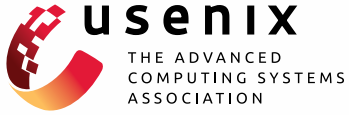


Model-driven Elasticity and DoS Attack Mitigation in Cloud Environments

Cornel Barna and Mark Shtern, ***York University;*** **Michael Smit,** ***Dalhousie University;***
Hamoun Ghanbari and Marin Litoiu, ***York University***

https://www.usenix.org/conference/icac14/technical-sessions/presentation/barna

This paper is included in the Proceedings of the
11th International Conference on Autonomic Computing (ICAC '14).

June 18–20, 2014 • Philadelphia, PA

ISBN 978-1-931971-11-9

Model-driven Elasticity and DoS Attack Mitigation in Cloud Environments

Cornel Barna
York University
Toronto, Ontario, Canada
cornel@cse.yorku.ca

Mark Shtern
York University
Toronto, Ontario, Canada
mark@cse.yorku.ca

Michael Smit
Dalhousie University
Halifax, Nova Scotia, Canada
msmit@dal.ca

Hamoun Ghanbari
York University
Toronto, Ontario, Canada
hamoun@cse.yorku.ca

Marin Litoiu
York University
Toronto, Ontario, Canada
mlitoiu@yorku.ca

Abstract

Workloads for web applications can change rapidly. When the change is an increase in customers, a common adaptive approach to maintain SLAs is elasticity, the on-demand allocation of computing resources. However, application-level denial-of-service (DoS) attacks can also cause changes in workload, and require an entirely different response. These two issues are often addressed separately (in both research and application). This paper presents a model-driven adaptive management mechanism which can correctly scale a web application, mitigate a DoS attack, or both, based on an assessment of the business value of workload. This approach is enabled by modifying a layered queuing network model previously used to model data centers to also accurately predict short-term cloud behavior, despite cloud variability over time. We evaluate our approach on Amazon EC2 and demonstrate the ability to horizontally scale a sample web application in response to an increase in legitimate traffic while mitigating multiple DoS attacks, achieving the established performance goal.

1 Introduction

The use of software-defined infrastructure enables the addition or removal of resources (computation, storage, networking, etc.) to or from a deployed web application at run-time in response to changes in workload or in the environment. Central to this *elasticity* is the use of mechanisms that autonomically decide when to make these changes. Many approaches have been proposed and tested (see for example a recent survey [7]), including reactive approaches that establish thresholds or elasticity policies which determine when changes will be made (e.g., [10, 18, 17, 5]) and proactive approaches that attempt to anticipate future requirements using techniques like queuing models [30, 9], simulation-generated state machines [28], or reinforcement learning [4]. The focus is typically on meeting a desired service level by ensuring provisioned resources are sufficient to handle a workload, perhaps while minimizing the total infrastructure cost [9].

The typical assumption of elasticity mechanisms is that all traffic arriving at the application is desirable. This is not always the case. For example, an application-level denial of service (DoS) attack has many of the same characteristics of an increase in legitimate visitors, especially a low-and-slow DoS attack [25], but represents undesirable load on the application. Such attacks are increasing in volume and sophistication [6], due in part to freely available tools [14, 22]. A common response to a denial-of-service attack at the application layer is to add resources to ensure the service remains available (e.g. [14, 8, 16]), which resembles elasticity. However, deploying sufficient resources to handle a major DoS attack is expensive [23], with little return on investment. Another example is a cost of service attack, where the goal is not to deny service but to increase the cost of offering a service [12, 26]; or heavy traffic from an online community (the so-called Slashdot Effect) that does not generate revenue.

In this paper, we propose, implement, and evaluate a unified approach to enabling elasticity and mitigating DoS attacks. Rather than view DoS attempts as malicious traffic (in contrast to legitimate traffic), or even an evolved definition of "any workload beyond our capacity" [3], we define DoS traffic to be any segment of workload we cannot handle *while still providing value to the organization*. This perspective offers the opportunity to view self-management as a business decision based on a cost-benefit analysis of adding resources: if there is benefit (e.g. increased sales, ad impressions, profit, brand reputation, etc.) that exceeds the expected cost, then add resources; otherwise, manage the traffic. Workload is regarded not as malicious or legitimate, but rather as either (potentially) undesirable or desirable.

We describe three primary contributions of this work: an adaptive management algorithm for choosing which

portions of a workload need additional resources and which portions represent undesirable traffic and should be mitigated; adapting a layered queuing network (LQN) model to cloud environments to enable proactive cost-benefit analysis of workload; and an implementation and a series of experiments to evaluate this approach in the Amazon EC2 IaaS cloud environment.

Our algorithm (§2) examines portions of the workload and assesses whether incoming traffic is desirable or undesirable. This decision is based on a runtime software quality metric called the cloud efficiency metric [26], which at its most basic calculates the total cost of the software-defined infrastructure and calculates the ratio to the revenue generated by incoming traffic (though in the general case, value can be defined very broadly). Traffic considered undesirable is handled as described in previous work [3, 2], where instead of being discarded it is forwarded to a *checkpoint*. At this checkpoint, a challenge is issued and the user is asked to verify that they are a legitimate (and valuable!) visitor (for example, using a CAPTCHA test as in [20]). The management is completely autonomic; this avoids the known problems with the complexity of manually tuning threshold-based elasticity rules [10].

Estimating the cost-benefit potential requires a proactive approach that takes measurements from the deployed infrastructure and makes short-term predictions. Our overall approach does not prescribe which mechanism should be used; we have chosen to illustrate our approach using a LQN model called OPERA. Section 3 describes the challenges in using OPERA to predict cloud behavior in practice. We present experimental results demonstrating how OPERA diverges from reality over time due to the unpredictable variability of cloud services [24], describe the modifications required to account for unexplained delays, and a second set of results that demonstrate with the modifications the LQN remained synchronized with the actual performance of the real cloud system.

We implemented our algorithm (§4), deployed a sample e-commerce application protected with our updated protection/elasticity autonomic manager, and tested response to a several attack scenarios: DoS alone, increase in customer traffic alone, and both combined (the common case where a site becomes more popular but also attracts negative attention). Our results (§5) show that the application is protected from all forms of surging traffic by adding servers or mitigating undesirable traffic, as appropriate. We also identify a limitation of our approach: the reaction time is slow enough that a temporary backlog of requests can be created, which skews calculations and leads to the temporary mitigation of desirable traffic until the model recovers. We present an example of this limitation.

2 Methodology

The goal of our approach is to treat desirable traffic (which generates business value) differently than undesirable traffic (which consumes resources disproportionate to the value created). This novel broad view of elasticity better reflects business objectives, while also addressing issues that have historically been dealt with separately. To achieve this goal, the autonomic manager must be capable of differentiating between the two, and adding or removing resources to ensure desirable traffic encounters sufficient quality of service without overspending, while routing undesirable traffic through an additional checkpoint. In this section, we introduce an algorithm for an adaptive manager with these capabilities.

The overall model of the approach resembles a standard feedback-loop, with an adaptive manager accepting monitoring data, using a predictive model to inform decision-making, and executing decisions autonomically using a deployment component capable of adding and removing resources. The managed system is a standard three-tier web application.

The behavior of the adaptive manager is described in Algorithm 1. At each iteration, a new set of metrics is observed from the managed system and its environment, including current workload, current performance, and current deployment information (which includes any ongoing traffic redirection or scaling activities). Some of this information is provided for each distinct class of traffic. For example, traffic accessing features related to browsing an e-commerce catalog might be grouped into a single class of traffic; similarly, features related to the checkout process might get their own class. These classes (sometimes called usage scenarios) allow traffic to be treated more granularly than simply looking at overall traffic to the application. A class of traffic corresponds to classes of services used in Layered Queuing models (§3.1).

Included in the set of performance metrics is the current cost efficiency (CE) [26], a runtime software quality metric that captures the ratio of the benefit derived from the application to the cost of offering the application: $CE = \frac{\text{application benefit function}}{\text{infrastructure cost function}}$.

Due to size constraints, the details of the cost efficiency metrics are not included in this paper, but the reader can find an in-depth presentation in [26]. To summarize, the **cost function** must capture the real cost of offering a given application on the cloud for the given period (e.g., per hour). This function excludes other costs related to the application, e.g. the cost of development, the cost of goods sold, and customer support. It is not a measure of overall profitability; rather, it captures current infrastructure costs. The **benefit function** must capture the benefit the application provides to the

Algorithm 1: Decision Algorithm – The algorithm used by the adaptive manager to choose appropriate actions for the managed system.

input : $\mathbb{C}^u$ – the set of unaltered traffic classes;
input : $\mathbb{C}^f$ – the set of all classes of traffic redirected to a checkpoint;
input : L^u – the vector of current load on unaltered classes of traffic;
input : L^f – the vector of current load on each class of traffic redirected to a checkpoint;
input : M_m – the vector of measured performance metrics;
input : svr_{cur} – the number of current web servers;
input : svr_{max} – the maximum number of web servers that can be allowed to run;
input : err – the accepted error for the model estimations;
output: $\mathbb{A}$ – the deployment plan.

1 Use LQM to compute the estimated performance metrics, M_e, for the load L^u;
2 **while** $\left|1 - \frac{M_e}{M_m}\right| > err$ **do**
3 $D \leftarrow$ *Kalman*(M_m, L^u, *LQM*); $M_e \leftarrow$ *LQM*(D, L^u);
4 $svr_{ce} \leftarrow$ the maximum number of servers that can be added and still be cost effective;
5 $A \leftarrow$ {**do nothing**};
6 **if** M_m violates SLOs **then**
7 $svr \leftarrow min(svr_{max}, svr_{cur} + svr_{ce})$;
8 $n \leftarrow$ *CalculateServersToAdd*(L^u, M_m, svr_{cur}, svr);
9 **if** $n > 0$ **then**
10 $\mathbb{A} \leftarrow$ {**add** n web servers};
11 **else**
12 $\mathbb{C} \leftarrow$ *TrafficClassesToRedirect*(L^u, M_m, err, $\mathbb{C}^u$);
13 $\mathbb{A} \leftarrow$ {**redirect** traffic classes $\mathbb{C}$};
14 **else**
15 set in the model the number of web servers to $svr_{cur} + svr_{ce}$;
16 $\mathbb{C} \leftarrow$ *TrafficClassesToRestore*(L^u, L^f, M_m, err, $\mathbb{C}^f$);
17 **if** $\mathbb{C} \neq \emptyset$ **then**
18 $svr \leftarrow svr_{cur} - 1$;
19 $\mathbb{C}_{tmp} \leftarrow \emptyset$;
20 **while** $\mathbb{C}_{tmp} \neq \mathbb{C}$ **do**
21 $svr \leftarrow svr + 1$;
22 set in the model the number of web servers to svr;
23 $\mathbb{C}_{tmp} \leftarrow$ *TrafficClassesToRestore*(L^u, L^f, M_m, err, $\mathbb{C}^f$);
24 **if** $svr - svr_{cur} > 0$ **then**
25 $\mathbb{A} \leftarrow$ {**add** $svr - svr_{cur}$ web servers} $\cup$ {**stop redirecting** traffic classes $\mathbb{C}$};
26 **else**
27 $\mathbb{A} \leftarrow$ {**stop redirecting** traffic classes $\mathbb{C}$};
28 **else**
29 $n \leftarrow$ *CalculateServersToRemove*(L^u, M_m, svr_{cur});
30 **if** $n > 0$ **then**
31 $\mathbb{A} \leftarrow$ {**remove** n web servers};
32 **return** A

organization. Typically, organizations have mechanisms for assessing this benefit at least at the macro-level. The benefit may come from many sources: revenue, advertising, brand awareness, customer satisfaction, number of repeat customers, or any number of business-specific metrics. For example, a denial of service attack on an e-commerce website would reduce the value of incoming traffic (as fewer visitors would be able to make purchases), which would reduce the overall cost efficiency. If an adaptive manager were in use and were to add additional resources to handle the DoS traffic, the current value would be maintained, but the cost would increase: the overall effect would be the same.

OPERA is used to estimate a set of performance metrics (CPU utilization, response time, throughput), which are compared to the measured set of performance metrics to ensure the model is still synchronized with the system (line 2; if not, it is re-tuned using Kalman filters [32]).

On line 6, we test compliance with service-level objectives (SLOs); if any measured performance metrics are non-compliant (perhaps due to a decrease in cost efficiency, or an increase in response time), a remedial action must be taken (logic regarding cool-down times to avoid thrashing is omitted for clarity). The remedial action is chosen from two options (adding servers because the traffic has value or redirecting traffic to a checkpoint because it does not). To decide on which action is appropriate, we call a function which uses the predictive model to estimate the impact of adding one or more web servers, up to a maximum cap on the number of servers. If adding web servers is estimated to bring performance metrics in compliance with the SLO, this solution is executed and the chosen number of servers is added (line 8). Performance metrics include the cost efficiency metric, which means that if additional traffic does not add business value, adding servers will increase the cost and therefore lower cost efficiency. In this case the approach will be rejected and 0 will be returned. The algorithm will then consider redirecting some traffic to a checkpoint instead. The algorithm considers each class of traffic separately. For example, it might be appropriate to redirect the traffic of users who access only free services, while preserving the traffic of those who pay a monthly subscription fee. Or we might give priority to the class of traffic that includes the actual checkout process, versus the customer discussion forums. The function TrafficClassesToRedirect determines which class of traffic should be redirected. The complete definition is provided in [3], but conceptually the LQN model is used to produce a set of performance metric estimates based on blocking various classes of traffic. The set returned consists of the classes that produce the best performance metrics (including cost efficiency). Traffic to these classes are redirected to a checkpoint for verification of legitimacy; this checkpoint also serves as a *speedbump* for legitimate traffic.

In contrast, if measured performance complies with the SLOs, the algorithm checks to see if we can restore classes of traffic (or if we could restore classes of traf-

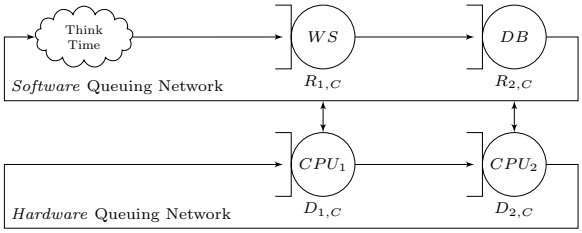


Figure 1: Software and hardware layers in a LQN of a 2-tier web system.

fic if we added additional servers, line 16) or if we can reduce the number of servers. The model is used to predict performance measures (including cost efficiency) under each possible action. If the model estimates that performance metrics would remain in compliance with the SLO after restoring traffic classes, these classes are restored; similarly for removing one or more servers (line 29). It is important to note that servers can be removed even while traffic is being redirected; the elasticity function is focused on the desirable traffic, not on overall traffic. The number of servers to remove is calculated as the largest possible reduction before the model estimates SLOs would be violated.

The algorithm will terminate returning a set of actions (which may be a null set), specifying which traffic classes to redirect (or restore) and specifying the number of servers to add (or remove).

This general approach allows an administrator to specify their expected SLOs, to define a benefit function, and to define classes of traffic; however, they are not expected to write procedural rules or detailed policies. The adaptive manager is responsible for deciding both what action to take (managing traffic or adding/removing resources) and the magnitude of that action (how much traffic to manage, how many resources to add/remove). The inclusion of a general cost efficiency metric allows the adaptive manager to make decisions that reflect business objectives, rather than going to great expense to handle large amounts of traffic.

3 A Layered Queuing Network for Cloud Environments

A layered queuing network model (LQN) is at the heart of our methodology. While our algorithm is general, we implement it using a particular layered queuing network (named OPERA) which we have used successfully to model transactional web applications deployed on hardware infrastructure. In the process of validating its accuracy in cloud environments, we found that over time it diverged from reality, and that for some values (such as modeled response time) it was consistently below the actual values. This section describes using a LQN to model an application on the cloud.

3.1 Previous model

For a transactional web application such as those examined in this paper, the user interaction with the system is modeled using *classes of services* (or simply *classes*), a service or a group of services that have similar statistical behavior and have similar requirements. When a user begins interacting with a service, a *user session* is created, and persists until the user logs out or becomes inactive. We define N as the number of active users at some moment t; these users can be distributed among different classes of services. For a system with C classes, we define N_c as the number of users in class C, thus $N = N_1 + N_2 + \cdots + N_C$. N is also called *workload intensity* or *population* while combinations of N_c are called *workload mixes* or *population mixes*.

Any software-hardware system can be described by two layers of queuing networks [21, 19]. The first layer models the software resource contention, and the second layer models the hardware contention. To illustrate the idea, consider a web based system with two software tiers, a web application server (WS) and database (DB) server (see Figure 1). Each server runs on dedicated hardware, which for the purpose of this illustration we will limit to CPUs only (CPU_1 and CPU_2 respectively). The hardware layer can be seen as a queuing network with two queues, one for each hardware resource. The software layer also has two queues, one for the WS process and another for the DB process, which queue requests waiting for an available thread (or for *critical sections*, *semaphores*, etc.). The software layer also has a *Think Time* centre that models the delay between requests that replicate how to model how long a user waits before sending their next request.

Each resource has a *demand* (or *service time*, i.e. the time necessary for a single user to get service from that resource) for each *class*. If in this example there are two classes of service, there will be four demands for the hardware layer: each class will have a demand for each CPU. The service times (demands) at the software layer are the *response times* of the hardware layer. In our case, for class C, they are $R^s_{1,C}$ and $R^s_{2,C}$, and they include the demand and the waiting time at the hardware layer (we use the upper script s to denote software metrics that belong to the software layer). Ideally, hardware demand is based on measured values; however, this is impractical for CPUs because of the overhead imposed by collecting such measurements. In our approach the CPU demands are estimated using Kalman filters. Once the model is created, it is iteratively tuned, also using Kalman filters.

This model has been used to inform a variety of

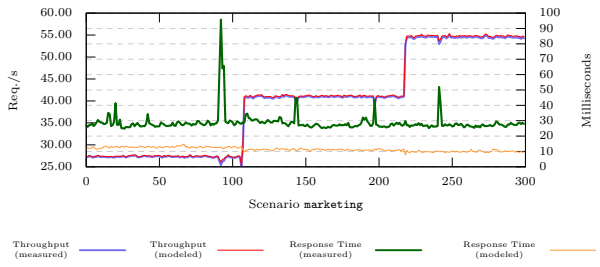


Figure 2: Observed versus estimated values for 2 key performance metrics, showing the `marketing` scenario only.

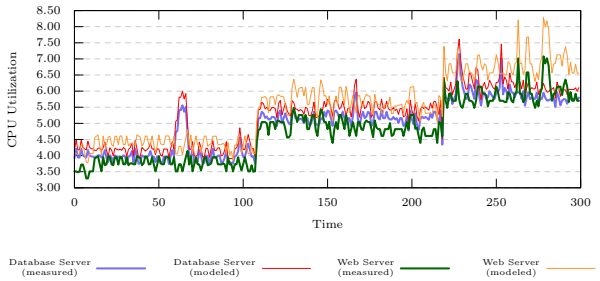


Figure 3: The CPU utilization for the `marketing` scenario; the model is not sychronized with observed measurements.

adaptive systems, including implementing elasticity policies [9] and mitigating DoS attacks [3, 2]. In earlier work, Zheng et al. [32] present a method for tuning model parameters for a web application. Properly tuned models have been shown to accurately estimate performance metrics [32].

3.2 OPERA in the Cloud

Despite the proven track record of this model, when we switched from modelling a private data center to modelling the public cloud (Amazon AWS EC2), we noticed significant differences between the model's estimates and the actual observed performance metrics.

To illustrate the issue, we deployed a sample web application (see §5) and generated an increasing workload to a single class of service, `marketing` scenario. Figure 2 shows the measured and modeled response time (right Y-axis) and throughput (left Y-axis) when the workload is increased. The picture shows that the model is well-synchronized with the system for throughput, but not for the response time; the measured response time is almost three times the estimated one. As the workload increases, the synchronization procedure manages to tune the model for the throughput, but does not improve the accuracy of predicted response time.

Figure 3 shows the CPU utilization for the database server and the application server, both observed and predicted. The distance between the observed and predicted values is noticeable, and diverges further as workload increases, demonstrating that this metric is also not synchronized between the model and reality.

LQN models are well-established approaches to predicting the performance of web applications; after ruling out measurement errors, we concluded our particular model is missing an unknown factor related to the known variance in EC2 [24, 27]. In order to improve the estimated metrics and reduce the modelling error overall, we added a "cloud delay centre" designed to capture all the undocumented work or delay. The cloud delay queue is shared by all classes of traffic, it has no limit, it exists at the software level, and it is the first queue encountered by incoming requests.

Following this change, we again tested synchronization and obtained substantially better results §5. The modified LQN is used throughout the remainder of this paper to make decisions about workload, to inform adaptive systems, and to implement elasticity policies; it demonstrated close alignment with measured values throughout this process. Other approaches that use LQN models to predict cloud behavior but do not validate their models against an actual cloud (e.g. [15],[1]) may wish to adopt a similar solution.

4 Implementation

We have implemented our adaptive management algorithm to manage applications deployed to the Amazon EC2 public cloud infrastructure, using a framework that can automatically deploy a topology in EC2 by launching all instances required (application servers, database, load balancer), install the required software on each machine (e.g. Tomcat and all dependencies, the web application and its libraries, etc.), and configure each application (for example, add the application servers to the load balancer). The framework provides an API that accepts requests to modify the deployed topology – for example, if the request is to horizontally scale the web tier, the framework can launch an instance for the application server, install all required software, and add the new server to the load balancer. The framework is general enough to allow us to install and configure any type of application in a linux environment, once the deployer provides installation and management scripts.

Our implementation uses Amazon CloudWatch detailed monitoring to acquire performance metrics from the deployed instances. As discussed in [27], these metrics are delayed by one minute from when they are recorded. Our implementation runs the main algorithm, and acquires Cloudwatch metrics, once per minute. The arrival rate, throughput, and response time for each class of traffic are acquired from a reverse proxy on the load balancer at the same time[1]. This reverse proxy moni-

tors incoming requests and assigns them to the appropriate traffic class (based on the URL); the classification rules can be modified at runtime if necessary. The administrator of an application is responsible for defining their traffic classes based on their business logic. Measuring response time at the load balancer allows us to focus on the component of user-experienced response time that we can control. While end-to-end response time will be higher and more variable, adapting to slow user connections is outside the scope of this paper.

5 Experiments

To evaluate the contributions of this paper, we performed a set of experiments. The first examines our modified LQN to assess its ability to synchronize with the managed application so its estimates have predictive value. Experiments 2-4 examine a sample web application.

In all experiments we have used a *bookstore* application that we have developed using Java EE, which emulates an e-commerce web application. We defined six classes of traffic: `marketing` (browsing reviews and articles); `product selection` (searching the store catalog, product comparisons); `buy` (add to cart); `pay` (complete checkout); `inventory` (inventory tracking); and `auto bundle` (an upselling / discounting system). The workload used is generated randomly by a workload generator using an unevenly weighted mix of the 6 classes. Each class of traffic has a different performance impact on the deployed application.

Each experiment starts with a deployed topology, with a single application server in the web tier. The web tier is scaled horizontally by adding and removing resources. The traffic filtering approach described in [3] is used to refer undesirable traffic to a captcha to serve as the checkpoint.

5.1 Experiment 1: Synchronizing the model with public cloud resources

To verify the ability of our LQN to synchronize with reality in a cloud environment, we generated a constant workload and compared the estimated performance metrics to actual measurements at each sampling interval (Figure 4). We generated workload for all scenarios, though not all are shown due to space constraints. At each iteration or sampling interval, we measure the arrival rate for each scenario. We feed this workload into the model, and then solve the model to calculate the estimated metrics. If the error between measured and estimated values exceeds a specified threshold (10% in our case), we run the Kalman filters on the model in order to find more accurate values for the model parameters (this process is called *tuning the model*).

At $t = 0$, the estimated CPU utilization numbers (Fig. 4a) for the database server are almost double the measured values. Within 25-50 iterations, the Kalman filter settles on accurate model parameters, and the difference between measured and estimated values was around 1%. Importantly, once synchronization is achieved, it is not lost. Before we added the Cloud Delay Center, this synchronization was never achieved.

In plots 4b-4c, we show the measured response time (green line, using right Y-axis) and measured throughput (blue line, using left Y-axis) for two classes of traffic. After some initial error, the estimated values and measured values also remain within several percentage points, even through peaks and valleys.

We conclude from this data that the modified LQN has addressed the challenges of modelling a cloud environment by treating the variability of the cloud as a delay center in an LQN, and modifying the demand on that delay center using Kalman filters to account for unpredictable variability.

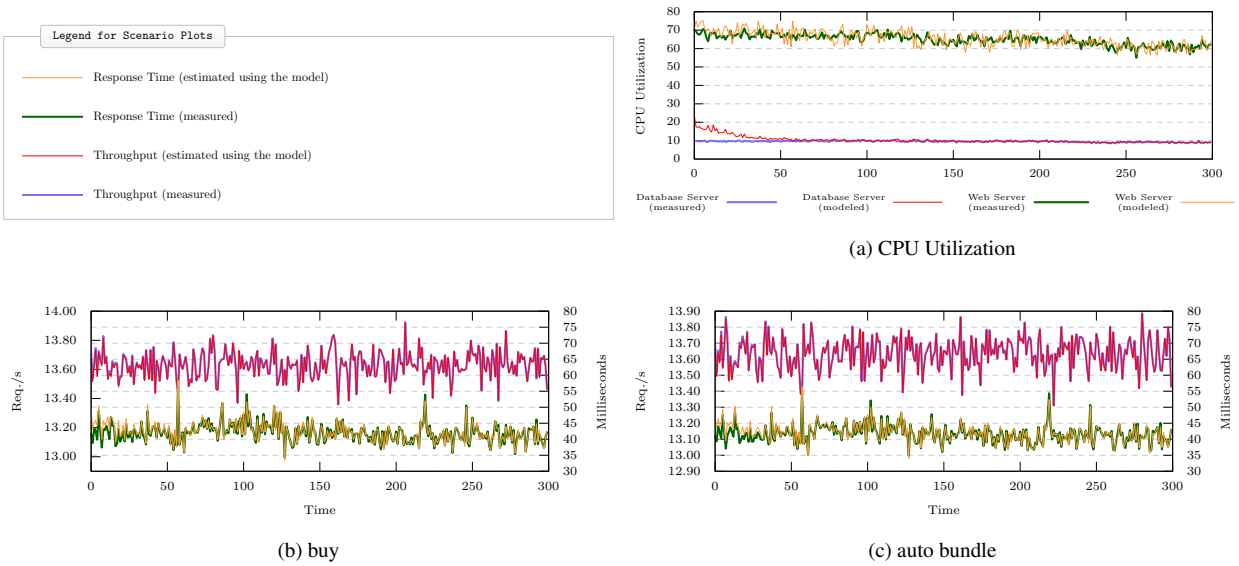
5.2 Experiment 2: Elasticity in the public cloud

This experiment examines the adaptive management algorithm's ability to provide elasticity when overall traffic increases and decreases. We use a simplified cost metric to calculate the cost of our EC2 deployment, and use the volume of the `pay` class of traffic as our benefit function. This is roughly analogous to prioritizing checkout activity over other activities. Application-level DoS attacks are not expected to generate traffic to this traffic class, because reaching the checkout page usually requires user interaction, a valid account, and valid credit card numbers[2]. The workload mix remained constant over the experiment, and therefore so did the cost efficiency metric.

Figure 5 shows the measurements obtained during this experiment. The workload generated for each scenario is captured (approximately) as the arrival rate (blue line in figures 5b–5g, on the left Y-axis). We started with a baseline workload; at iteration 50, we increased the workload. The adaptive manager added a new web server (purple line in figure 5a, on the right Y-axis). When the workload is increased, there is a brief spike in the CPU utilization metric for the web servers (red line in figure 5a, on the

[1] Initially, our framework used SNMP and JMX on the deployed VMs and application servers; however, these monitoring tools did not capture performance metrics for individual classes of traffic. Additionally, some values are measured incorrectly (such as CPU utilization) when running on a virtual machine in a cloud environment.

[2] Of course, our focus is the general approach and not optimal selection of the benefit function.



(a) CPU Utilization

(b) buy

(c) auto bundle

Figure 4: **Experiment #1.** Comparing estimated values to measured values for a selection of traffic classes for consistent workload.

left Y-axis), but also on the response time for each scenario (green line in figures 5b–5g, on right Y-axis). This violated the SLO, and caused the addition of a server because the cost efficiency metric remained within an acceptable range.

After workload increases at iteration 120 and again at 190, again a new server is added each time. Notice that after adding new servers, the CPU utilization for the web servers and the response time for the various classes of traffic have about the same values as before. This suggests that the number of servers added each time (estimated using the model) was appropriate to maintain SLOs.

At iteration 220, we dropped the workload sharply. The algorithm decided that two web servers could be safely removed. Again, the performance metrics remained acceptable following this removal.

The spikes in some of the measurements are largely due to delays in actuated new servers, leading to some backlog of requests and temporarily higher response times (off the graph, in some cases).

5.3 Experiment 3: Elasticity while mitigating DoS attacks

This experiment tests one of our key contributions: achieving elasticity to maintain SLOs for our desirable traffic, while detecting and redirecting undesirable traffic. To emulate a DoS attack, we dramatically increased the volume of traffic generated to one (or more) of the traffic classes. Our measurements from this experiment are shown in Figure 6.

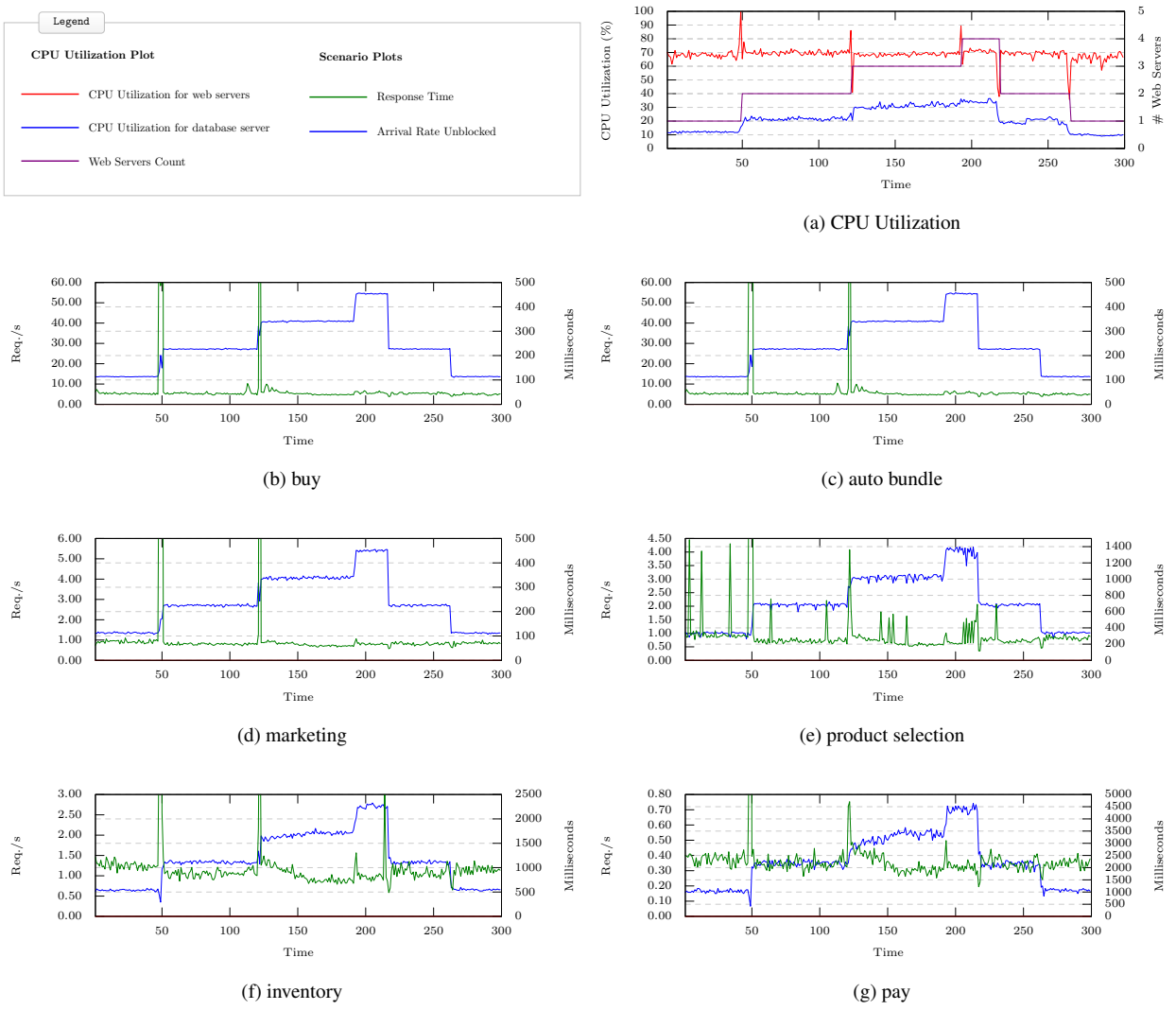
At iteration 4, we increase the workload across all traffic classes, and a new virtual machine is added. Performance degrades across all performance metrics in all classes, including the average CPU utilization. However, the addition of the new server restores performance metrics to acceptable levels.

At iteration 15 a first attack starts on a URL in the traffic class `marketing`. The arrival rate abruptly jumps from around 1.5 requests per second to close to 100 requests per second. The degradation in performance is immediate. Our algorithm provides the new metrics and workload levels to the LQN. The algorithm contemplates adding additional servers, but the increased cost is not offset by an increase in benefit, as the marketing traffic class does not generate revenue directly, and does not contribute to the benefit function. It instead determines it is necessary to redirect some traffic classes to a checkpoint and, after solving the model for various possible redirection schemes, determines (correctly) that the best course of action is to filter the requests on `marketing`.

A few iterations later (iteration 27), we emulated a second simultaneous attack on the `product selection` traffic class. Using a similar process, the algorithm once again identified the scenario under attack and redirected traffic to a checkpoint.

While these two attacks continued, the workload to the other classes increased for unrelated reasons. We can see response time (particularly for `inventory` and `pay`) increase. Once an SLO is violated, the algorithm decides two servers will be necessary to handle the continuing increase in desirable traffic (see purple line in figure 6a). This decision appears to be the correct one, as all performance metrics return to satisfactory levels (without being over provisioned).

At iteration 97, the first DoS attack (on `marketing`) is stopped, and the algorithm stops redirecting traffic for



(a) CPU Utilization

(b) buy

(c) auto bundle

(d) marketing

(e) product selection

(f) inventory

(g) pay

Figure 5: **Experiment #2.** Increasing and decreasing the workload resulted in the addition/removal of servers, while maintaining key performance metrics at acceptable levels.

that class of traffic. By iteration 120, the temporary increase in desirable traffic also resides, and at iteration 130 a web server is removed leaving the web cluster with three machines. Note there is still an ongoing DoS attack, and removing resources is counter-intuitive, but again the performance metrics suggest this was the correct decision as they are maintained at acceptable levels. When the second attack finishes, and the last redirection is halted, the system performance metrics stay within the SLOs. We do see an increase in some response times and had the experiment continued we expect the algorithm may have added an additional server.

This experiment shows that our algorithm is able to assure the elasticity of the web application and make good decisions to achieve SLOs and maintain cost efficiency even while the application is under one or more attacks.

5.4 Experiment 4: A limitation of the implementation

The previous experiments demonstrated the strength of our approach and our algorithm. However, there is a limitation: the reaction time is slower than is sometimes required to respond appropriately. This problem can be addressed by adding a statistical model that responds rapidly using a heuristic (as in [3]). The measurements from this experiment are shown in Figure 7.

When the first DoS attack starts on `marketing`, the system identified the problem and started redirecting traffic. At iteration 67, a second attack causes a severe and rapid degradation of performance metrics across all traffic classes and servers. Due to delays between iterations, the system is unprotected and the internal queues are filled. Our algorithm analyses the data and decides that two more classes need to be protected: `buy` and `auto`

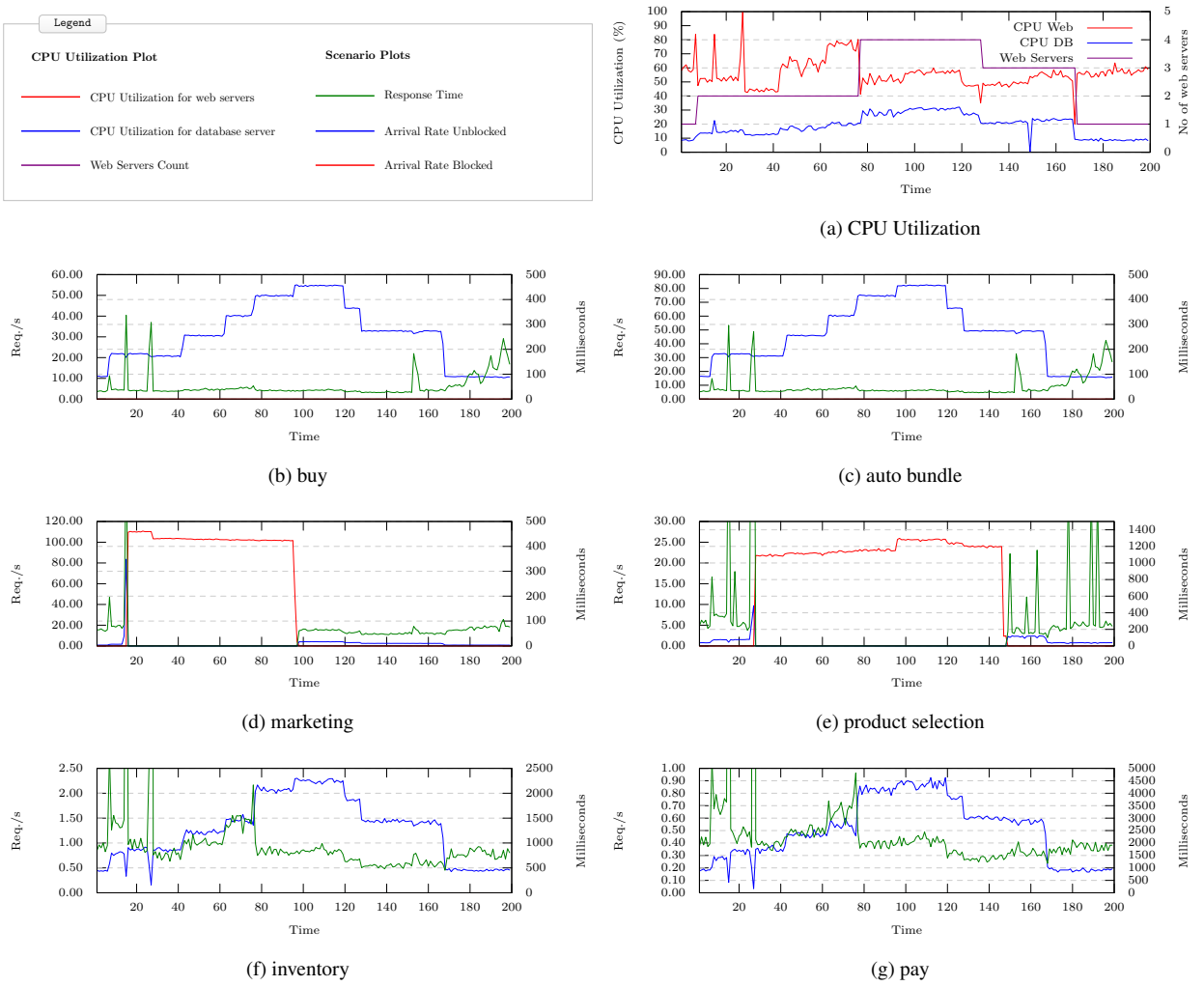


Figure 6: **Experiment #3.** Overlapping DoS attacks on two traffic classes; the algorithm mitigated these attacks while adjusting the number of VMs for the remaining workload.

`bundle`. The decision is correct in that it restores SLOs, but of course incorrect since no malicious traffic is targeting these classes.

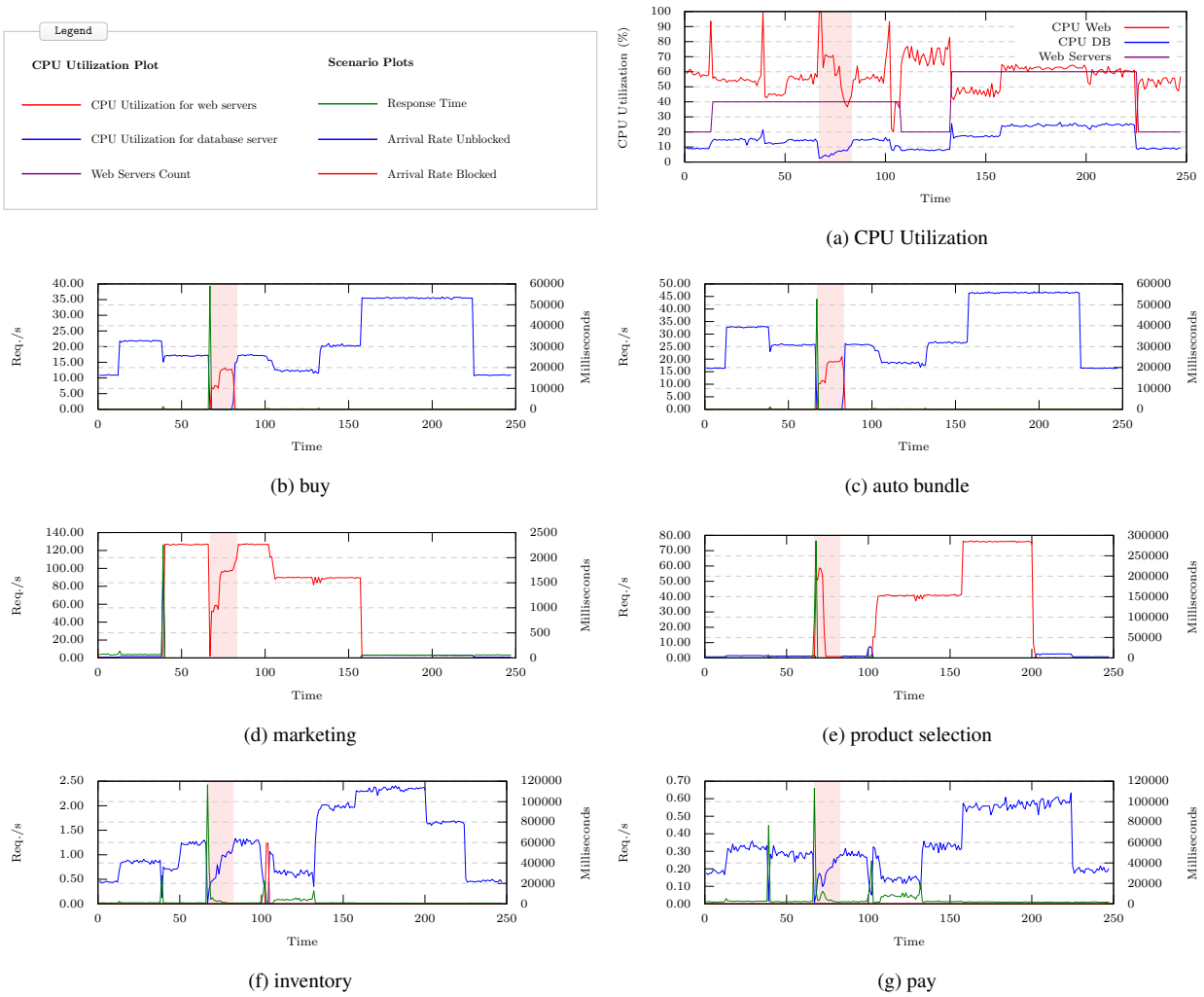
When the scenario under attack is resource-intensive, it will take multiple iterations to process the backlog of DoS requests that made it through before we started directing traffic. Although traffic to `inventory` and `pay` is not redirected, there is a significant drop in the arrival rate and an increase in response time. The drop in arrival rate is normal behaviour, because normal users will not make a new request to the server until they receive the response from their previous request.

6 Related Work

There are many approaches in achieving elasticity. Companies such as Amazon, Azure, RightScale, and Rackspace offer pre-defined rule-based autoscalers. The application owners manually define rules (often threshold based) for triggering scaling out/in actions. Then, at runtime, the autoscalers monitor the application performance metrics and evaluate them against the rules. When a rule condition becomes true, the system executes the rule's action such as adding or removing VM. Some researchers argue that specifying good threshold based rules is a challenging task [10, 29].

A potential weakness of pre-defined rule system is the thrashing effect when the system constantly adds or removes VM due to a fast changing workload. To address this problem, Iqbal et al. [13] combine rule-based scaling with statistical predictive models. Xiong et al. [31] demonstrated that an application analytic model could be used as an efficient method to identify the relationship between a number of required servers, workload and QoS.

Many researchers have designed and developed elastic algorithms without considering the cost factor; however, Han et al. [11] propose an elastic algorithm which considers both the cost and performance. An elastic al-



(a) CPU Utilization

(b) buy

(c) auto bundle

(d) marketing

(e) product selection

(f) inventory

(g) pay

Figure 7: **Experiment #4.** A DoS attack causes overcorrection due to a slow reaction, and additional classes are blocked.

gorithm will identify the application tier to scale in order to resolve the QoS issue while keeping the overall deployment cost as low as possible. A queuing analytic performance model is utilized for identification of the inefficient application tier.

The main weakness of the above approaches is that all user requests are considered desirable to the application owner. This may not be true in actual deployment environments. Many researchers and practitioners agree that DoS attacks are one of the biggest threats in the today's security landscape. Our novel approach distinguishes between desirable and undesirable traffic using cost efficiency metrics that consider not only the cost of the infrastructure, but also the business value of the workload.

7 Conclusion

This paper presented a model-driven adaptive management architecture and algorithm to scale a web application, mitigate a DoS attack, or both, based on an assessment of the business value of workload. The business value is measured through an efficiency metric as a ratio between the revenue and cost. The approach is enabled by a layered queuing network model previously used to model data centers but adapted for cloud. The model accurately predicts short-term cloud behavior, despite cloud variability over time. We evaluated our approach on Amazon EC2 and demonstrate the ability to horizontally scale a sample web application in response to an increase in legitimate traffic while mitigating multiple DoS attacks, achieving the established performance goal. We also showed the limitation of the approach which can be overcome through further work.

Acknowledgements

This research was supported by the SAVI Strategic Research Network (Smart Applications on Virtual Infrastructure), funded by NSERC (The Natural Sciences and Engineering Research Council of Canada) and by Connected Vehicles and Smart Transportation (CVST) funded by Ontario Research Fund.

References

[1] BACIGALUPO, D., VAN HEMERT, J., USMANI, A., DILLENBERGER, D., WILLS, G., AND JARVIS, S. Resource management of enterprise cloud systems using layered queuing and historical performance models. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on* (April 2010), pp. 1–8.

[2] BARNA, C., SHTERN, M., SMIT, M., TZERPOS, V., AND LITOIU, M. Model-based adaptive dos attack mitigation. In *ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* (New York, NY, USA, 2012), SEAMS 2012, ACM, pp. 119–128.

[3] BARNA, C., SHTERN, M., SMIT, M., TZERPOS, V., AND LITOIU, M. Mitigating DoS attacks using performance model-driven adaptive algorithms. *Transactions on Autonomous and Adaptive Systems 9*, 1 (Mar. 2014), 3:1–3:26.

[4] BARRETT, E., HOWLEY, E., AND DUGGAN, J. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience 25*, 12 (2013), 1656–1674.

[5] CARON, E., RODERO-MERINO, L., DESPREZ, F., AND MURESAN, A. Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds. Rapport de recherche RR-7857, INRIA, 2012.

[6] DOBBINS, R., MORALES, C., ANSTEE, D., ARRUDA, J., BIENKOWSKI, T., HOLLYMAN, M., LABOVITZ, C., NAZARIO, J., SEO, E., AND SHAH, R. Worldwide Infrastructure Security Report. Tech. rep., Arbor Networks, 2010.

[7] GALANTE, G., AND DE BONA, L. A survey on cloud computing elasticity. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on* (2012), pp. 263–270.

[8] GARG, A., AND NARASIMHA REDDY, A. L. Mitigation of DoS attacks through QoS regulation. In *Quality of Service, 2002. 10^{th} IEEE International Workshop on* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 45–53.

[9] GHANBARI, H., SIMMONS, B., LITOIU, M., BARNA, C., AND ISZLAI, G. Optimal autoscaling in a iaas cloud. In *Proceedings of the 9th International Conference on Autonomic Computing* (New York, NY, USA, 2012), ICAC '12, ACM, pp. 173–178.

[10] GHANBARI, H., SIMMONS, B., LITOIU, M., AND ISZLAI, G. Exploring alternative approaches to implement an elasticity policy. In *Proceedings of the 4th IEEE International Conference on Cloud Computing* (Washington DC, USA, 2011), IEEE.

[11] HAN, R., GHANEM, M. M., GUO, L., GUO, Y., AND OSMOND, M. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. *Future Generation Computer Systems 32* (2014), 82–98.

[12] IDZIOREK, J., AND TANNIAN, M. Exploiting cloud utility models for profit and ruin. In *IEEE International Conference on Cloud Computing* (2011), pp. 33–40.

[13] IQBAL, W., DAILEY, M. N., CARRERA, D., AND JANECEK, P. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems 27*, 6 (2011), 871–879.

[14] KARGL, F., AND MAIER, J. Protecting web servers from distributed denial of service attacks, 2001.

[15] LI, J. Z., CHINNECK, J., WOODSIDE, M., LITOIU, M., AND ISZLAI, G. Performance model driven QoS guarantees and optimization in clouds. In *in Proceedings of Workshop on Software Engineering Challenges in Cloud Computing @ ICSE 2009* (2009).

[16] LONG, M., WU, C.-H. J., HUNG, J. Y., AND IRWIN, J. D. Mitigating performance degradation of network-based control systems under denial of service attacks. In *Proc. 30th Annual Conf. of IEEE Industrial Electronics Society IECON 2004* (Washington, DC, USA, 2004), vol. 3, IEEE Computer Society, pp. 2339–2342.

[17] MARSHALL, P., KEAHEY, K., AND FREEMAN, T. Elastic site: Using clouds to elastically extend site resources. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing* (Washington, DC, USA, 2010), CCGRID '10, IEEE Computer Society, pp. 43–52.

[18] MAURER, M., BRANDIC, I., AND SAKELLARIOU, R. Enacting slas in clouds using rules. In *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I* (Berlin, Heidelberg, 2011), Euro-Par'11, Springer-Verlag, pp. 455–466.

[19] MENASCÉ, D. A. Simple analytic modeling of software contention. *SIGMETRICS Performance Evaluation Review 29*, 4 (2002), 24–30.

[20] MOREIN, W. G., STAVROU, A., COOK, D. L., KEROMYTIS, A. D., MISRA, V., AND RUBENSTEIN, D. Using graphic turing tests to counter automated DDoS attacks against web servers. In *Proceedings of the 10th ACM conference on Computer and communications security* (New York, NY, USA, 2003), CCS '03, ACM, pp. 8–19.

[21] ROLIA, J. A., AND SEVCIK, K. C. The method of layers. *IEEE Transactions on Software Engineering 21*, 8 (1995), 689–700.

[22] ROMAN, J., RADEK, B., RADEK, V., AND LIBOR, S. Launching distributed denial of service attacks by network protocol exploitation. In *Proceedings of the 2nd international conference on Applied informatics and computing theory* (Stevens Point, Wisconsin, USA, 2011), AICT'11, World Scientific and Engineering Academy and Society (WSEAS), pp. 210–216.

[23] SACHDEVA, M., SINGH, G., AND KUMAR, K. Deployment of Distributed Defense against DDoS Attacks in ISP Domain. *International Journal of Computer Applications 15*, 2 (February 2011), 25–31. Published by Foundation of Computer Science.

[24] SCHAD, J., DITTRICH, J., AND QUIANE-RUIZ, J.-A. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment 3*, 1 (2010).

[25] SHTERN, M., SANDEL, R., LITOIU, M., BACHALO, C., AND THEODOROU, V. Towards mitigation of low and slow application ddos attacks. In *IEEE International Workshop on Software Defined Systems (SDS), IEEE International Conference on Cloud Engineering (IC2E)* (Accepted, 2014).

[26] SHTERN, M., SMIT, M., SIMMONS, B., AND LITOIU, M. A runtime cloud efficiency software quality metric. In *New Ideas and Emerging Results (NIER) track, Proc. of the 2014 Intl. Conference on Software Engineering (ICSE)* (Accepted, 2014).

[27] SMIT, M., SIMMONS, B., AND LITOIU, M. Distributed, application-level monitoring of heterogeneous clouds using stream processing. *Future Generation Computer Systems 29*, 8 (2013), 2103–2114.

[28] SMIT, M., AND STROULIA, E. Autonomic configuration adaptation based on simulation-generated state-transition models. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on* (Aug 2011), pp. 175–179.

[29] SULEIMAN, B., AND VENUGOPAL, S. Modeling performance of elasticity rules for cloud-based applications. In *Enterprise Distributed Object Computing Conference (EDOC), 2013 17th IEEE International* (Sept 2013), pp. 201–206.

[30] URGAONKAR, B., SHENOY, P., CHANDRA, A., GOYAL, P., AND WOOD, T. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst. 3*, 1 (2008), 1:1–1:39.

[31] XIONG, K., AND PERROS, H. Service performance and analysis in cloud computing. In *Services-I, 2009 World Conference on* (2009), IEEE, pp. 693–700.

[32] ZHENG, T., YANG, J., WOODSIDE, M., LITOIU, M., AND ISZLAI, G. Tracking time-varying parameters in software systems with extended kalman filters. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research* (2005), IBM Press, pp. 334–345.