedwig, J. Li, C. Pu, and D. Neumann. Automated control for elastic n-tier workloads based on empirical modeling. In *Proceedings of the 8th ACM international conference on Autonomic computing*, ICAC '11, pages 131–140, New York, NY, USA, 2011. ACM.

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

[18] M. Ohara, P. Nagpurkar, Y. Ueda, and K. Ishizaki. The data-centricity of web 2.0 workloads and its impact on server performance. In *ISPASS*, pages 133–142. IEEE, 2009.

[19] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. Osborne/McGraw-Hill, Berkeley, CA, USA, 2nd edition, 2000.

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

[21] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell. Modeling virtual machine performance: challenges and approaches. *SIGMETRICS Perform. Eval. Rev.*, 37(3):55–60, Jan. 2010.

[22] 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*, FAST'11, pages 12–12, USA, 2011.