# Efficient Provisioning of Bursty Scientific Workloads on the Cloud Using Adaptive Elasticity Control

Ahmed Ali-Eldin
Dept. of Computing Science
Umeå University, Sweden
ahmeda@cs.umu.se

Maria Kihl
Dept. of Electrical and
Information Technology,
Lund University
Maria.Kihl@eit.lth.se

Johan Tordsson
Dept. of Computing Science
Umeå University, Sweden
tordsson@cs.umu.se

Erik Elmroth
Dept. of Computing Science
Umeå University, Sweden
elmroth@cs.umu.se

## ABSTRACT

Elasticity is the ability of a cloud infrastructure to dynamically change the amount of resources allocated to a running service as load changes. We build an autonomous elasticity controller that changes the number of virtual machines allocated to a service based on both monitored load changes and predictions of future load. The cloud infrastructure is modeled as a $G/G/N$ queue. This model is used to construct a hybrid reactive-adaptive controller that quickly reacts to sudden load changes, prevents premature release of resources, takes into account the heterogeneity of the workload, and avoids oscillations. Using simulations with Web and cluster workload traces, we show that our proposed controller lowers the number of delayed requests by a factor of 70 for the Web traces and 3 for the cluster traces when compared to a reactive controller. Our controller also decreases the average number of queued requests by a factor of 3 for both traces, and reduces oscillations by a factor of 7 for the Web traces and 3 for the cluster traces. This comes at the expense of between 20% and 30% over-provisioning, as compared to a few percent for the reactive controller.

## Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Performance of systems**]

## General Terms

Algorithms, Performance, Reliability

## Keywords

Cloud computing, Elasticity, Proportional Control

*ScienceCloud'12,* June 18, 2012, Delft, The Netherlands.
Copyright 2012 ACM 978-1-4503-1340-7/12/06 ...$10.00.

## 1. INTRODUCTION

Elasticity of the cloud infrastructure is the ability of the infrastructure to allocate resources to a service based on the running load as fast as possible. An *elasticity controller* aims to allocate enough resources to a running service while at the same time avoiding costly over-provisioning. The problem for an elasticity controller is thus to decide when, and how much, to scale up or down. Scaling can be done either horizontally, by increasing or decreasing the number of Virtual Machines (VMs) allocated, or vertically, by changing the hardware configuration for CPU, memory, etc. of already running VMs. The resources allocated to a service can vary between a handful of VMs to tens of thousands of VMs depending on the load requirements. Most Infrastructure as a Service (IaaS) providers does not host a single service but rather quite a few scalable services and applications. Given the scale of the current and future cloud datacenters and services, these are impossible to manage manually, making autonomic management a key issue for clouds.

Recently, the scientific computing community started discussing the potential use of cloud computing infrastructures to run scientific experiments such as medical NLP processing [6] and workflows for astronomical data released by the Kepler project [25]. Most of the applications are embarrassingly parallel [11]. There are some limitations to the wide adoption of the cloud paradigm for scientific computing as identified by Truong et al. [23] such as the lack of cost evaluation tools, cluster machine images and, as addressed in this paper, autonomic elasticity control.

There are many approaches to solve the elasticity problem [5, 7, 10, 17, 18, 20, 24, 27, 28], each with its own strengths and weaknesses. Desired properties of an elasticity controller include the following:

- Fast: The time required by the controller to make a decision is a key factor for successful control, for example, limited look-ahead control is shown to have superior accuracy but requires 30 minutes to control 60 VMs on 15 physical servers [14].
- Scalable: The controller should be scalable with respect to the number of VMs allocated to a service and with respect to the time of running the algorithm. There are many techniques that can be used for esti-

mation of the load and elasticity control which are not scalable with either time or scale e.g., regression based control is not scalable with respect to the algorithm execution time [1].

- Adaptive: Scientific workloads and Internet traffic are very dynamic in nature [2, 15]. Elasticity controllers should have a proactive component that predicts the future load to be able to provision resources a priori. Most prediction techniques such as neural networks build a model for the load in order to predict the future. Another desired property of an adaptive controller is the ability to change the model whenever the load dynamics change.
- Robust and reliable: The changing load dynamics might lead to a change in the controller behavior [9, 19]. A controller should be robust against changing load dynamics. A robust controller should prevent oscillations in resource allocation i.e., the controller should not release resources prematurely. A reactive controller (step controller) is a controller that only allocates new VMs to a service when the load increases and deallocates the VMs once the load decreases beyond a certain level. This type of controller thus reduces the number of VMs provisioned and minimizes the provisioning costs, at the expense of oscillations.

Our previous work [1] studies different ways to combine reactive and proactive control approaches for horizontal elasticity. The two simple hybrid controllers proposed combine reactive scaling up with proactive scale-down. These controllers act on the monitored and predicted service load, but ignore multiple important aspects of infrastructure performance and service workload. In this paper, our previous work is extended by an enhanced system model and controller design. The new controller takes into account the VM startup time, workload heterogeneity, and the changing request service rate of a VM. It thus controls the allocated capacity instead of only the service load. The controller design is further improved by adding a buffer to the controller to store any delayed requests for future processing. This buffer model characterizes many scientific workloads where jobs are usually queued for future processing. The proposed controller can be used by both the cloud service provider and the cloud user to reduce the cost of operations and the cost of running a service or an experiment in the cloud. The controller can be used also to control the elasticity of a privately run cloud or cluster.

The performance of the controller is tested using two sets of traces, a Web workload from the FIFA world cup [3] and a recently published workload from a Google cluster composed of around 11 thousand machines [26]. The Web trace is selected as it is a well known and rather bursty workload and thus challenging for an elasticity controller. The cluster traces, consisting mostly of MapReduce jobs, are chosen to evaluate the behavior of our approach on traces more similar to scientific workloads.

The rest of this paper is organized as follows. Section 2 describes the system model and the design of the proposed controller. In Section 3, the simulation framework and the experiments are described and the results are discussed. Section 4 discusses some of the different approaches available in the literature for building elasticity controllers. Section 5 contains the conclusions.

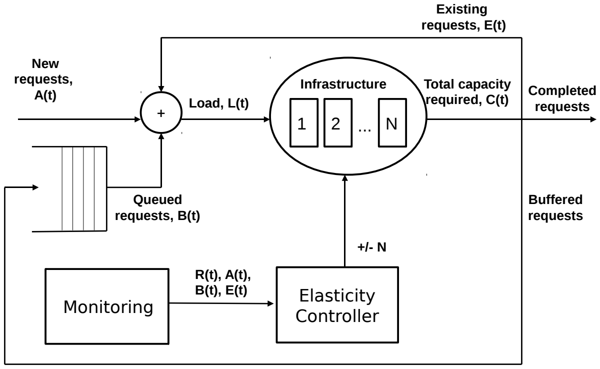


**Figure 1: Queueing model and elasticity control for a cloud service.**

# 2. CONTROLLER DESIGN

## 2.1 System model

In this work, the cloud infrastructure is modeled as a closed loop control system and queueing models are used to design a feedback elasticity controller. The cloud infrastructure is modeled as a $G/G/N$ stable queue in which the number of servers $N$ required is variable [16] as shown in Figure 1. This is a generalization of the work by Khazaei et al. [13] where a cloud is modeled as an $M/G/m$ queue with a constant number of servers, $m$. We assume that the system serves generic requests that can be anything from a Web query to Pubmed [12] to a job in a workflow to process astronomical data [25].

The number of VMs allocated to a service at any time unit, $N$, changes according to the controller output. When the load increases, VMs are added and when it decreases VMs are removed. We assume that it takes one time unit for a VM to boot and get intialized. In practice, it also takes time to shut-down a VM, but for most applications, no more requests are sent to a VM after a shutdown command is issued. It is thus assumed that the effect of VM shut down on capacity is instantaneous.

In our model, requests not served are buffered and delayed as shown in Figure 1. We make no assumptions about a finite buffer size, but the designed controller uses the number of buffer requests as one criteria for adding VMs. The buffer length is also used as a performance metric in the evaluation section. A buffered request is delayed and thus the larger the number of buffered requests, the slower the request response time. Assuming the queue is stable, the average service rate over time is equal to the average arrival rate over time. Whenever the system is at risk of instability due to increase in the demand, the elasticity controller increases $N$ to enforce system stability. The elasticity control problem can be stated as follows: the elasticity controller should add or remove VMs to ensure system stability, i.e., over a long period of time, the number of serviced requests (the service capacity) is equal to the total number of received requests received with an error tolerance (number of

**Table 1: Overview of used notation.**

| Variable | Description |
|---|---|
| $N$ | Number of VMs deployed |
| $L(t)$ | Total service load at time t |
| $R(t)$ | Total service capacity available at time $t$ |
| $C(t)$ | Service capacity required at time $t$ |
| $A(t)$ | Arriving (new) requests at time $t$ |
| $D(t)$ | Increase/decrease in required capacity at time $t$ |
| $B(t)$ | Size of buffer at time $t$ |
| $E(t)$ | Amount of already processing requests at time $t$ |
| $K$ | Number of queued requests before starting a new VM |
| $r$ | Number of time units needed to start all buffered requests |
| $T_d$ | Estimation interval (time between two estimations) |
| $\overline{L_{T_d}}$ | Average load over the last estimation interval |
| $\overline{L_t}$ | Average load over all time |
| $\tilde{D}$ | Predicted value for request change rates over next $T_d$ time units |
| $P$ | Estimated ratio between $\tilde{D}$ and average load |
| $M_{Avg}$ | The average of the median request service rates per unit time over the the $T_d$ |

buffered requests). This should be achieved irrespective of the change in the request arrival rate and while maintaining the number of VMs to a minimum.

## 2.2 Estimating future usage

The optimal total service capacity, $C(t)$, required for time $t$ is:

$$C(t) = C(t-1) + D(t), \qquad (1)$$

where $C(t-1)$ is the capacity required in the last time step and $D(t)$ is the increase or decrease in capacity needed in order to meet SLAs while maintaining the number of VMs to a minimum. The controller is activated each $T_d$ time units. When evoked at time $t$, it estimates the change in workload for the next $T_d$ time units, $D(t+1), D(t+2), \ldots, D(t+T_d)$. VM allocations are adjusted at times $t+1, t+2, \ldots, t+T_d$ according to these predictions, followed by a new prediction for $t + T_d \ldots t + 2T_d$. We define $A(t)$ as the arrival rate of new requests to the service. A suitable initial service configuration could be $C(0) = A(0)$.

We define the total workload of the service, $L(t)$, as the sum of the arriving requests, the existing, already processing requests, $E(t)$, and any buffered requests to be served. No assumptions are thus made about the time needed to serve a request, which can vary from seconds to hours. We use $B(t)$ to denote the number of requests in the buffer at time $t$. If enough VMs are allocated to initialize all buffered requests in the next time unit, these machines may become idle and be released shortly after, causing oscillations in resource allocations. We thus define $r$, a system parameter specifying over how many time units the currently buffered load should be started. Now, the total workload at time $t$ can be written as:

$$L(t) = A(t) + E(t) + \frac{B(t)}{r}. \qquad (2)$$

The capacity change required can be written as

$$D(t) = L(t) - R(t) \qquad (3)$$

where $R(t)$ denotes the currently allocated capacity. Assuming that $A(t)$ remains constant for the next time unit, the estimated change in the current service capacity required, $\tilde{D}$ for the future $T_d$ time units can be estimated by

$$\tilde{D} = P\overline{L_{T_d}} \qquad (4)$$

where $P$ represents the estimated rate of adding or removing VMs. We define $\overline{L_{T_d}}$ to be the average periodical service load over the past $T_d$ time units,

$$\overline{L_{T_d}} = \frac{\sum_{i=0}^{T_d} L(t-i)}{T_d}. \qquad (5)$$

Similarly, we define $\overline{L_t}$, as the average load over all time as follows:

$$\overline{L_t} = \frac{\sum_{i=0}^{t} L(i)}{t}. \qquad (6)$$

Now, $P$ represents the estimated ratio of the average change in the load to the average load over the next $T_d$ time units and $\overline{L_{T_d}}$ is the estimated average capacity required to keep the buffer size stable for the next $T_d$ time units. $P$ is positive if there is an increase in total workload (new service requests, buffered requests, and requests that need to be processed longer); negative if this sum decreases (with the buffer empty); and zero if the system is at a steady state and the buffer is empty.

We define $P$ to be the ratio between $D(t)$ and the average system load over time,

$$P = \frac{D(t)}{\overline{L_t}}. \qquad (7)$$

This value represents the change in the load with respect to the average capacity. By substituting Equations 7 in Equation 4,

$$\tilde{D} = \frac{\overline{L_{T_d}}}{\overline{L_t}} D(t). \qquad (8)$$

This formulation is a proportional controller [21] where $D(t)$ is the error signal and $\overline{L_{T_d}}/\overline{L_t}$, the normalized service capacity, is the gain parameter of the controller. By substituting Equation 5 in Equation 8, we obtain

$$\tilde{D} = \frac{\sum_{i=0}^{T_d} L(t-i)}{\overline{L_t}} \frac{D(t)}{T_d}. \qquad (9)$$

If $T_d$ is optimal, i.e., estimations occur when the rate of change of the load changes, then $D(t)/T_d$ is the slope of the changing load multiplied by the ratio between the instantaneous load and the overtime average load.

## 2.3 Determining suitable estimation intervals

The interval between two estimations, $T_d$, is a crucial parameter affecting the controller performance. It is used to calculate $P$ and $\tilde{D}$ and $T_d$ also controls the reactivity of the controller. If $T_d$ is set to one, the controller performs predictions every time unit. At the other extreme, if $T_d$ is set to $\infty$, the controller performs no predictions at all. As the workloads observed in datacenters are dynamic [2], setting an adaptive value for $T_d$ that changes according to the load dynamics is important.

We define the maximum number of buffered requests, $K$, as the tolerance level of a service i.e., the maximum number of requests queued before making a new estimation or adding a new VM, thus:

$$T_d = \begin{cases} K/|\tilde{D}| & \text{if K>0 and } |\tilde{D}| \neq 0 \\ 1 & \text{if K=0 or } \tilde{D} = 0 \end{cases} \tag{10}$$

The value of $K$ can be used to model SLAs with availability guarantees. A low value for $K$ provides quicker reaction to load changes, but will also result in oscillations as resources will be provisioned and released based on the last few time units only. Conversely, $K$ is large, the system reacts slowly to changing load dynamics. Similarly, $r$ affects the rate with which buffered requests should be started, and thus impose similar tradeoffs between oscillations and quickly reacting to load increases.

## 2.4 Hybrid elasticity control

The main goal of an elasticity controller is to allocate enough resources to enforce the SLAs while decreasing total resource usage. It is very hard to anticipate whether an observed increase in load will continue to rise to a large peak [2] as there are no perfect estimators or controllers. Using pure estimation for scale-up decisions is dangerous as it can lead to system instability and oscillations if the load dynamics change suddenly while the controller model is based on the previous load.

```
Data: $r$, $K$
Result: Perform resource (de)allocation to keep the
        system stable
1  $Proactive\_Aggregator \leftarrow 0$;
2  $T_d \leftarrow 1$;
3  for each time step $t$ do
4      Update $R(t), A(t), B(t)$, and $E(t)$ from monitoring
       data;
5      Calculate $D(t)$ using Equation 3;
6      if Time from last estimation $\geq T_d$ then
7          Calculate $\overline{L_{T_d}}$ from Equation 5;
8          Calculate $\overline{L_t}$ from Equation 6;
9          Calculate $P$ from Equation 7;
10         Calculate $\tilde{D}$ from Equation 8;
11         Update $M_{Avg}$;
12         Calculate $T_d$ from Equation 10;
13     $N_{Reactive} \leftarrow \lceil D(t)/M_{Avg} \rceil$;
14     $Proactive\_Aggregator += \tilde{D}/M_{Avg}$;
15     $N_{Proactive} \leftarrow \lfloor Proactive\_Aggregator \rfloor$;
16     $Proactive\_Aggregator -= N_{Proactive}$;
17     if $N_{Reactive} > K$ then
18         if $N_{Proactive} > 0$ then
19             Deploy $N_{Proactive} + N_{Reactive}$ servers
20         else
21             Deploy $N_{Reactive}$ servers
22     else
23         (Un)deploy $N_{Proactive}$ servers
```

**Algorithm 1:** Hybrid elasticity controller with both proactive and reactive components.

Our design is a hybrid controller where a reactive component is coupled with the proposed proactive component for scale up and a proactive only component for scale down. The details of the implementation are given in Algorithm 1. The controller starts by receiving monitoring data from the monitoring subsystem in Line 4. If $T_d$ time units passed since the last estimation, the system model is updated by reestimating $\tilde{D}$ and $T_d$ as shown from Line 6 to Line 12. The unit of $\tilde{D}$ is requests per time unit.

The actual calculation of the number of VMs to be added or removed by the different controllers is done between lines 13 and 16. In some applications, $\tilde{D}$ is divided by $M_{Avg}$ in Line 14 to find the number of servers required. The rate of the proactive controller can be steered by multiplying $\tilde{D}$ by a factor, e.g., to adding double the estimated VMs for some critical applications. The reactive controller is coupled with the proactive controller to reach a unified decision as shown from Line 17 to Line 22. For scale up decisions, when the decisions of both the reactive component and the proactive component are to scale up, the decisions are added. For example, if the reactive component decides that two more VMs are required while the proactive component decides that three VMs are needed, five VMs are added. The reactive component is reacting for the current load while the proactive component is estimating the future load based on the past load. If the reactive component decides that a scale up is needed while the proactive decides that a scale down is needed then the decision of the reactive component alone is performed because the reactive component's decision is based on the current load while the proactive component's decision may be based on a skewed model that needs to be changed.

For the reactive and proactive components to calculate the number of VMs required for a given load, the controller needs to know the service capacity of a VM i.e., the number of requests serviced per VM every time unit. This number is variable as the performance of a VM is not constant.

In addition, there are different request types and each request takes different time to service. As a solution, we calculate $M_{Avg}$, the average of the median request service rates per VM per unit time over the past estimation period $T_d$. $M_{Avg}$ is used by the reactive and proactive components to calculate the number of VMs required per unit time to service all the requests while meeting the different SLAs. The median is chosen as it is a simple and efficient statistical measure which is robust to outliers and skewed distributions. No assumptions are made about the service rate distribution for a VM. $M_{Avg}$ is a configurable parameter which can be changed based on deployment requirements.

# 3. EXPERIMENTAL EVALUATION

To validate the controller, a three-phase discrete-event simulator [4] was built using python that models a service deployed in the cloud. Two different workloads are used for the evaluation, the complete traces from the FIFA 1998 world cup [3] and a set of Google cluster traces [26].

In all evaluations, the controller time step, i.e., the time it takes to start a VM is selected to be 1 minute, which is a reasonable assumption [22]. The effects of the controller's decision to increase the resources provisioned does thus not appear until after one minute has elapsed and the new VMs are running. The granularity of the monitoring data used by the controller is also 1 minute.

The controller's performance is compared to a completely reactive controller similar to the one proposed by Chieu et al. [8]. The design of the reactive controller is shown in

```
Data: $r$, $K$
Result: Perform resource (de)allocation to keep the system stable
for each time step $t$ do
    Update $R(t), A(t), B(t)$, and $E(t)$ from monitoring data;
    Calculate $M_{Avg}$;
    Calculate $D(t)$ using Equation 3;
    $N_{Reactive} \leftarrow \lceil D(t)/M_{Avg} \rceil$;
    if $N_{Reactive} > 0$ and $D(t) > K$ then
        Deploy $N_{Reactive}$ servers
    if $N_{Reactive} < -2$ then
        Undeploy $N_{Reactive}$ servers
```

**Algorithm 2:** Reactive elasticity controller.

Algorithm 2. The calculation of the median service rate is done every minute as this is the time between two load estimations. In order to reduce oscillations in the reactive controller, scale down is not done until the capacity change $D(t)$ is less than the current provisioned capacity $C(t)$ by $2M_{Avg}$, i.e., scale down is only done when there are more than two extra VMs provisioned. In the experiments, we refer to our controller in Algorithm 1 as $C_{\text{Hybrid}}$ and the reactive controller in Algorithm 2 as $C_{\text{Reactive}}$.

With a few exceptions, most of the work available on elasticity control compares performance with static provisioning. However, we chose to compare our controller with a reactive controller to highlight the tradeoffs between over-provisioning, SLA violations, and oscillations.

## 3.1 Performance Metrics

Different metrics can be used to quantify the controller performance. We define $OP$ to be the number of over-provisioned VMs by the controller per time unit aggregated over the whole trace. $\overline{OP}$ is the average number of over-provisioned VMs by the controller per minute. Similarly, $UP$ and $\overline{UP}$ are the aggregate and the average number of under-provisioned VMs by the controller per minute. We also define $V$, the average number of servers required to service the buffered load per minute,

$$V = \Sigma \frac{\text{Buffered Load}}{\text{Median Service rate of a VM}}. \qquad (11)$$

$V$ represents the way the buffers get loaded. It does not represent the optimal number of servers required to service the load but rather represents average required number of VMs due to the queue build up. We use $\overline{N}$ to denote the average number of VMs deployed over time.

## 3.2 Web workload performance evaluation

The Web workload contains 1.3 billion Web requests recorded at servers for the 1998 FIFA world cup in the period between April 30, 1998 and July 26, 1998. The aggregate number of requests per second were calculated from these traces. In the simulation, the requests are grouped by time of arrival. The focus is not on individual requests but rather on the macro-system performance. For the experiments, the average service rate of a VM is drawn from a Poisson distribution with an average equal to $\lambda$ requests per second. It is assumed that the time required to process one request is 1 second. The tolerance level $K$ is chosen to be 5, i.e., 5 requests may be buffered before the controller reacts.

Assuming perfect load balancing, the performance of the controller is the only factor affecting performance in the simulation. We set $r$ to 60 seconds, i.e., queued requests are emptied over one minute.

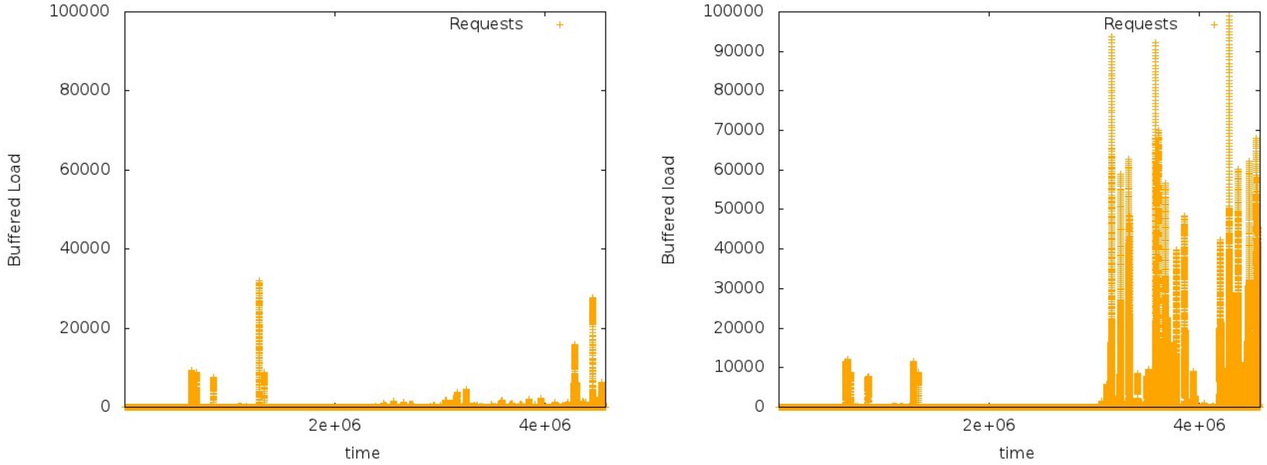
The controller is configured for the worst case scenario by using the maximum load recorded during the past minute as the input request arrival rate to the controller. This assumption can be relaxed by monitoring the average or median load for the past minute instead. Using the median or the average will result in provisioning less resources for most workloads.

As the Web traces are quite old, we have multiplied the number of requests by a factor $F$ in order to stress test the controller performance under different load dynamics. For different experiments, $\lambda$ is also changed. Table 2 shows the performance of the two controllers when $F$ takes the values of 1, 10, 20, 30, and 40 while $\lambda$ takes the values of 100, 200, 300, and 400. Due to the size of the trace, the aggregate metrics $UP$ and $OP$ are quite large.

For the over-provisioning metrics, $OP$ and $\overline{OP}$, it is clear that $C_{\text{Hybrid}}$ has a higher over-provisioning rate compared to $C_{\text{Reactive}}$. This is intuitive because $C_{\text{Hybrid}}$ provisions resources ahead in time to be used in the future and delays the release of resources in order to decrease oscillations. When $\lambda$ changes such that the ratio between $\lambda$ and $F$ is constant at 10, $OP$ and $\overline{OP}$ are reduced for both controllers compared to when only $F$ increases and the rate of increase of both values is lower. Notably, these values are quite small if we compare them to static provisioning. If capacity would be statically provisioned for the workload, 42 servers would be needed when $F = 1$ and $\lambda = 100$, whereas using the proactive controller or even the reactive controller, the average number of provisioned servers is around 3, reducing resource use by around 92.5% compared to static provisioning but at the cost of an increase in the number of delayed requests.

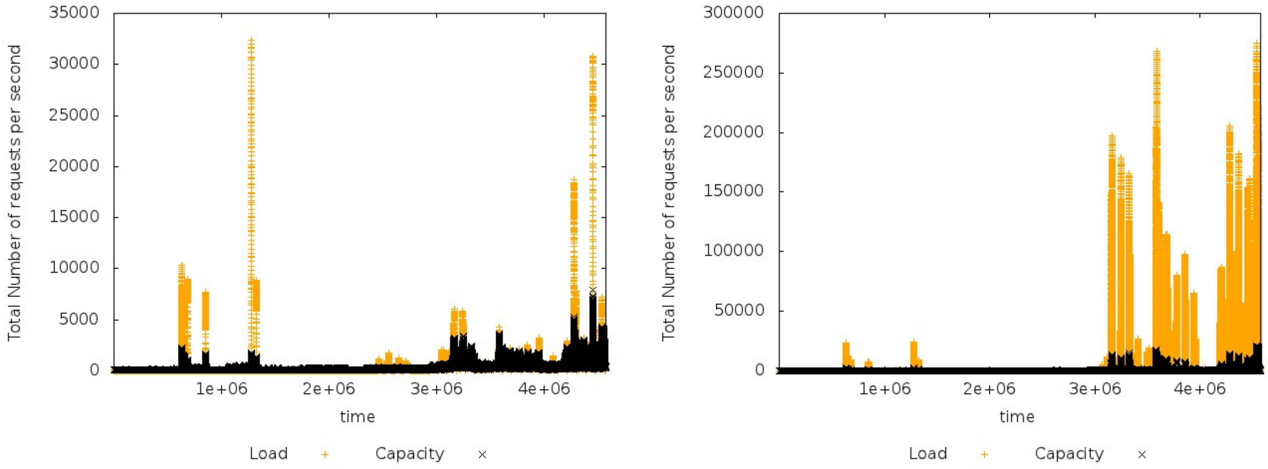
Looking at the aggregate and the average under-provisioning metrics, $UP$ and $\overline{UP}$, $C_{\text{Hybrid}}$ is superior to $C_{\text{Reactive}}$. Since $C_{\text{Hybrid}}$ scales up and down proactively, it prevents oscillations. It proactively allocates VMs to and does not release resources prematurely, thus decreasing the amount of buffered and delayed requests. In fact, $C_{\text{Reactive}}$ shows a very high rate of under-provisioning due to the fact that all delayed requests are buffered. The average number of under-provisioned servers of $C_{\text{Reactive}}$ is 70 times that of $C_{\text{Proactive}}$ when $F = 1$ and $\lambda = 100$. Since $C_{\text{Reactive}}$ is always lagging the load and releases resources prematurely causing oscillations, the performance of the controller is quite bad. In real life, the buffer is not infinite and thus requests are dropped. In comparison, using the proposed controller, $C_{\text{Hybrid}}$, the request drop rate is quite low. Again, we note that for a ratio between $\lambda$ and $F$ of 10, under-provisioning ($UP$ and $\overline{UP}$) is reduced for both controllers compared to when only $F$ increases. Actually, for $C_{\text{Hybrid}}$, $UP$ and $\overline{UP}$, are almost constant while for $C_{\text{Reactive}}$, they decrease significantly with the increase of $\lambda$ and $F$ while having a constant internal ratio. We attribute this to the higher service capacity of the VMs allowing the system to cope with higher system dynamics. For future work, we plan to investigate the effect of the VM capacity on the performance of an elasticity controller with different system dynamics and if the property we have just noted is general for any arrival process.

The $V$ columns in Table 2 show the average number of



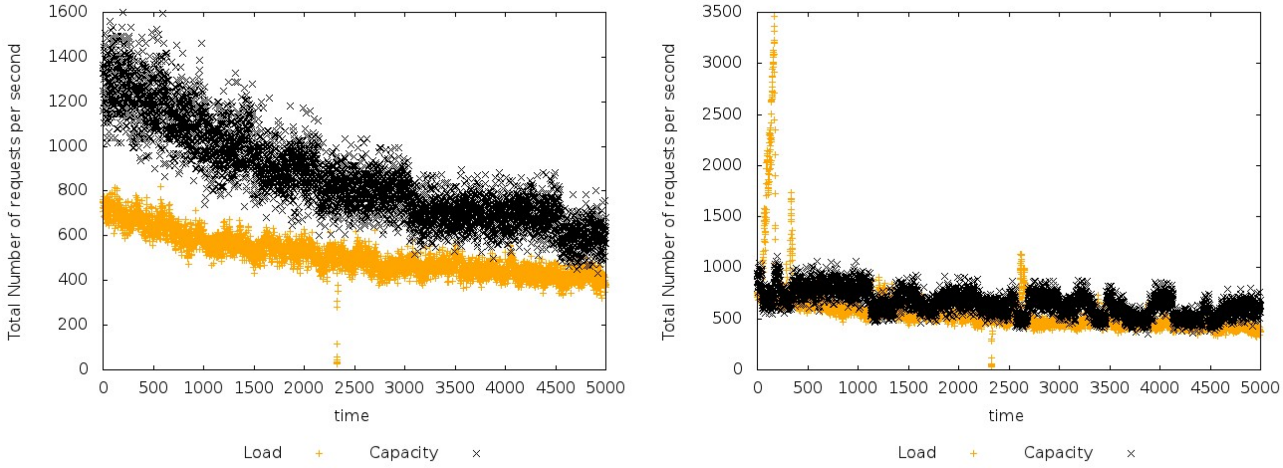
(a) $C_{Proactive}$ (b) $C_{\text{Reactive}}$

**Figure 2: Number of buffered requests over time for the Web workload.**



(a) $C_{\text{Hybrid}}$ (b) $C_{\text{Reactive}}$

**Figure 3: Load and the provisioned capacity over time for the Web workload.**



(a) $C_{\text{Hybrid}}$ (b) $C_{\text{Reactive}}$

**Figure 4: Load and the provisioned capacity for 1.5 hours of the Web workload.**

**Table 2: Web workload performance overview.**

| $C_{\text{Hybrid}}$ results | | | | | | | | $C_{\text{Reactive}}$ results | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $F$ | $\lambda$ | $OP$ | $\overline{OP}$ | $UP$ | $\overline{UP}$ | $V$ | $\overline{N}$ | $OP$ | $\overline{OP}$ | $UP$ | $\overline{UP}$ | $V$ | $\overline{N}$ |
| 1 | 100 | 41905 | 0.548 | 3883 | 0.05 | 2.5 | 3 | 35600 | 0.47 | 267402 | 3.49 | 5.98 | 2.95 |
| 10 | 100 | 535436 | 6.99 | 8315 | 0.1 | 19.7 | 26.09 | 206697 | 2.7 | 8835898 | 115.45 | 135.97 | 23.28 |
| 20 | 100 | 1075447 | 14.05 | 98678 | 1.29 | 38.9 | 51.76 | 380059 | 4.966 | 19611571 | 256.26 | 297.29 | 46.14 |
| 30 | 100 | 1617452 | 21.14 | 148896 | 1.9 | 58.1 | 77.46 | 555503 | 7.25 | 30944637 | 404.35 | 466 | 69.15 |
| 40 | 100 | 2155408 | 28.16 | 197660 | 2.58 | 77.3 | 103.11 | 732157 | 9.567 | 42265699 | 552.28 | 634.57 | 92.14 |
| 20 | 200 | 654596 | 8.55 | 35380 | 0.46 | 19.3 | 27.57 | 225979 | 2.95 | 5187614 | 67.78 | 87.63 | 22.86 |
| 30 | 300 | 761956 | 9.96 | 30951.0 | 0.4 | 19.3 | 28.94 | 235436 | 3.07 | 3783052 | 49.4 | 69 | 22.71 |
| 40 | 400 | 857608 | 11.2 | 30512 | 0.4 | 19.3 | 30.16 | 241854 | 3.16 | 3180898 | 41.56 | 61.04 | 22.7 |

**Table 3: Number of VMs added and removed for the Web workload with $F = 1$ and $\lambda = 100$.**

| | $X_R$ | $X_P$ |
|---|---|---|
| $C_{\text{Proactive}}$ | 1141 | 1152 |
| $C_{\text{Reactive}}$ | 15029 | N/A |

VMs required to empty the buffer in one minute or the average number of minutes required by a single VM to empty the buffer. This is illustrated by figures 2(a) and 2(b) that show the average buffered requests per VM at any second for $C_{\text{Hybrid}}$ and $C_{\text{Reactive}}$ respectively when $N = 100$ and $K = 1$. In Figure 2(a) there are three major peaks when the buffered load per VM is above 1000 requests resulting in a relatively small $V$ and $\overline{UP}$ in the table. These three peaks are a result of sudden large load increases. On the other hand, Figure 2(b) shows more peaks with buffered load more than 50000 requests per VM, resulting in a relatively high $V$ and $\overline{UP}$. The average required buffer size for $C_{\text{Reactive}}$ per VM in order to service all requests is almost 3 times the buffer size required by $C_{\text{Proactive}}$. Thus, for a limited buffer size, $C_{\text{Reactive}}$ drops many more requests than $C_{\text{Hybrid}}$.

Figures 3(a) and 3(b) show the load and the provisioned capacity using both controllers for the full trace when $\lambda = 100$ and $F = 1$. These plots show the macro behavior of the controllers. The total number of buffered requests is the reason for having very high load peaks for $C_{\text{Reactive}}$. The largest spike seen in Figure 3(a) was around the fifth of May at 11:15. For ten minutes, the load suddenly increases tenfold and then starts oscillating causing some instability in $C_{\text{Proactive}}$. Notable, as the buffered load is emptied over $r$ time units, the capacity does not increase with the same rate as the numbered of buffered requests increase.

To study the micro behavior of the controllers, figures 4(a) and 4(b) show the load and controller output for one and half hour from 21:21:12 on 25 June, 1998 to 22:44:31 on the same day. These figures show that $C_{\text{Proactive}}$ tracks the envelope of the load by keeping the provisioned capacity slightly higher than the load while $C_{\text{Reactive}}$ oscillates the provisioned capacity with the increase or decrease of the load. Note that as the capacity of a single VM is variable, the capacity in Figure 4(a), which is measured in number of requests, appears to be oscillating. What actually happens is that the number of provisioned VMs drops gradually from 13 to 6 with no oscillations.

**Table 4: Properties of the cluster workload.**

| | execution time | queue time | total time |
|---|---|---|---|
| Median | 347.4 s | 3.6 s | 441.6 s |
| Average | 3961.8 s | 220.76 s | 4182.5 s |
| $90th$ percentile | 3803 s | 3325 s | 4409 s |

Table 3 shows $X_R$, the total number of servers added and removed by the reactive component of a controller, and $X_P$, the total number of servers added and removed by the proactive component, using $C_{\text{Hybrid}}$ and $C_{\text{Reactive}}$ for a simulation run with $F = 1$ and $\lambda = 100$. The total number of server added or removed by $C_{\text{Proactive}}$ is 2293 servers almost one seventh of the total number of server added or removed by the reactive controller. These results illustrate how the reactive controller increases resource oscillations.

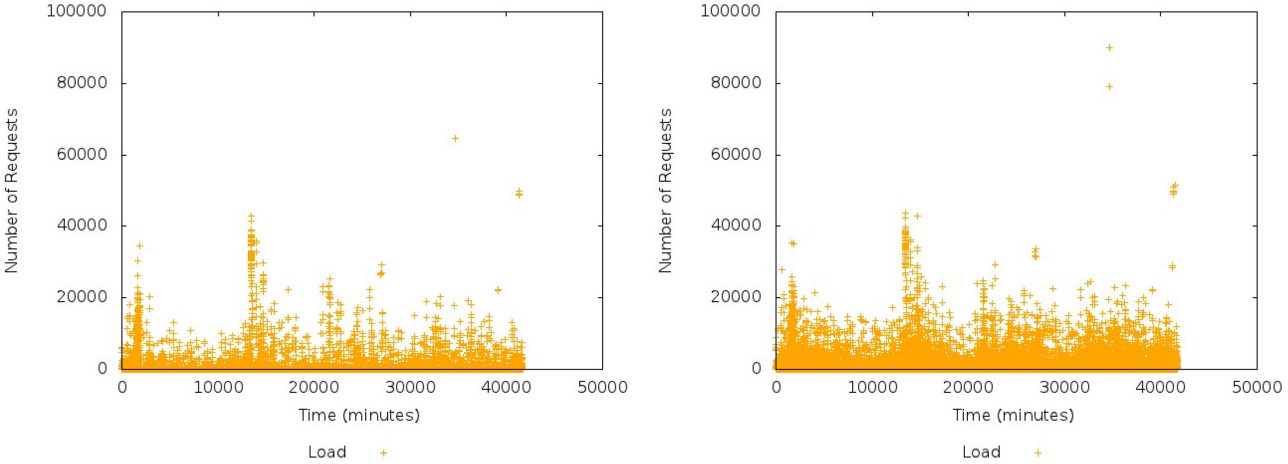
## 3.3 Cluster workload performance evaluation

Recently, Google published a new sample dataset of resource usage information from a Google production cluster [26]. The traces are from an cluster with 11000 servers. As this cluster is used to run a mixture of MapReduce and other computationally intensive jobs, the traces are representative for scientific workloads.

In this experiment, there is no risk of causing oscillations if $r = 1$ since most of the requests take more than 1 minute to serve. The median job length in the workload is 347.5 seconds or almost 6 minutes. Table 4 summarizes the statistical properties of the tasks in the workload. Due to lack of space we do not comment more on the properties of the workload. $K$ is set to 5 jobs also for this experiment. We define that a job is delayed if it remains in the queue for more than 1 minute. The median number of tasks that can be processed on a single machine in the trace is 170, while the minimum is 120. To be more conservative, we set the number of tasks assigned to a server to 100.

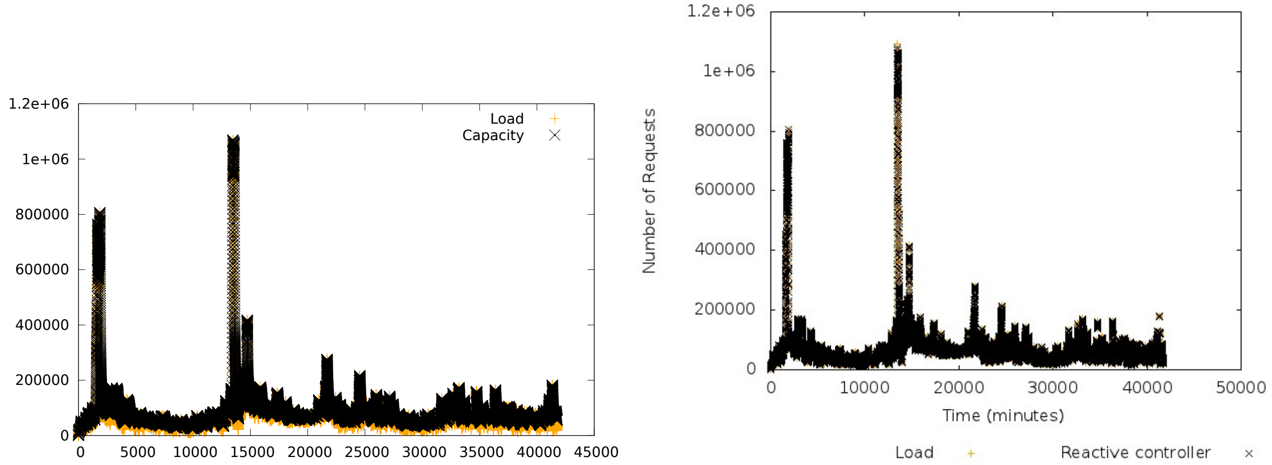
Table 5 shows the performance of the proactive and reactive controllers. The amount of under-provisioning using the $C_{\text{Reactive}}$ is almost three times that of $C_{\text{Proactive}}$. This comes at the cost of over-provisioning on average 164 VMs compared to around 1.4 VMs for the reactive controller. However, the amount of over-provisioning is still low, around 25%, as $C_{\text{Proactive}}$ used 847 VMs on average, as compared to 687 VMs for the reactive controller.

While $\overline{OP}$ and $\overline{UP}$ may be crucial for a workload like the Web trace, they are less important for a workload of jobs like the cluster trace where a job can wait in the queue



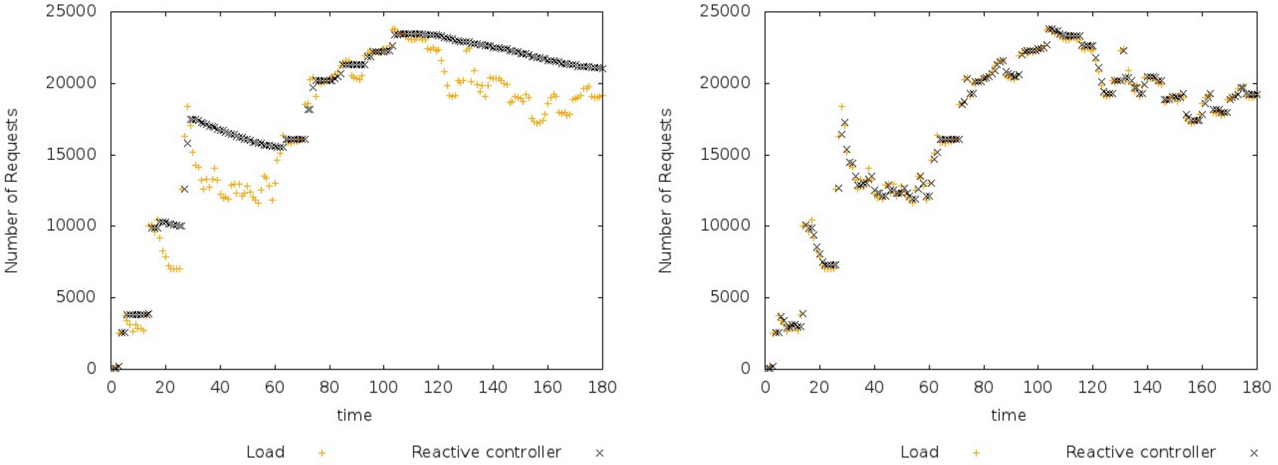
(a) $C_{Proactive}$ (b) $C_{\text{Reactive}}$

**Figure 5: Number of buffered requests over time for the cluster workload.**



(a) $C_{Proactive}$ (b) $C_{\text{Reactive}}$

**Figure 6: Load and the provisioned capacity over time for the cluster workload.**



(a) $C_{\text{Hybrid}}$ (b) $C_{\text{Reactive}}$

**Figure 7: Load and the provisioned capacity for 3 hours of the cluster workload.**

**Table 5: Cluster workload performance overview.**

| | $C_{\text{Proactive}}$ | $C_{\text{Reactive}}$ |
|---|---|---|
| $\overline{OP}$ | 164 | 1.369 |
| $\overline{UP}$ | 1.76 | 5.384 |
| $\overline{N}$ | 847 | 686.9 |
| $V$ | 3.48 | 10.22 |
| $X_R$ | 75415.0 | 505289 |
| $X_P$ | 78564.0 | N/A |

for minutes. More importantly for this workload type is $V$, the average number of buffered tasks. $C_{\text{Proactive}}$ keeps the average number of buffered tasks below $K$. On the contrary, the reactive controller's average buffer length is double the allowed buffer size $K$ and three times that of the proactive controller. This is illustrated in figures 5(a) and 5(b) that show the number of buffered requests over time.

We also note that in total, the number of VMs added and removed by the reactive controller is 505289 compared to 153979 by the proactive controller. This means that the reactive controller results in more oscillations also for the cluster workload.

Figures 6(a) and 6(b) show the load and provisioned capacity for $C_{\text{Proactive}}$ and $C_{\text{Reactive}}$. The proactive controller tracks the envelope of the workload, i.e., the capacity stays ahead of the load most of the time, whereas the reactive controller always lags the load by at least one time unit. Due to the large number of points plotted, the load appears as if it is completely covered with the capacity. In order to see the micro behavior of the two controllers we plot the load and capacity for both controllers for the first 3 hours of the trace in figures 7(a) and 7(b). The figures show how oscillations are reduced using the proactive controller. For example, for the sudden decreases in load at minutes 15 and 30, $C_{\text{Reactive}}$ quickly deallocated VMs followed by reallocations as load increased again. In contrast, $C_{\text{Proactive}}$ kept most of the allocated VMs, causing less oscillations. To summarize the experiments, the workload characteristics and the SLA requirements influence the performance of both controllers considerably. We also note that our elasticity controller is highly scalable with respect to service workload and infrastructure size. In the performed evaluations, the controller required on average a few milliseconds to make a decision.

# 4. RELATED WORK

Elasticity is an incarnation of the dynamic provisioning problem which has been studied for over a decade [7] from the perspectives of both server provisioning and cloud computing. Different approaches have been proposed to solve the problem in both its old and new incarnations. Some previous research considered only vertical elasticity [17, 27]. while many others considered horizontal elasticity in different contexts [20, 28].

Urgaonkar et al. [24] were among the first to discuss the effect of virtualization on the provisioning problem or what we call horizontal elasticity. They proposed an adaptive controller composed of a predictive and a reactive components. The predictive component acts in the time scale of hours or days. It provisions resources based on the tail distribution of the load. The reactive component acts in the time scale of minutes to handle flash crowds by scaling up the resources provisioned. The model of the predictive controller is tuned according to the under-provisioning of resources seen in the past a few hours. Scale down is not considered.

Gandhi et al. [10] propose a similar controller. The main difference is in the predictive controller design. Their predictive controller identifies patterns in the workload using a workload forecaster which discretizes it into consecutive, disjoint time intervals with a single representative demand value. Workload forecasting is done on the time scale of days i.e., the model of the predictive controller is changed at most once a day. In their approach there is no way to tune the model of the predictive controller and they do not consider scale down of resources.

Malkowski et al. [18] propose a controller for n-tiered applications. They add to the predictive and reactive controller a database of previous system states with good configurations. The elasticity controller starts by looking up if the current state of the system in the database. If the state is found then the configuration corresponding to the state is used. Otherwise, the reactive controller determines the underutilized state or over-utilized state and provisions resources according to the load. In addition, the predictive controller uses Fourier transforms to forecast the future workload for each tier from the past.

A much simpler approach is proposed by Calheiros et al. [5]. They model a cloud provider using basic queueing theory techniques. They assume heterogeneous requests that take constant time to process.

# 5. CONCLUSION

In this paper, we consider the problem of autonomic elasticity control for cloud infrastructures. The infrastructure is modeled as a $G/G/N$ queue with variable $N$. The model is used to design an adaptive proportional controller that proactively adapts based on the changes in the load dynamics. The controller takes into account resource heterogeneity, delayed requests, and variable VM service rates. A hybrid controller combines the designed controller with a reactive controller that reacts to sudden increases in the load. The combined controller tries to follow the workload envelope and avoids premature resource deallocation.

Using simulations we compare the proposed controller to a completely reactive controller. Two traces with different characteristics are used, Web traces from the FIFA world cup that are quite bursty in nature with simple requests and cluster traces from Google with jobs as long as 1 hour. Simulation results show that our proposed controller outperforms the reactive controller by decreasing the SLA violation rate by a factor between 70 for the Web workload and 3 for the cluster one. The reactive controller required three times larger buffers compared to our controller. The results also show that the proposed controller reduces resource oscillations by a factor of seven for the Web workload traces and a factor of three for the cluster traces. As a tradeoff, the hybrid controller over-provisions between 20% and 30% resources as compared to a few percent for the reactive one.

# 6. ACKNOWLEDGMENTS

We would like to thank the reviewers for their constructive comments. Financial support has been provided in part by the European Community's Seventh Framework Programme under grant agreement #257115, the Lund Center for Control of Complex Engineering Systems, and the Swedish Government's strategic effort eSSENCE.

## 7. REFERENCES

[1] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *NOMS 2012, IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2012. in press.

[2] I. Ari, B. Hong, E. Miller, S. Brandt, and D. Long. Managing flash crowds on the Internet. 2003.

[3] M. Arlitt and T. Jin. "1998 world cup web site access logs", August 1998.

[4] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall, Upper Saddle River, N.J., fourth edition, 2005.

[5] R. N. Calheiros, R. Ranjan, and R. Buyya. Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In *International Conference on Parallel Processing (ICPP)*, pages 295 –304, sept. 2011.

[6] K. Chard, M. Russell, Y. Lussier, E. Mendonca, and J. Silverstein. Scalability and cost of a cloud-based approach to medical nlp. In *Computer-Based Medical Systems (CBMS), 2011 24th International Symposium on*, pages 1–6. IEEE, 2011.

[7] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116. ACM, 2001.

[8] T. Chieu, A. Mohindra, A. Karve, and A. Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on*, pages 281 –286, oct. 2009.

[9] A. J. Ferrer, F. HernÃąndez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. ForgÃş, T. Sharif, and C. Sheridan. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66 – 77, 2012.

[10] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah. Minimizing data center sla violations and power consumption via hybrid resource provisioning. In *International Green Computing Conference and Workshops (IGCC)*, pages 1 –8, july 2011.

[11] T. Gunarathne, T. Wu, J. Qiu, and G. Fox. Cloud computing paradigms for pleasingly parallel biomedical applications. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 460–469. ACM, 2010.

[12] J. Herskovic, L. Tanaka, W. Hersh, and E. Bernstam. A day in the life of pubmed: analysis of a typical dayâĂŹs query log. *Journal of the American Medical Informatics Association*, 14(2):212, 2007.

[13] H. Khazaei, J. Misic, and V. Misic. Modelling of cloud computing centers using m/g/m queues. In *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*, pages 87 –92, june 2011.

[14] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[15] H. Li. Realistic workload modeling and its performance impacts in large-scale escience grids. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):480–493, 2010.

[16] H. Li and T. Yang. Queues with a variable number of servers. *European Journal of Operational Research*, 124(3):615–628, 2000.

[17] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal. Optimal multivariate control for differentiated services on a shared hosting platform. In *46th IEEE Conference on Decision and Control*, pages 3792–3799. IEEE, 2007.

[18] 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*, pages 131–140. ACM, 2011.

[19] M. Morari. Robust stability of systems with integral control. *IEEE Transactions on Automatic Control*, 30(6):574–577, 1985.

[20] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowstron. Everest: Scaling down peak loads through i/o off-loading. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages 15–28. USENIX Association, 2008.

[21] K. Ogata. *Modern control engineering*. Prentice Hall PTR, 2001.

[22] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. *SIGPLAN Not.*, 46:111–120, Mar. 2011.

[23] H. Truong and S. Dustdar. Cloud computing for small research groups in computational science and engineering: current status and outlook. *Computing*, pages 1–17, 2011.

[24] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst.*, 3:1:1–1:39, March 2008.

[25] J. Vöckler, G. Juve, E. Deelman, M. Rynge, and G. Berriman. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, pages 15–24. ACM, 2011.

[26] J. Wilkes. More google cluster data, November 2011.

[27] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. *International Conference on Autonomic Computing*, page 25, 2007.

[28] Q. Zhu and G. Agrawal. Resource provisioning with budget constraints for adaptive applications in cloud environments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 304–307. ACM, 2010.