Cluster Comput
DOI 10.1007/s10586-017-0899-z

OnlineElastMan: self-trained proactive elasticity manager for cloud-based storage services

Ying Liu[1] · **Daharewa Gureya**[1] · **Ahmad Al-Shishtawy**[2] · **Vladimir Vlassov**[1]

Received: 10 March 2017 / Accepted: 2 May 2017

Abstract The pay-as-you-go pricing model and the illusion of unlimited resources in the Cloud initiate the idea to provision services elastically. Elastic provisioning of services allocates/de-allocates resources dynamically in response to the changes of the workload. It minimizes the service provisioning cost while maintaining the desired service level objectives (SLOs). Model-predictive control is often used in building such elasticity controllers that dynamically provision resources. However, they need to be trained, either online or offline, before making accurate scaling decisions. The training process involves tedious and significant amount of work as well as some expertise, especially when the model has many dimensions and the training granularity is fine, which is proved to be essential in order to build an accurate elasticity controller. In this paper, we present OnlineElastMan, which is a self-trained proactive elasticity manager for cloud-based storage services. It automatically evolves itself while serving the workload. Experiments using OnlineElastMan with Cassandra indicate that OnlineElastMan continuously improves its provision accuracy, i.e., minimizing provisioning cost and SLO violations, under various workload patterns.

✉ Ying Liu
yinliu@kth.se

Daharewa Gureya
gureya.daharewa@gmail.com

Ahmad Al-Shishtawy
ahmad@sics.se

Vladimir Vlassov
vladv@kth.se

1 KTH Royal Institute of Technology, Stockholm, Sweden

2 Swedish Institute of Computer Science, Kista, Sweden

Keywords Elasticity controller · Cloud storage · Workload prediction · SLO · Online training · Time series analysis

1 Introduction

Hosting services in the Cloud are becoming more and more popular due to a set of desired properties provided by the platform, such as low application setup cost, professional platform maintenance and elastic resource provisioning. Elastically provisioned services are able to use platform resources on demand. Specifically, VMs are spawned when they are needed for handling an increasing workload and removed when the workload drops. Since users only pay for the resources that are used to serve their demand, elastic provisioning saves the cost of hosting services in the Cloud.

On the other hand, services are usually provisioned to match a certain level of quality of service (QoS), which is usually defined as a set of service level objectives (SLOs) in Cloud context. Thus, there are two contradictory goals to be achieved, i.e., saving the provisioning cost and meeting the SLO, while services are elastically provisioned.

Elastic provisioning is usually conducted automatically by an elasticity controller, which monitors the system status and makes corresponding decisions to add or remove resources. An elasticity controller needs to be trained, either online or offline, in order to make it smart enough to make such decisions. Generally, the training process allows the controller to build up a model that correlates monitored parameters, such as CPU or incoming workload, to controlled parameters, i.e., the SLO, which could be, for example, percentile request latency. The accuracy of the model, directly affects the accuracy of the elasticity controller, which dominates service provisioning cost and commitment of the SLO.

It is non-trivial to build an accurate and efficient elasticity controller. Recent works have been focusing on improving the accuracy of elasticity controllers by building different control models with various monitored/controlled metrics [1–8]. However, none of the works have considered the practical usefulness of an elasticity controller, which involves the following challenges. First, an elasticity controller usually needs to be tailored according to a specific application. To be concise, sometimes, it requires complicated instrumentations to the provisioned application or even not possible to obtain the metrics that are used to build the control model. Furthermore, even with all the metrics, it requires tremendous and tedious works to train the control model. A general training procedure involves the redeployment and reconfiguration of the application and collecting and analyzing data by running various workloads against various configurations of the application. Second, the hosting environment of the provisioned application may change due to some unmonitored factors, for example, platform interference or background maintenance tasks. Then, even with well-trained control models, it may not be able to adjust to these factors and leads to inaccurate control decisions. Third, it is always too late for the elasticity controller to react to a workload increase when the workload is already saturating the application. Thus, we argue that prediction of the workload is always a compulsory element to an elasticity controller.

In this work, we propose OnlineElastMan, which is a generic elasticity controller for distributed storage systems. It excels its peers with its practical aspects, which includes straightforward obtainable control metrics, automatically online trained control models and embedded generic workload prediction module. It makes OnlineElastMan an "out-of-the-box" elasticity controller, which can be deployed and adopted by different storage systems without complicated tailoring/configuring efforts. Specifically, OnlineElastMan requires only monitoring on the two most generic metrics, i.e., incoming workload and service latency, which is obtainable from most of the storage systems without complicated instrumentation. Using the monitored metrics, OnlineElastMan analyzes the workload composition in depth, which includes read/write request intensity and data size of the requested item, which defines the dimensions of a control model. OnlineElastMan can easily plug in more interested dimensions if needed. After fixing the dimensions, a multi-dimensional control model can be automatically built and trained online while the storage system is serving requests. After a sufficient amount of warm up on the control model, OnlineElastMan is able to issue accurate control decision based on the incoming workload. Furthermore, the control model continuously improves itself online to adjust to unknown/unmodeled events of the operating environment. Additionally, a generic workload prediction module is also integrated to facilitate the decision making of OnlineElastMan. It allows OnlineElastMan to scale the storage system well in advance to prevent SLO violations caused on workload increase and scaling overhead [7]. Specifically, the prediction module aggregates multiple prediction algorithms and chooses the most appropriate prediction algorithm based on the current workload pattern using a weight majority selection algorithm. Contributions of the paper are as follows.

- Implementation of an "out-of-the-box" generic elasticity controller framework, which is easily applicable to most of the distributed storage systems.
- Integration of an online self-trained control model to OnlineElastMan, which avoids repetitive and tedious system reconfiguring and model training.
- Proposal of a multi-dimensional control model based on workload characteristics, which proves to have better control accuracy.
- Realization of a generic workload prediction module in OnlineElastMan, which is adjustable to multiple workload patterns.
- Open-source implementation[1] of OnlineElastMan framework.

2 Problem statement

There is a large body of work on elasticity controllers for the Cloud [2–8]. Most of them focus on improving the control accuracy of the controller by introducing novel control techniques and models. However, none of them tackles the practical issues regarding the deployment and application of the controllers. We examine the usefulness of an elasticity controller while deploying it in a Cloud environment. Specifically, we investigate the configuration steps for an elasticity controller before it starts provision services. Typically, it involves the following steps to setup an elasticity controller.

1. Acquire metrics for the elasticity controller from the provisioned application or the host platform.
2. Deploy the provisioned application in order to construct a training case for the elasticity controller.
3. Configure the provisioned application according to the deployment.
4. Configure and run a specific synthesized workload against the application.
5. Collect training data from the training case and train the control model accordingly.
6. Repeat step 2–5 until the control model is fully trained before serving the real workload.

[1] https://github.com/gureya/OnlineElasticityManager.

It is intuitively clear that the more metrics considered in a control model, the more accurate it will be. However, increasing the metric dimensions of a control model comes with a significant amount of overhead during the training phase. Specifically, training a control model with only 3 dimensions results in 27 (3^3) training cases when only 3 trials/runs are conducted for each dimension. This means that steps 2–5 needs to be repeated 27 times to train the control model. Obviously, it is extremely time consuming to train a control model manually, especially when the model has many dimensions, which is needed for higher control accuracy.

OnlineElastMan alleviates the training process with online training. Specifically, the model automatically trains and evolves itself while serving the workload. After a short period of warm up, the controller is able to provision the underlying application accurately. Thus, it is no longer needed to manually and repetitively reconfigure the system in order to train the model. Furthermore, in order to make OnlineElastMan as general as possible, its input metrics are easily obtainable from the application. Specifically, it directly uses the information in the incoming workload, which does not need application specific instrumentation, and service latency, which is the most accurate and direct reflection of QoS and can be easily sampled from system entry points or proxies.

On the other hand, previous works [2,7] have demonstrated that, in order to keep the SLO commitment, a storage system needs to scale up in advance to tackle with a workload increase since scaling a storage system involves non-negligible overhead. Thus, we have made a design choice to integrate a workload prediction module for OnlineElastMan. Again, to make it as general as possible, the workload prediction module is able to produce accurate workload prediction for various workload patterns. Specifically, it has integrated several prediction algorithms that are designed to cope with different time series patterns. The most appropriate prediction algorithm is chosen online using a weight majority selection algorithm.

3 Background

In this section we lay out the necessary background for the paper. This include Cloud computing, elastic services, stateful services, feedback control, and feedforward control.

3.1 Cloud computing and elastic services

Cloud computing, with its pay-as-you-go pricing model, provides an attractive solution to host the ever-growing number of web applications [9]. 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, by provisioning additional resources, 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.

3.2 Stateful services

Modern applications, such as Social Networks, Wikis, and Blogs, are data-centric, which require frequent data access [10]. 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 [11]. With the rapid increase of the number of users, the poor scalability of a typical data-tier with ACID [12] properties limited the scalability of web 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 [13], Dynamo [14], and Cassandra [15]. 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 applications is that a certain service, feature, or topic might suddenly become popular resulting in a workload spike [16,17]. The fact that storage is a stateful service complicates the problem since only a particular subset of servers host data of the popular item. For stateful services, scaling is usually combined with a rebalancing step necessary to redistribute the data among the new set of servers.

These challenges have led to the need for an automated management of the data-tier, to make it capable to quickly and efficiently respond to changes in the workload in order to meet the required SLOs of the storage service.

3.3 Feedback versus feedforward control

In computing systems, a controller [18] or an autonomic manager [19] 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 tolerate noise and disturbance such as unexpected 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 advantages of feedforward control include being faster than feedback control in reaching the optimum point and avoiding oscillations and overshoot.

The major drawback of feedforward control is that it is sensitive to unexpected disturbances that are not accounted for (modelled) in the system model. Addressing this issue may results in a relatively complex system model, compared to feedback control, that tries to capture all possible states of the modelled system. Another approach is to apply online training that continuously adapts the system model in order to reflect changes in the physical system.

3.4 Target system

We are targeting multi-tier web applications (the left side of Fig. 1). We are focusing on managing the data-tier because of its major effect on the performance of web applications, which are mostly data centric [10]. For the data-tier, we assume horizontally scalable key-value stores due to their popularity in many large scale web 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 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 [13] and Cassandra [15], provide rebalancing tools.

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

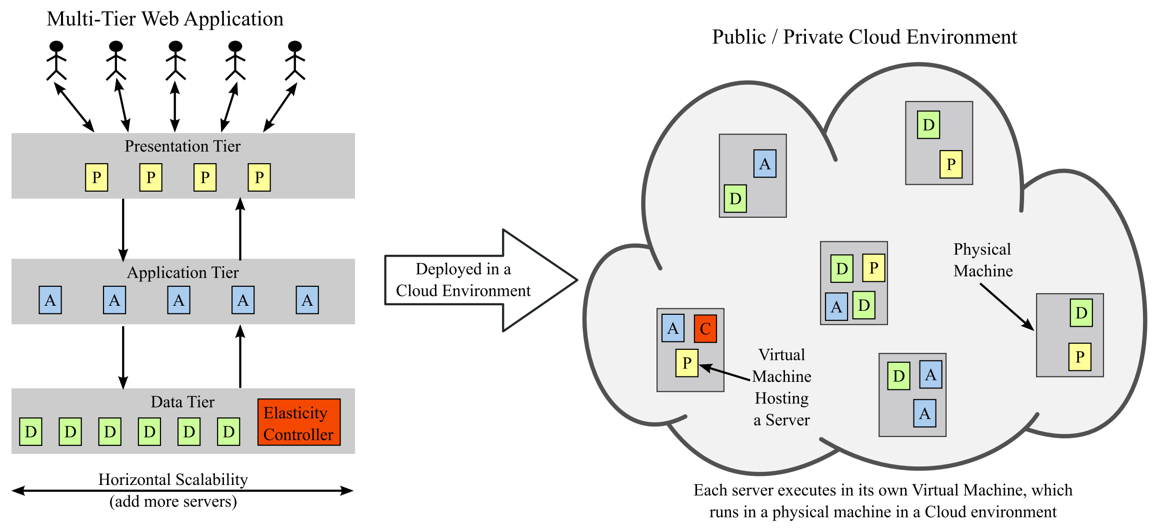

Multi-Tier Web Application
Presentation Tier
P
Application Tier
A
Data Tier
D
Elasticity Controller
Horizontal Scalability (add more servers)
Deployed in a Cloud Environment
Public / Private Cloud Environment
Physical Machine
Virtual Machine Hosting a Server
C
Each server executes in its own Virtual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

Fig. 1 Multi-tier web application with elasticity controller deployed in a cloud environment

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 [20,21].

3.5 Cassandra

We have chosen Cassandra as our targeted underlying distributed storage system. Cassandra [15] is open sourced under Apache licence. It is a distributed storage system which is highly available and scalable. It stores column-structured data records and provides the following key features:

- **Distributed and decentralized architecture** Cassandra is organized in a peer-to-peer fashion. Specifically, each node performs the same functionality in a Cassandra cluster. However, each node manages a different namespace, which is decided by the hash function in the DHT. Comparing to Master-slave, the design of Cassandra avoids single point of failure and maximizes its scalability.
- **Horizontal scalability** The peer to peer structure enables Cassandra to scale linearly. The consistent hashing implemented in Cassandra allows it to swiftly and efficiently locate a queried data record. Virtual node techniques are applied to balance the load on each Cassandra node.
- **Tunable data consistency level** Cassandra provides tunable data consistency options, which is realized through different combinations of read and write APIs. These APIs use *ALL, EACH_QUORUM, QUORUM, LOCAL_QUORUM, ONE, TWO, LOCAL_ONE, ANY, SERIAL, LOCAL_SERIAL* to describe read/write calls. For example, the *ALL* option means the Cassandra reads/writes all the replicas before returning to clients. The explanation of each read/write option can be easily found on Apache Cassandra website.
- **An SQL like query tools—CQL** The common access interface in Cassandra is exposed using Cassandra Query Language (CQL). CQL is similar to SQL in its semantics. For example, a query to get a record whose id equals to 100 results the same statement in both of CQL and SQL (*SELECT * FROM USER_TABLE WHERE ID= 100*). It reduces the learning curve for developers to use CQLs and get started with Cassandra.

4 OnlineElastMan design

In this section, we present the design of OnlineElastMan by explaining its three major components, i.e., workload prediction, online model training, and elasticity controller. Figure 2 presents the architecture of OnlineElastMan. Components operate concurrently and communicate by message passing.

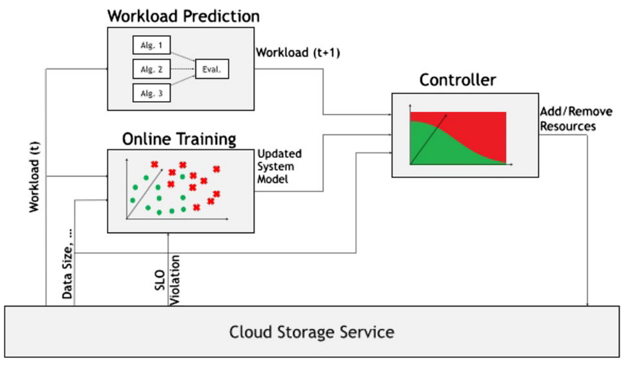


Fig. 2 OnlineElastMan architecture

Briefly, workload prediction module takes input from the current workload and predicts workload for a near future (the next control period). Online Model training module updates the current model by mapping and analyzing the monitored workload and the performance of the system. Then, the elasticity controller takes the predicted workload and consults the updated performance model to issue scaling commands by calling the Cloud API to add or remove servers for the underlying storage system.

4.1 Monitored parameters

Auto-scaling technique requires a monitoring component that gathers various metrics that reflect the realtime status of the targeted system at an appropriate granularity (e.g per second, per minute, per hour). It is essential to review the metrics that can be obtained from the target system and the metrics that best reflect the status of the system. To ease the configuration of OnlineElastMan framework and to make it as general as possible, we consider the target storage system as a black box. OnlineElastMan adopts the most general and direct metrics that dominate the QoS of the targeted storage system. Specifically, we take the **workload**, which causes the variations in those system metrics, directly as the input. OnlineElastMan requires the workload monitoring to provide the read/write intensity and the size of the requested data in the workload in small intervals. The monitored data can be obtained by sampling the traffic passing through the entry points, e.g. proxies or load balancers, of the storage system. The **percentile latency**, which defines and directly reflects the QoS, is collected either from entry proxies or the storage system itself depending on the design and workflows of storage systems. Then, the collected percentile latency is used to adjust and improve control decisions/models. In Sect. 5.1.1, we provide details on how we obtain these metrics in a distributed storage system, such as, Cassandra [15].

4.2 Multi-dimensional online model

One of the core components in OnlineElastMan is the multi-dimensional SML (Statistical Machine Learning) model, which is learnt online. It correlates the input metrics (workload characteristics) with the SLO (percentile latency). The goal of the model is to keep the target system operating with the percentile latency varying only in a small controlled range. It is intuitively clear that with more provisioned resources (VMs), the system is able to respond to requests with reduced latency. However, on the other hand, we would also like to provision as little VMs as possible to save the provisioning cost. Thus, the controlled latency range is always desired to be slightly under (just satisfying) the percentile latency requirement defined in the SLO to minimize the provisioning cost. We refer this region to be **optimal operational region** (OOR), where a system is not very much over-provisioned but satisfying the SLO.

In order to keep the system operating in the OOR while the incoming workload is dynamic, an elasticity controller needs to react to the workload changes and allocating/de-allocating VMs to the system. Previous works [3,4,7,22] designs an elasticity controller based on an offline statistical model. OnlineElastMan builds the model online and continuously improves/updates the model while serving requests. The online training feature frees system administrators from the tedious offline model training procedure, which includes repetitive system configurations, system deployments, model updates, etc., before putting the controller online. Additionally, the continuous evolving model in OnlineElastMan enables the system to survive with factors that are not considered in the model, e.g. platform interference [23,24].

Specifically, the online model is built with the monitored parameters mentioned in Sect. 4.1. It classifies whether a VM is able to operate in the OOR under the current workload, which breaks down to the intensity of read and write requests and the requested data size. Ideally, a storage node hosted in a VM can be either operating under commitment to the SLO or with violation to the SLO. Therefore, with a given workload and VM flavor, the classifier model is a line that separates the plane into two regions, in which the SLO is either met or violated as shown in Fig. 5. Different models need to be built for different VM flavors and different storage systems hosted. While building the model as depicted in Fig. 3, there are several configurable parameters that affect the accuracy of the model.

Granularity of the model Since the collected data for the model can be decimal, it is impossible to analyze the data with infinite combinations. We group the collected data with a pre-defined granularity, which makes a two-dimensional plane to be separated to small squares or a three-dimensional plane to be separated to small cubes. These squares and cubes

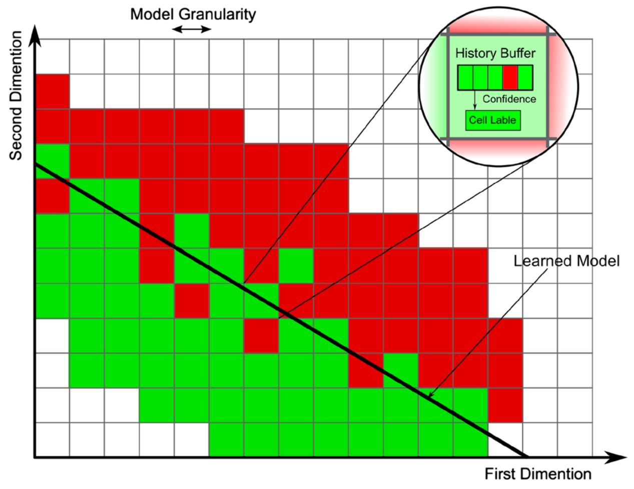


Fig. 3 Building the multi-dimensional online model

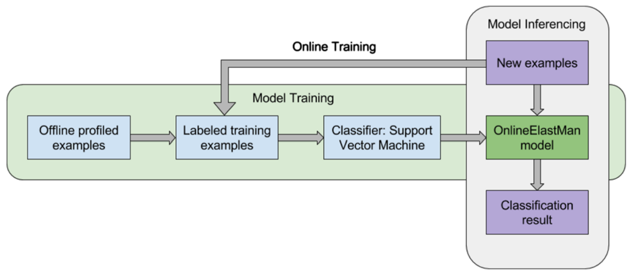


Fig. 4 Classification using SVM

are the groups where data are accumulated and analyzed. The granularity of data groups can be configured depending on the memory limits and the precision requirements of the model.

Historical data buffer For data collected and mapped to each group, we maintain a historical record for the most recent n reads and writes.

Confidence level The historical data in each group is analyzed to define whether the workload that corresponds to the data collected in this group violates the SLO or not. For example, 95% confidence level implies that 95% of all the Read/write percentile latency sampled satisfy the SLO.

Update frequency The model updates itself periodically with a fixed configurable rate. A higher update frequency allows the model to swiftly adapt to execution environment changes while a lower update frequency makes the model more stable and tolerate transient execution environment changes.

4.2.1 SVM binary classifier

SVMs have become popular classification techniques in a wide range of application domains [25]. They provide good performance even in cases of high-dimensional data and a small set of training data. Figure 4 shows the flow of a classification task using SVM. Briefly, we first train the model offline using systematically profiled data. Then, we put the model online to let it evolve itself.

Here, we describe the algorithm of SVM applied to build the model of OnlineElastMan. Each instance of the training set contains a class label and several features or observed variables. The goal of SVM is to produce a model based on the training set. More concretely, given a training set of instance-label pairs $(x_i, y_i), i = 1, ..., l$ where $x_i \in R^n$ and $y_i \in \{1, -1\}^l$, the SVM classification solves the following optimization problem:

$$min_{w,b} \qquad \| w \|^2 + C \sum_i \xi_i \tag{1}$$

subject to:

$$\begin{aligned} y^{(i)}(w^T x^i + b) &\geq 1 - \xi_i, \quad & i = 1, 2, \ldots, m \\ \xi_i &\geq 0, \quad & i = 1, 2, \ldots, m \end{aligned} \tag{2}$$

After solving, the SVM classifier predicts 1 if $w^T x + b \geq 0$ and -1 otherwise. The decision boundary is defined by the following line:

$$w^T x + b = 0 \tag{3}$$

Generally, the predicted class can be calculated using the linear discriminant function:

$$f(x) = wx + b \tag{4}$$

x refers to a training pattern, **w** as the weight vector and **b** as the bias term. **wx** refers to the dot product, which calculates the sum of the products of vector components $w_i x_i$. For example, in case of training set with three features (e.g. x, y, z), the discriminant function is simply:

$$f(x) = w_1 x + w_2 y + w_3 z + b \tag{5}$$

SVM provides the estimates for w_1, w_2, w_3 and b after training.

Given Eq. 3, the SML model is a line (Fig. 5) when only monitoring read/write request intensity in the workload or a plane (Fig. 6) when another dimension, i.e., data size, is modeled. Figure 7 is a 2 dimensional projection of Fig. 6, which shows that different data sizes cause different separations of the 2 dimensional model space. It indicates that data size plays an essential role to build an accurate control model for storage systems. The line/plane separation in the model represents the maximum workload that a VM can serve under the specified SLO (percentile latency).

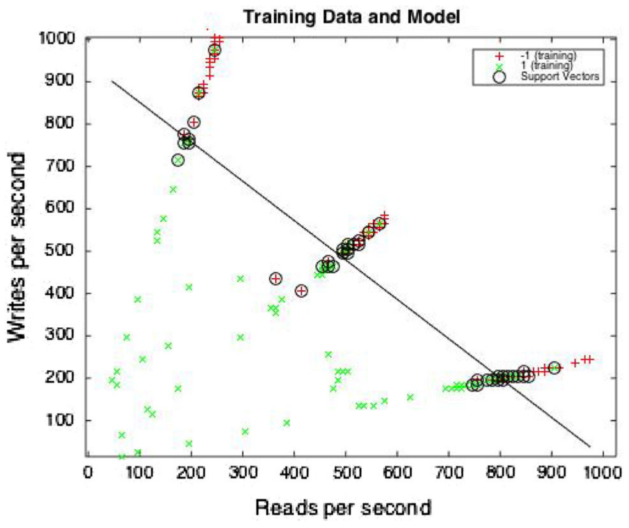


Fig. 5 2 dimensional SVM performance model

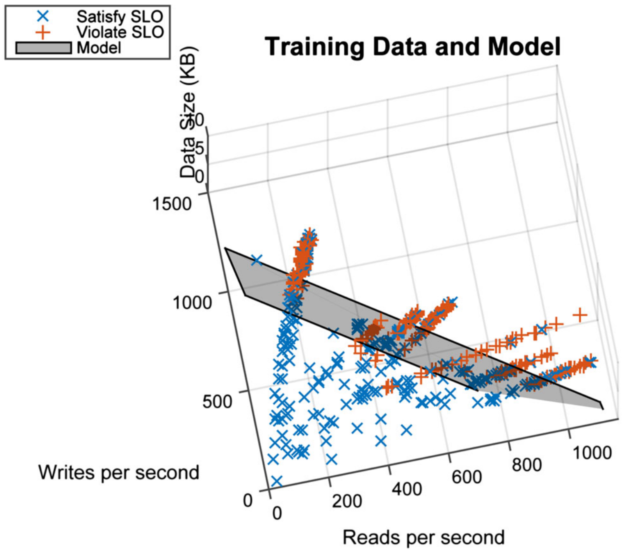


Fig. 6 3 dimensional SVM performance model taking into account request data size

Online model Training Using the SVM model training technique, the performance model is updated periodically according to the **update frequency** using the data in the **historical data buffer** processed with the **confidence level**.

We believe that every VM can have significant performance difference even when they are spawned with the same favor. This can be caused by the interference from the host platform [23,24] or background tasks, such as data migration [7]. Thus, individual SML model is built for each VM participating in the system. They automatically evolve and update continuously while the system is serving workload. Periodically, the updated SML models for each VM are sent to the elasticity controller module to make scaling decisions.

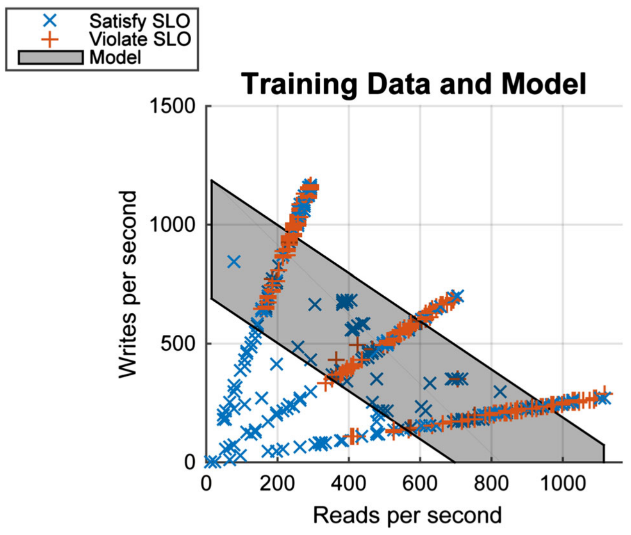


Fig. 7 Top angle view of the SVM model, where the model plane is projected to a 2 dimensional surface and the *shaded area* is caused by the varying data sizes

4.3 Elasticity controller

An elasticity controller makes scaling decisions in configurable control periods/intervals to prevent system from oscillations. When making a scaling decision, the elasticity controller collects the aggregation of the input workload of all VMs (W) and the aggregation of the capacity of all VMs (C). The capacity of a VM is the maximum workload that it can handle under the SLO, which is obtained from the multi-dimensional SML model. The elasticity controller also observes the input workload (w_x) and capacity (c_x) for each VM individually to identify fine-grained SLO violations. Specifically, the capacity of each VM is calculated by intersecting the plane of its SML model with a line from the origin that points to the current workload representation, which is a point corresponding to read and write workload intensity and the averaged data size. The capacity of the VM is the intersection point, which represents the capability to serve workload with specific read/write request intensity of a specific data size. If the current workload representation point is beyond the capacity representation point in the model, the SLO is violated.

The responsibility of an elasticity controller is to keep the provisioned system operating with commitment to the SLO. The strictest requirement is that each VM operates with the commitment to the SLO, which can be denoted by $\forall i \in N, w_i < c_i$, where N is the complete set of all participating VMs. However, this is not trivial to achieve without over-provisioning the system because of the imbalance of workload distribution. It is challenging to balance workload in storage systems with respect to each VM. This is because that storage systems are stateful, i.e., usually each VM is responsible only for a part of the total data stored. Thus, a specific request can only be served by a specific set of VMs, which host the requested data. Given that different storage systems have different data distribution as well as load balancing strategies and OnlineElastMan is designed to be a generic framework to provision storage systems elastically, we choose not to manage workload/data distribution for provisioned systems. Furthermore, managing data distribution or rebalancing among VMs is orthogonal to the design goal of OnlineElastMan. Nevertheless, OnlineElastMan provides/outputs suggestions for workload distributions to each participating VMs based on their capacity learnt from our SML models.

In order to tolerant load imbalance among VMs to some extent, OnlineElastMan introduces an optional tolerance factor α when computing scaling decisions to prevent too much over-provisioning. Specifically, a scaling up decision is issued when the SLO violation $c_x < w_x$ is observed from more than α VMs, where $\alpha \geq 0$. When $\alpha = 0$, there is no tolerance on load imbalance. The number of VMs to add is calculated individually for each VM and aggregated globally. $\frac{w_x - c_x}{c_x}$ number of VMs with the same flavor as c_x is expected to be added. Thus, when $\frac{w_x - c_x}{c_x} < 0$, it represents that a VM has more capacity than the incoming workload. We aggregate results of $\frac{w_x - c_x}{c_x}$ on each VM flavors and ceiling the aggregated results. When the result on a specific VM flavor is negative, we do nothing because it is in a scaling up procedure. When the result on a specific VM flavor is positive, we add the number of VMs of that flavor accordingly.

For scaling down, there is also a corresponding load imbalance tolerance factor β. β denotes the number of VMs, which are over-provisioned, in each VM flavor. A scaling down procedure is triggered by first satisfying that there is no VM that violates the SLO, which gives $\forall i \in N, w_i < c_i$, where N is the complete set of all participating VMs. Then, the number of VMs to de-allocate is calculated through similar process comparing to scaling up. The aggregated results of $\frac{w_x - c_x}{c_x}$ on each VM flavors are floored after subtracting β. Last, the corresponding number of VMs are de-allocated when the floored results are greater than zero.

When a scaling up/down decision is made, the elasticity controller interact with Cloud/platform API to request/release VMs. Where applicable, the elasticity controller also calls the API to rebalance data to the newly added VMs or to decommission the VMs that are about to be removed. Adding/removing VMs to a distributed storage system introduce a significant amount of data rebalance load in the background. This leads to fluctuations on sensitive performance measures, such as percentile latency. Usually, the extra data rebalancing load is not long lasting. So, this fluctuation can be filtered out in our SML model with proper setting on the **confidence level** and **update frequency** introduced in Sect. 4.2.

4.4 Workload prediction

An optional but essential component of OnlineElastMan is the workload prediction module. It is always too late to make a scaling out decision when the workload is already increased since preparing VMs involve non-negligible overhead, especially for storage systems, which require data to be migrated to the newly added VMs. Thus, there is a prediction module that facilitates OnlineElastMan to make decisions in advance.

Often, there are patterns that can be found in the workload, such as the diurnal pattern [26]. These patterns become vague when the workload is distributed to each VM. Thus, we are not predicting the incoming workload for each VM. Rather, the workload is predicted for the whole system. Then, it is proportionally calculated for each VM based on the current workload portion that is served by the VM. Finally, instead of using the current incoming workload to make a scaling decision in the previous section, we are able to use the predicted workload as the input.

However, even predicting the workload for the whole system is not trivial since there are many factors that contribute to the fluctuation of the workload [27]. Some workloads have repetitive/cyclic pattern, such as diurnal patterns or seasonal patterns while some other workloads experience exponential growth over a short period of time, which can be caused by market campaigns or special offers. Considering that there are no perfect predictors and different applications' workloads are distinct, no single prediction algorithm is general enough to be suitable for most workloads. Thus, we have studied and analyzed several prediction algorithms that are designed for different workload patterns, i.e., **the regression trees**, **first-order autoregressive**, **differenced first-order autoregressive**, **exponential smoothing**, **second-order autoregressive** and **random walk**. Then, a weighted majority algorithm (Sect. 4.4.3) is used to select the best prediction algorithm.

4.4.1 Regression trees model

Regression trees predict responses to data and are considered as a variant of decision trees. They specify the form of the relationship between predictors and a response. We first build a tree using the time series data through a process known as recursive partitioning (Algorithm 1) and then fit the leaves values to the input predictors like Neural Networks. Particularly, to predict a response, we follow the decisions in the tree from the root node all the way to a leaf node which contains the response.

Algorithm 1: Recursive Partitioning Algorithm

```
Data: A set of $N$ data points, $x_i$, i = 1,...,n
Result: A regression tree
if termination criterion exist then
    Generate Leaf Node and allocate it a Given Value;
    Return Leaf Node;
else
    Identify Best Splitting test $s*$;
    Generate node $t$ with $s*$;
    Left_branch($t$) =
    $RecursivePartitioning(< x_i, y_i >: x_i = s*)$;
    Right_branch($t$) =
    $RecursivePartitioning(< x_i, y_i >: x_i \neq s*)$;
    Return Node $t$;
```

4.4.2 ARIMA

Autoregressive moving average (ARMA) is one of the most widely used approaches to time series forecasting. ARMA model is convenient for modelling time series data which is stationary. In order to handle non-stationary time series data, ARMA model adopts a differencing component to help deal with both stationary and non-stationary data. This class of models with differencing component is referred to as the autoregressive integrated moving average (ARIMA) model. Specifically, ARIMA model is made up of autoregressive (AR) component of lagged observations, a moving average (MA) of past errors and a differencing component (I) needed to make a time series to be stationary. The MA component is impacted by past and current errors while the AR component shows the recent observations as a function of past observations [28].

In general, an ARIMA model is parametrized as ARIMA(p,d,q), where **p** is the number of autoregressive terms (order of AR), **d** is the number of differences needed for stationarity, and **q** is the number of lagged forecast errors in the prediction equation (order of MA). The following equation represents a time series expressed in terms of AR(n) model:

$$Y^{'}(t) = \mu + \alpha_1 Y(t-1) + \alpha_2 Y(t-2) + \cdots + \alpha_n Y(t-n) \quad (6)$$

Equation 7 represents a time series expressed in terms of moving averages of white noise and error terms.

$$Y^{'}(t) = \mu + \beta_1 \epsilon(t-1) + \beta_2 \epsilon(t-2) + \cdots + \beta_n \epsilon(t-n) \quad (7)$$

In OnlineElastMan, apart from regression tree, we have integrated five ARIMA models, which are the first-order autoregressive ($ARIMA(1,0,0)$), the differenced first-order autoregressive ($ARIMA(1,1,0)$), the simple exponential smoothing ($ARIMA(0,1,1)$), the second-order autoregressive ($ARIMA(2,0,0)$) and the random walk ($ARIMA(0,1,0)$). In our view, they can capture almost all the common workload patterns. For example, the first-order autoregressive model performs well when the workload is stationary and autocorrelated while, for non-stationary workload, a random walk model might be suitable. Then, the challenge is to detect and select the most appropriate prediction model during runtime.

4.4.3 The weighted majority algorithm

A Weighted Majority Algorithm (WMA) is implemented to select the best prediction model during runtime. It is a machine learning algorithm that is used to build a combined algorithm from a pool of algorithms [29]. The algorithm assumes that one of the known algorithms in the pool will perform well under the current workload without prior knowledge about the accuracy of the algorithms. The WMA have many variations suited for different scenarios including infinite loops, shifting targets and randomized predictions. We present our WMA implementation in Algorithm 2. Specifically, the algorithm maintains a list of weights w_1,...,w_n for each prediction algorithm. The prediction result from the most weighted algorithm, based on a weighted majority vote, is selected and returned.

Algorithm 2: The Weighted Majority Algorithm

```
1. Initialize the weights $w_1$, ..., $w_n$ of all the prediction
algorithms to a positive weight (1).
2. Return the prediction result of the prediction algorithm with
the highest weight.
3. Compare the predicted value with the real value, penalize the
prediction algorithms, which missed the prediction more than a
predefined tolerance interval $n$, by multiplying their weights with
a fixed penalize factor $m$ ($0 \leq m < 1$).
4. Wait until next prediction interval and go to 2.
```

The prediction module of OnlineElastMan is shown in Fig. 8. Additional prediction algorithms can be plugged into the prediction module to handle more workload patterns.

4.5 Putting everything together

OnlineElastMan operates according to the flowchart as illustrated in Fig. 9. The incoming workload is fed to two modules,

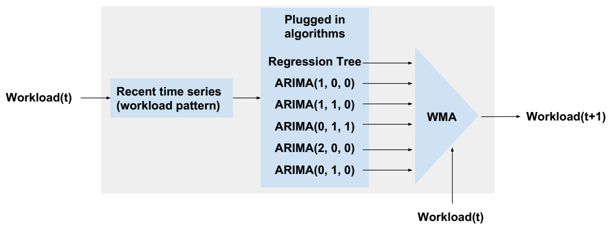


Fig. 8 Architecture of the workload prediction module

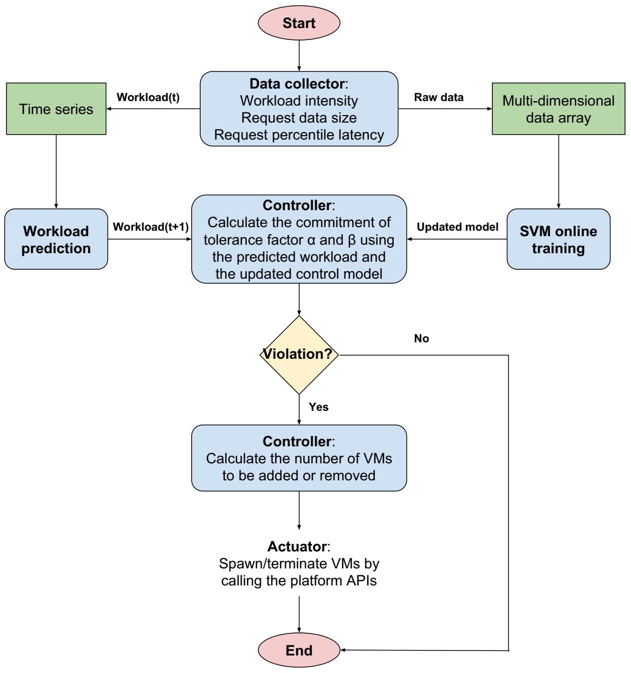

Start
Time series
Workload(t)
Data collector: Workload intensity Request data size Request percentile latency
Raw data
Multi-dimensional data array
Workload prediction
Workload(t+1)
Controller: Calculate the commitment of tolerance factor α and β using the predicted workload and the updated control model
Updated model
SVM online training
Violation?
No
Yes
Controller: Calculate the number of VMs to be added or removed
Actuator: Spawn/terminate VMs by calling the platform APIs
End

Fig. 9 Control flow of OnlineElastMan

i.e., the prediction module and the online training module. The prediction module utilizes the current workload characteristics to predict the workload in the next control period using the algorithm described in Sect. 4.4. The online training module records the current workload composition and samples the service latency under current workload. Then, the module trains the performance model with the **update frequency**. The actuation is calculated based on the predicted workload for the next control period using the updated performance model according to the algorithm explained in Sect. 4.3. Finally, the actuation is carried out on the Cloud platform that hosts the storage service.

5 Evaluation

We evaluate OnlineElastMan from two aspects. First, we show the accuracy of the prediction module, which consists of six prediction algorithms. It directly influences the provision accuracy of OnlineElastMan since it is an essential input parameter for the performance model. Then, we present the evaluation results of OnlineElastMan when it dynamically provisions a Cassandra cluster with the application of the online multi-dimensional performance model.

Our evaluation is conducted in a private Cloud, which runs OpenStack software stack. Our experiments are conducted on VMs with two virtual cores (2.40 GHz), 4 GB RAM and 40 GB disk size. They are spawned to host storage services or benchmark clients. OnlineElastMan is configured separately

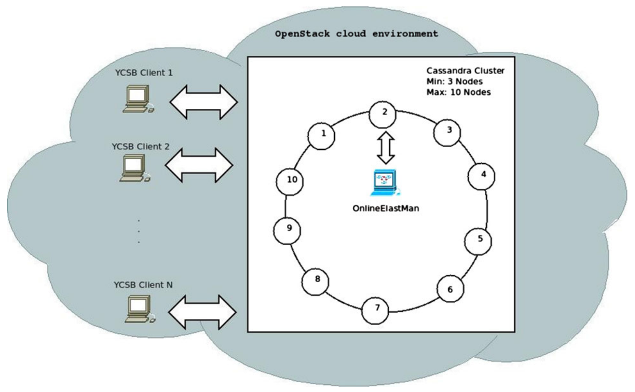


Fig. 10 Different number of YCSB clients are used to generate workload with different intensity. OnlineElastMan resizes the Cassandra cluster according to the workload

on one of the VMs. The overview of the evaluation setup is presented in Fig. 10.

5.1 Evaluation environment

5.1.1 Underlying storage system

Cassandra (version 2.0.9) is deployed as the underlying storage system and provisioned by OnlineElastMan. Cassandra is chosen because of its popularity to be used as a scalable backend storage by many companies, e.g. Facebook. Breifly, Cassandra is a distributed replicated database, which is organized with distributed hash tables. Since a Cassandra cluster is organized in a peer to peer fashion, it achieves linear scalability. Minimum instrumentation is introduced to Cassandra's read and write path as shown in Fig. 11. The instrumented library samples and stores service latency of requests in its repository. OnlineElastMan's data collector component periodically, every 5 min in our experiments, pulls collected access latencies from the repository on each Cassandra node. The collected request samples from each Cassandra node are used by the prediction module and the online training module of OnlineElastMan as shown in Fig. 9 The Cassandra rebalance API is called to redistribute data when adding/removing Cassandra nodes.

5.1.2 Workload benchmark

We adopt YCSB (Yahoo! Cloud System Benchmark) (version 0.1.4) to generate workload for our Cassandra cluster. We choose YCSB because of its flexibility to synthesize various workload patterns, including the varying read/write request intensity and the size of the data propagated. Specifically, we configure YCSB clients with the parameters shown in Table 1. In order to generate stable workload to Cassandra, a fixed request rate (1200 req/s) is set to each YCSB client

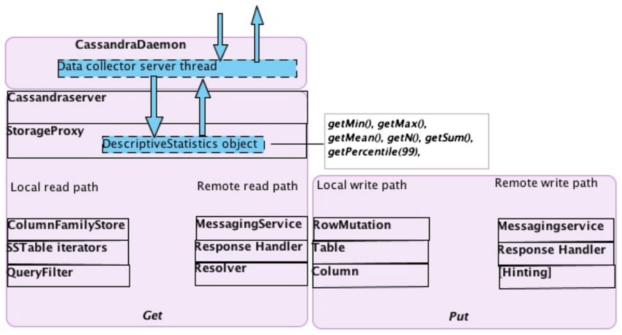


Fig. 11 Cassandra instrumentation for collecting request latencies

Table 1 YCSB configuration

Number of threads	16
Request distribution	Uniform
Record count	100,000
Read proportion	Varied (0.0–1.0)
Update proportion	Varied (0.0–1.0)
Data size	varied (1–20) KB
Replication factor	3
Consistency level	level ONE

hosted on a separate VM. We vary the total amount of workload generated by adding or removing VMs that host YCSB clients.

5.1.3 Multi-dimensional performance model

Our performance model is trained automatically when the input workload varies. OnlineElastMan takes input from the monitored parameters as specified in Sect. 4.1. Specifically, the workload features, including read and write request intensity and request data size, and the corresponding service latency, obtained from Cassandra instrumentation, are associated to train the model. Details on model training is presented in Sect. 4.2.

In practice, the model starts empty and needs to get trained online automatically for some time. This is because that the model is application and platform specific. Thus, it needs a warm up training phase. According to our experiment experience, it takes approximately 20–30 min to train a performance model from scratch. After warm up, the model can be used to facilitate the decision making process of the elasticity controller while serving the workload.

Figure 12 depicts the model built and used in our evaluation. It consists of three input parameters or dimensions, i.e., read/write request intensity and the data size. The controlled parameter is the 99th percentile read latency, which is set to be 35*ms* in our case. As shown in the figure, with more training data, the model (the shaded surface) evolves

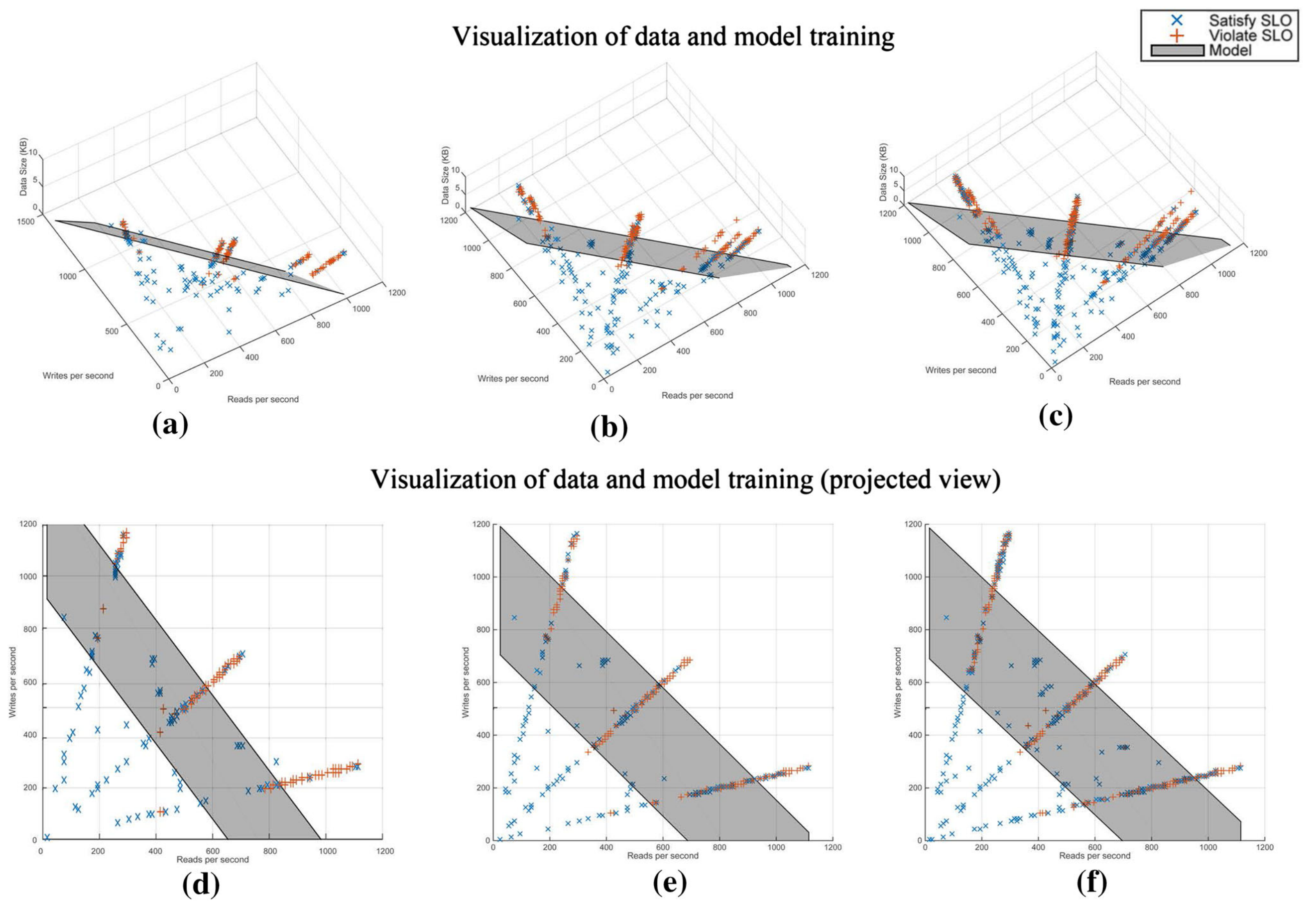


Fig. 12 The training illustration of the 3 dimensional performance model. **a**–**c** are ordered by the length of training period. **d**–**f** are the visualization of **a**–(**c** with data size dimension projected on the other 2 dimensions

itself to a more accurate state. Practically, the performance model is dynamic and evolves while serving the workload. So, it can automatically evolves to a more accurate model that reflects the changes of the operating environment and the provisioned storage system. To be specific, the model adapts to unknown factors, such as application interference or platform maintenance, gradually using updated training data. A more accurate model leads to better provision accuracy when the elasticity controller consults it.

In our experiments, we found out that the rate at which the model evolves affects the accuracy of the decisions made by the controller. The **confidence level** and **update frequency** (as introduced in Sect. 4.2) dictates how fast the model evolves. Ideally, we should have enough confidence about the status (violate SLO or satisfy SLO) of a data point before its status changes. Setting the confidence level low and the update frequency high may result into the model oscillating (unstable model) while the opposite settings of these two parameters may delay the evolution of the model. In our experiments, we set the confidence level as 0.5, i.e., if 50% of all read and write latency queue samples satisfy the SLO then the corresponding data point satisfies SLO and vice versa. The update frequency is set to 5 min. For applications that have distinct phases of operations, to prevent frequent retraining, one can maintain a set of models and dynamically selects the best model for the current input pattern [30].

5.2 Evaluation on workload prediction

We evaluate the prediction accuracy of the workload prediction module using a synthetic workload generated by YCSB. We have synthesized workload with different shapes of workload increase and decrease regarding the total request intensity with a fixed read/write ratio. Figure 13 presents the actual workload generated and the workload predicted by our prediction module. In addition, the choice of the dominant prediction algorithm proposed by the weight majority algorithm is also shown in the figure. As a result, our prediction module is able to achieve as low as 4.60% on the Mean Absolute Percentage Error for such a dynamic workload pattern.

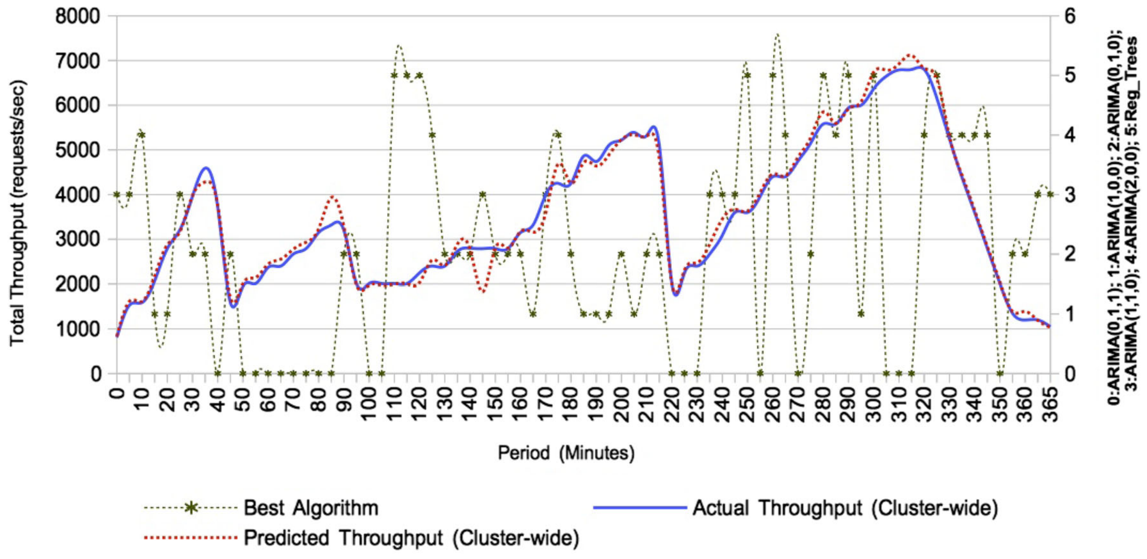


Fig. 13 Workload prediction: the actual workload V.S. the predicted workload

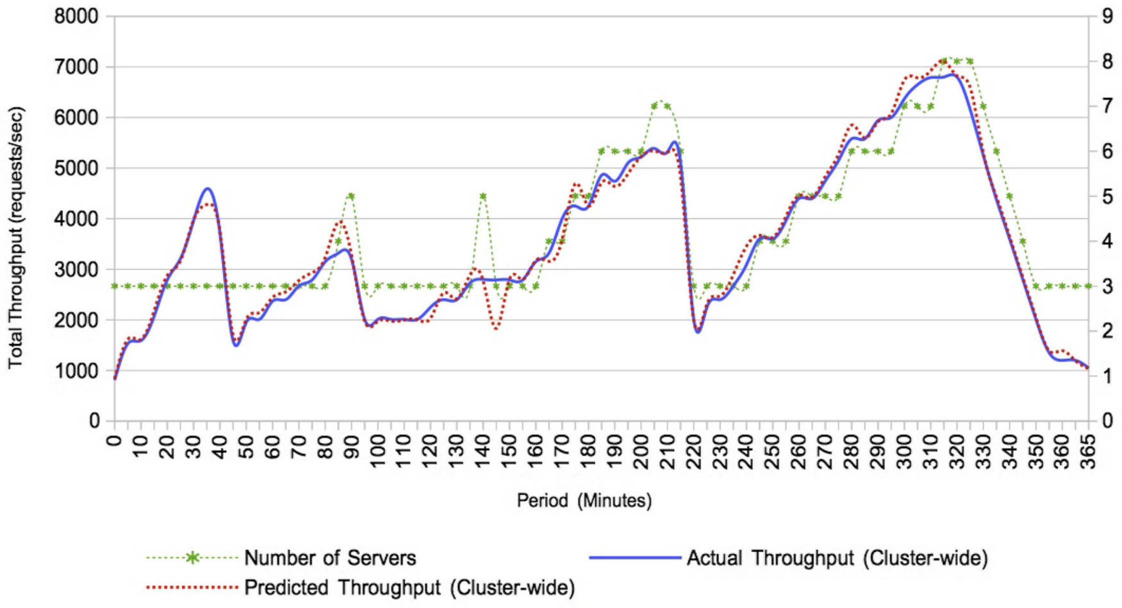


Fig. 14 VMs allocated according to the predicted workload and the updated control model

5.3 Evaluation of OnlineElastMan over Cassandra

We set the goal of OnlineElastMan to keep the 99th percentile of read latency to be $35ms$ as stated in the SLO. The evaluation is conducted with control period set to be 5 min. Even the workload of YCSB is configured to be uniform in our case, we still observe a non-trivial difference on the amount of workload served from different Cassandra storage VMs. To make a tradeoff between the uneven workload served on each VM and preventing over-provisioning, we set the tolerance factor $\alpha = 1$ and $\beta = 0.5$.

As shown in Fig. 14, we start the experiment with 3 Cassandra VMs. From 0 to 40 min, the multi-dimensional performance model is trained and warmed up. The elasticity controller starts to function from 40 min. From 40 to 90 min, the workload increases gradually. It is observable that from 40 to 70 min, the system is over-provisioned, as the percentile latency is far below the SLO boundary as shown in Fig. 15. This is because that the elasticity controller is set to operate with a minimum number of 3 VMs, which corresponds to the replication factor of Cassandra. With the increasing of workload, the elasticity controller gradually adds two VMs from

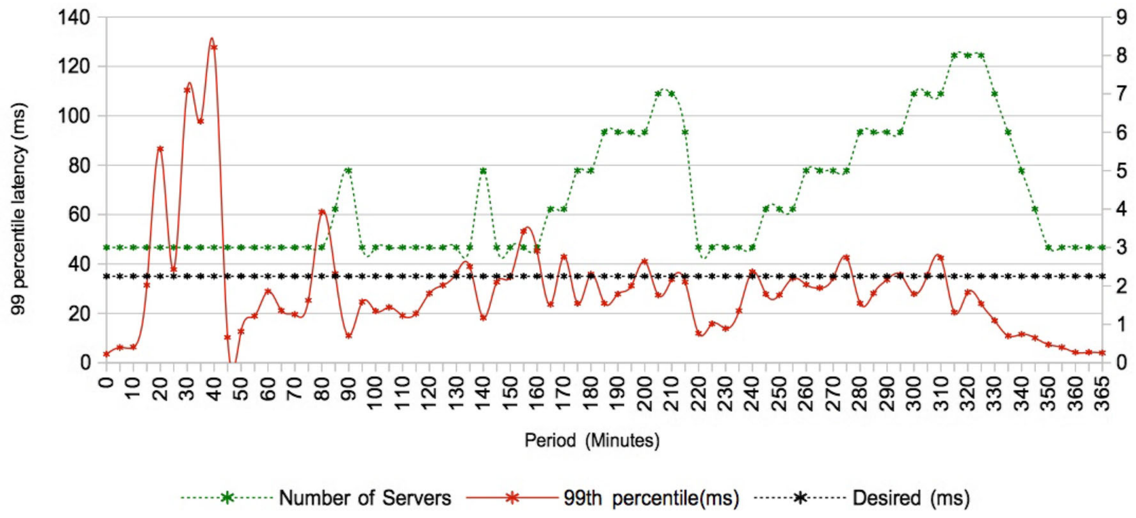


Fig. 15 The aggregated 99th percentile latency from all Cassandra VMs with the allocation of VMs indicated by OnlineElastMan under the dynamic workload

80 min. Then, the workload experienced a sharp decrease from 90 min, but the controller maintains a minimum of 3 Cassandra VMs. We continue to evaluate the performance of OnlineElastMan with another two rounds of workload increase and decrease with different scales (shown from 150 to 220 min and from 220 to 360 min). The evaluation indicates that OnlineElastMan is able to keep the 99th percentile latency commitment most of the time. On the other hand, we observe a small amount of SLO violations under the provisioning of OnlineElastMan. It is because of the tolerance factor α and β, which allows us to tolerate some imbalance of workload distribution to Cassandra nodes.

6 Related work

6.1 Elasticity controllers in practice

Most of the elasticity controllers available in public Cloud services and used nowadays in production systems are policy based and rely on simple if-then threshold based triggers. Examples of such systems include Amazon Auto Scaling (AAS) [31], Rightscale [32], and Google Compute Engine Autoscaling [33]. The wide adoption of this approach is mainly due to its simplicity in practice as it doesn't require pre-training or expertise to get it up and running. Policy based approaches are suitable for small-scale systems in which adding/removing a VM when a threshold is reached (e.g., CPU utilization) is sufficient to maintain the desired SLO. For larger systems, it might be non-trivial for users to set the thresholds and the correct number of VMs to add/remove.

Scryer [34] is a Netflixs predictive auto-scaling engine. It allows them to provision the right number of instances needed to handle the traffic of their customers. Unlike systems such as AAS, Scryer predicts what the needs will be prior to the time of need and provisions the instances based on those predictions. However, its genesis was triggered more by their relatively predictable traffic patterns, which is not always true in a dynamic environment such as Cloud.

6.2 Research on elasticity controllers

Most of the elasticity controllers, which go beyond a simple threshold based triggers, require a model of the target system in order to be able to reason about the status of the system and decide on control actions needed to improve the system. The system model is typically trained offline using historical data and the controller is tuned manually using expert knowledge of the expected workload patterns and service behavior.

Work in this area focuses on developing advanced models and novel approaches for elasticity control such as, ElastMan [4], SCADS Director [3], scaling HDFS [2], ProRenata [7], and Hubbub-scale [8]. Although achieving very good results, most of these controllers ignore the practical aspects of the solution which slowed down the adoption of such controllers in production systems. For example, SCADS Director [3] is tailord for a specific storage service with pre-requisits that are not common in storage systems (fine grained monitoring and migration of storage buckets). ElastMan [4], uses two controllers in order to efficiently handle diurnal and spiky workloads but it requires offline manual training of both controllers. Lim et al. [2] on scaling Hadoop distributed file system (HDFS) adopts CPU utilization, which highly correlates request latency, for scaling but it relies on the data migration API integrated in HDFS. ProRenaTa [7] minimizes the SLO violation during scaling by combining both proactive and reactive control approaches but it requires a specific prediction algorithm and the control model needs to be trained offline. Hubbub-Scale [8] and Augment Scaling [35] argue that platform interference can mislead an elasticity controller during its decision making, however, the interference measurement needs the access of many low level metrics, e.g. cache counters, of the platform.

OnlineElastMan, on the other hand, focuses on the research of the practical aspects of an elasticity controller. It relies only on the most generic and obtainable metrics from the system and alleviates the burden of applying an elasticity controller in production. Specifically, the auto-training feature of OnlineElastMan makes its deployment, model training and configuration effortless. Furthermore, an generic and extendable prediction model is integrated to provide workload prediction for various workload patterns.

6.2.1 Elastic scaling

The goal of an auto-scaling system is to automatically fine-tune acquired resources of a system to minimize resource provisioning costs while meeting SLOs. An auto-scaling technique automatically scales resources according to demand. Different techniques exist in the literature that addresses the problem of auto-scaling. As a result of the wide diversity of these techniques, that are sometimes combination of two or more methods, it is a challenge to find a proper classification of auto-scaling techniques [36]. However, these techniques could be divided into two categories: reactive and proactive. In outline, reactive approach reacts to real time system changes such as incoming workload while a proactive approach relies on historical access patterns of a system to anticipate future needs so as to acquire or release resources in advance. Each of these approaches have its own merits and demerits [7]. Under the proactive and reactive categories, the following are some of the widely used auto-scaling techniques: threshold-based policies, reinforcement learning, queuing theory, control theory and time series analysis. Time series analysis is purely a proactive approach, whereas threshold-based rules (used in Amazon and RightScale) is a reactive approach. Contrary, reinforcement learning, queuing theory and control theory could be used with both proactive and reactive approaches, But they also exhibit the following demerits:

- Reinforcement learning: This technique is efficient when used against slowly changing conditions. Therefore, it cannot be applied to real applications that usually suffer from sudden traffic bursts. The elasticity controller presented in [37] integrates several empirical models and switches among them to obtain better predictions. Diego [38] presents an elasticity controller that uses analytical modeling and machine-learning. They explained that by combining both approaches, it results in better controller accuracy.
- Queuing theory: Impose hard assumptions that may not be valid for real, complex systems. They are intended for stationary scenarios, thus models need to be recalculated when conditions of the application change. For example, [39] model a cloud service using queuing theory. Using that model they build two adaptive proactive controllers that estimate the future load on a service.
- Control theory: Setting the gain parameters can be a difficult task. Previous works [4,40,41] have extensively studied applying control theory to achieve fine grained resource allocations that conform to a given SLO. However, the offline training feature of the existing approaches, makes the deployment, model training and configuration of the elasticity controller difficult.

In time series techniques, a given performance metric is sampled periodically at fixed intervals and analysed to make future predictions. Typically these techniques are utilized for workload or resource usage prediction and are used to derive a suitable scaling action plan. For example, [42] used a Fourier transform-based scheme to perform offline extraction of long-term cyclic workload patterns. CloudScale [43] and PRESS [22] perform long-term cyclic pattern extraction and resource demand prediction to scale up. The techniques used in these works complement our work.

6.2.2 Online profiling and prediction

A significant amount of literature exists that can be applied for predicting the traffic incident on a service i.e. [3,7,22,34,44]. In most cases, to support different workload scenarios, more than one prediction algorithms are used. To support different workload scenarios, at least more than one prediction algorithm is used. In most cases the pattern of the workload to be predicted is defined or known, which is not in our case. The most important aspect is how switching is carried out among the prediction algorithms which is not clear in most of these previous works. We therefore propose a simple weighted majority algorithm to handle this.

For instance, AGILE [30] provides online, wavelet-based medium-term (up to 2 min) resource demand prediction with adequate upfront time to start new application servers before performance degrades i.e. before application SLO is affected by the changing workload pattern. In addition, AGILE uses online profiling to obtain a resource pressure model for each application it controls. This model calculates the amount of resources required to maintain an applications SLO violation rate at a minimal level. Unlike our model, AGILE derives resource pressure models for just CPU without considering other resources such as memory, network bandwidth, disk I/O, applications workload intensity etc. A multi-resource model can be built in two ways. Each resource can have a separate resource pressure model or a single resource pressure model can represent all the resources. In this work, we adopt the latter approach.

In this paper, since we do not know the pattern of our workload, we have chosen some of the types of ARIMA models that are commonly encountered. For a time series that is stationary and autocorrelated, a possible model for it is a first-order autoregressive model. On the other hand, if the time series is not stationary, the simplest possible model for it is a random walk model. However, if the errors of a random walk model are autocorrelated, perhaps a differenced first-order autoregressive model may be more suitable. [45] presents a detailed explanation of these models.

7 Conclusions and future works

In this paper, we have designed, implemented and open-sourced[2] OnlineElastMan, which is an "out-of-the-box" elasticity controller for distributed storage systems. It includes a self-training multi-dimensional performance model to alleviate model training efforts and provide better provision accuracy, a self-tuning prediction module to adjust the prediction to various workload patterns, and an elasticity controller to calculate and carry out the scaling decisions by analyzing the inputs from the performance model and the prediction module. The evaluation results of OnlineElastMan on Cassandra show that OnlineElastMan is able to provision a Cassandra cluster efficiently and effectively with respect to the percentile latency SLO in the showcase experiment.

For future work, the OnlineElastMan framework can be extended in two directions. First, it would be useful to extend the control model of OnlineElastMan with comprehensive metrics, e.g., CPU utilization, network statistics, disk I/Os, etc. Second, OnlineElastMan is essentially stateless. States are only preserved and used in the prediction and model training modules, which can be generated/trained during runtime. Thus, it is not difficult to decentralize OnlineElastMan for better scalability and fault tolerance.

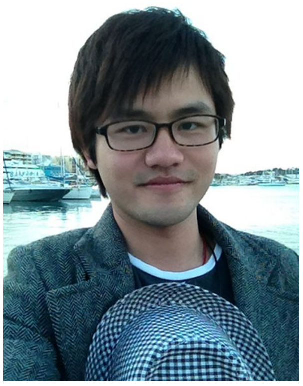
Acknowledgements This work was supported by the Erasmus Mundus Joint Doctorate in distributed computing program funded by the EACEA of the European Commission under FPA 2012-0030 and the End-to-End Clouds project funded by the Swedish Foundation for Strategic Research under the contract RIT10-0043. The authors would also like to thank the reviewers for their constructive comments and suggestions to improve the quality of the paper.

[2] https://github.com/gureya/OnlineElasticityManager.

References

1. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A review of auto-scaling techniques for elastic applications in cloud environments. J. Grid Comput. **12**(4), 559–592 (2014)
2. Harold, C. Lim, S.B., Jeffrey, S.C.: Automated control for elastic storage. In: Proceedings of the 7th International Conference on Autonomic Computing (ICAC '10), pp. 1–10. ACM, New York (2010)
3. Beth, T., Peter, B., Armando, F., Michael, J.F., Michael, I.J., David, A.P.: 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), pp. 12–12. USENIX Association, Berkeley, CA (2011)
4. Al-Shishtawy, A., Vlassov, V.: Elastman: autonomic elasticity manager for cloud-based key-value stores. In: Proceedings of the 22Nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '13), pp. 115–116. ACM, New York (2013)
5. Al-Shishtawy, A., Vlassov, V.: Elastman: elasticity manager for elastic key-value stores in the cloud. In: Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference (CAC '13), pp. 7:1–7:10. ACM, New York (2013)
6. Moulavi, M.A., Al-Shishtawy, A., Vlassov, V.: State-space feedback control for elastic distributed storage in a cloud environment. In: The 8th International Conference on Autonomic and Autonomous Systems (ICAS 2012), pp. 589–596 (2012)
7. Liu, Y., Rameshan, N., Monte, E., Vlassov, V., Navarro, L.: Prorenata: proactive and reactive tuning to scale a distributed storage system. In: 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 453–464 (2015)
8. Navarro, L., Vlassov, V., Rameshan, N., Liu, Y.: Hubbub-scale: towards reliable elastic scaling under multi-tenancy. In: 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) (2016)
9. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
10. Moriyoshi, O., Peter, N., Yohei, U., Kazuaki, I.: The data-centricity of web 2.0 workloads and its impact on server performance. In: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp 133–142 (2009)
11. Beth, T., Peter, B., Armando, F., Michael, J.F., Michael, I.J., David, A.P.: 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), pp. 12–12 (2011)
12. Ramakrishnan, R., Gehrke, J.: Database Management Systems, 2nd edn. Osborne/McGraw-Hill, Berkeley, CA (2000)
13. Roshan, S., Jay, K., Lei, G., Alex, F., Chinmay, S., Sam, S.: Serving large-scale batch computed data with project voldemort. In: The 10th USENIX Conference on File and Storage Technologies (FAST'12) (2012)
14. Giuseppe, D., Deniz, H., Madan, J., Gunavardhan, K., Avinash, L., Alex, P., Swaminathan, S., Peter, V., Werner, V.: Dynamo: amazon's highly available key-value store. In: Proceedings of

Twenty-First ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07), pp. 205–220. ACM, New York (2007)
15. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. SIGOPS Oper. Syst. Rev. **44**(2), 35–40 (2010)
16. Animoto's Facebook scale-up. http://blog.rightscale.com/2008/04/23/animoto-facebook-scale-up/ (2012)
17. Peter, B., Armando, F., Michael, J.F., Michael, I.J., David A.P.: Characterizing, modeling, and generating workload spikes for stateful services. In: Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10), pp. 241–252 (2010)
18. Joseph, L., Hellerstein, Y.D., Sujay, P., Dawn, M.T.: *Feedback Control of Computing Systems*. Wiley, New York (2004)
19. Horn, P.: Autonomic computing: IBM's perspective on the state of information technology. October 15 (2001)
20. Tickoo, O., Iyer, R., Illikkal, R., Newell, D.: Modeling virtual machine performance: challenges and approaches. SIGMETRICS Perform. Eval. Rev. **37**(3), 55–60 (2010)
21. Iyer, R., Illikkal, R., Tickoo, O., Zhao, L., Apparao, P., Newell, D.: VM3: measuring, modeling and managing VM shared resources. Comput. Netw. **53**(17), 2873–2887 (2009)
22. Zhenhuan, G., Xiaohui, G., Wilkes, J.: Press: predictive elastic resource scaling for cloud systems. In: International Conference on Network and Service Management (CNSM), pp. 9–16 (2010)
23. Vasić, N., Novaković, D., Miučin, S., Kostić, D., Bianchini, R.: Dejavu: accelerating resource allocation in virtualized environments. ACM SIGARCH Comput. Archit. News **40**(1), 423–436 (2012)
24. Novakovic, D., Vasic, N., Novakovic, S., Kostic, D., Bianchini, R.: Transparently identifying and managing performance interference in virtualized environments. Technical report, Deepdive (2013)
25. Gunn, S.R.: Support vector machines for classification and regression. Technical report, University of Southampton (1998)
26. Arlitt, M., Jin, T.: A workload characterization study of the 1998 world cup web site. IEEE Netw. **14**(3), 30–37 (2000)
27. Gusella, R.: Characterizing the variability of arrival processes with indexes of dispersion. IEEE J. Sel. Areas Commun. **9**(2), 203–211 (1991)
28. Box, G.E.P., Jenkins, G.: Time Series Analysis, Forecasting and Control. Holden-Day, Incorporated (1990)
29. Littlestone, N., Warmuth, M.K.: The weighted majority algorithm. Inf. Comput. **108**(2), 212–261 (1994)
30. Nguyen, H., Shen, Z., Gu,X., Subbiah, S., Wilkes, J.: AGILE: elastic distributed resource scaling for infrastructure-as-a-service. In: Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13), pp. 69–82. USENIX, San Jose (2013)
31. Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/
32. Right Scale. http://www.rightscale.com/
33. Google Compute Engine. https://cloud.google.com/compute/docs/load-balancing-and-autoscaling
34. Yuan, D., Joshi, N., Jacobson, D., Oberai, P.: Scryer: Netflix's Predictive Auto Scaling Engine. http://techblog.netflix.com/2013/12/scryer-netflixs-predictive-auto-scaling.html. Accessed June 2015
35. Navarro, L., Vlassov, V., Rameshan, N., Liu, Y.: Augmenting elasticity controllers for improved accuracy. In: 13rd IEEE International Conference on Autonomic Computing (ICAC) (2016)
36. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A review of auto-scaling techniques for elastic applications in cloud environments. J. Grid Comput. **12**(4), 559–592 (2014)
37. Malkowski, S.J., Hedwig, M., Li, J., Pu, C., Neumann, D.: Automated control for elastic n-tier workloads based on empirical modeling. In: Proceedings of the 8th ACM International Conference on Autonomic Computing, (ICAC '11), pp. 131–140. ACM, New York (2011)
38. Didona, D., Romano, P., Peluso, S., Quaglia, F.: Transactional auto scaler: elastic scaling of in-memory transactional data grids. In: Proceedings of the 9th International Conference on Autonomic Computing (ICAC '12), pp. 125–134. ACM, New York (2012)
39. Ali-Eldin, A., Tordsson, J., Elmroth, E.: An adaptive hybrid elasticity controller for cloud infrastructures. In: The 13th IEEE/IFIP Network Operations and Management Symposium (NOMS'12). Hawaii (2012)
40. Zhu, X., Young, D., Watson, B.J., Wang, Z., Rolia, J., Singhal, S., Hyser, C., Gmach, D., Gardner, R., Christian, T., Cherkasova, L., et al.: 1000 islands: Integrated capacity and workload management 17 the next generation data center. In: In Proceedings of the 5th International Conference on Autonomic Computing (ICAC), pp. 172–181
41. Lim, H.C., Babu, S., Chase, J.S.: Automated control for elastic storage. In: Proceedings of the 7th International Conference on Autonomic Computing (ICAC '10), pp. 1–10. New York (2010)
42. Cherkasova, L., Gmach, D., Rolia, J., Kemper, A.: Capacity management and demand prediction for next generation data center. In: IEEE International Conference on Web Services (ICWS), pp. 43–50. (2007)
43. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: Cloudscale: elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the 2Nd ACM Symposium on Cloud Computing (SOCC '11), pp. 5:1–5:14. ACM, New York (2011)
44. Roy, N., Dubey, A., Gokhale, A.: Efficient autoscaling in the cloud using predictive models for workload forecasting. In: IEEE International Conference on Cloud Computing (CLOUD), pp. 500–507 (2011)
45. Nau, R.: Statistical forecasting: notes on regression and time series analysis. http://people.duke.edu/~rnau/411home.htm. Accessed June 2015



Ying Liu is a post-doc researcher focusing on deep learning at Stockholm University. He received his Ph.D. from KTH Royal Institute of Technology, Stockholm, Sweden, in October 2016. Earlier, He completed his double master degrees at Royal Institute of Technology, Stockholm, Sweden and Universitat Politècnica de Catalunya, Barcelona, Spain. His research interests include, Machine Learning, Big Data Analytics, Large-Scale Distributed Systems and Algorithms, Distributed Storage Systems, Clouds, Autonomic Computing, Elasticity Management, and Auto-Scaling.

Daharewa Gureya is a Ph.D. Researcher at INESC-ID Lisboa/Instituto Superior Técnico. He holds a Master of Science degree (Erasmus Mundus European Master in Distributed Computing, a double degree master program) from Instituto Superior Técnico, Lisbon, Portugal, and KTH Royal Institute of Technology, Stockholm, Sweden. He also holds a Bachelor of Science (Hons) degree, majoring Computer Science, with a first class from the University of Nairobi, Kenya. His current research focus is performance issues in data-intensive applications on Multi-core Architectures.

Ahmad Al-Shishtawy is a senior researcher in the Computer Systems Laboratory (CSL) at the Swedish Institute of Computer Science (SICS). He received his Ph.D. from KTH Royal Institute of Technology, Stockholm, Sweden, in November 2012. Earlier, He completed his B.Sc. and M.Sc. at the Faculty of Computer and Information Sciences (FCIS), Ain Shams University, Cairo, Egypt. His research interests include, Large-Scale Distributed Systems and Algorithms; Big Data Analytics, Streaming Analytics, and Data Intensive Computing; Clouds, Multi-Clouds, Edge-Computing, and P2P Systems; Autonomic Computing, Elasticity Management, and Auto-Scaling.

Vladimir Vlassov is an Associate Professor of Computer Systems at the Department of Software and Computer Systems, School of Information and Communication Technology, KTH Royal Institute of Technology in Stockholm, Sweden. He worked as a visiting scientist and researcher at UMASS (2004) and at MIT, USA (1998). He has participated in a number of European projects, projects funded by Swedish funding agencies and a project funded by NSF USA. His research interests include data intensive computing and big-data analytics, autonomic computing, distributed and parallel computing. His current research focus is on data-intensive computing and stream processing; Cloud resource management; self-management of cloud-based services and applications.