Resource Provisioning of Web Applications in Heterogeneous Clouds

Jiang Dejun
VU University Amsterdam
Tsinghua University Beijing

Guillaume Pierre
VU University Amsterdam

Chi-Hung Chi
Tsinghua University Beijing

Abstract

Cloud computing platforms provide very little guarantees regarding the performance of seemingly identical virtual machine instances. Such instances have been shown to exhibit significantly different performance from each other. This heterogeneity creates two challenges when hosting multi-tier Web applications in the Cloud. First, different machine instances have different processing capacity so balancing equal amounts of load to different instances leads to poor performance. Second, when an application must be reprovisioned, depending on the performance characteristics of the new machine instance it may be more beneficial to add the instance to one tier or another. This paper shows how we can efficiently benchmark the individual performance profile of each individual virtual machine instance when we obtain it from the Cloud. These performance profiles allow us to balance the request load more efficiently than standard load balancers, leading to better performance at lower costs. The performance profiles also allow us to predict the performance that the overall application would have if the new machine instance would be added to any of the application tiers, and therefore to decide how to make best use of newly acquired machine instances. We demonstrate the effectiveness of our techniques by provisioning the TPC-W e-commerce benchmark in the Amazon EC2 platform.

1 Introduction

Cloud computing is an attractive platform to host Web applications. Besides the advantages of outsourcing machine ownership and system management, Clouds offer the possibility to dynamically provision resources according to an application's workload — and to pay only for the resources that are actually being used. Given a service-level objective (SLO) which states the response time guarantees an application provider wants to maintain, dynamic resource provisioning continuously adjusts the number of computing resources used to host the application. Additional capacity can be added dynamically when the load of user requests increases, and later released when the extra power is no longer necessary.

Resource provisioning for Web applications in the Cloud faces two important challenges. First, Web applications are not monolithic. At the very least a Web application is composed of application servers and database servers, which both can benefit from dynamic resource provisioning. However, the effect of reprovisioning an extra machine varies from tier to tier. When adding or removing capacity, one needs to decide which tier must be (de-)provisioned such that the performance remains within the SLO at the lowest cost. Second, computing resources in the Cloud are not homogeneous. This is obvious when considering the many different instance types in a platform such as Amazon's EC2 [1]. However, even an application which would decide to use a single instance type would not experience homogeneous performance. Figure 1(a) illustrates the performance of 30 'identical' EC2 instances when running the same application server or database server workloads [2]. Clearly, some instances are more suitable than others for efficiently running CPU-intensive application servers, but are less suitable for I/O intensive workloads. Other instances have faster I/O but slower CPU, and may be better used as database servers. Finally, we have fast machines which can be used for either of both, and slow machines which should either be given a modest task such as load balancing or de-provisioned altogether. On the other hand, Figure 1(b) shows the response time of individual instance running application server workload on EC2 over a period of 24-hours, measured at a 1-minute granularity [2]. Performance spikes occasionally occur with an average duration of 1 to 3 minutes, but overall the performance of individual instances is consistent over time. The performance spikes of an individual instance are presumably caused by the launch/shutdown opera-

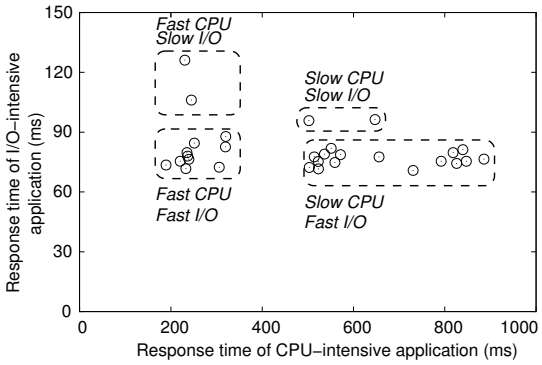
tions of the other virtual instances on the same physical machine. The same experiments run in the Rackspace Cloud show similar behavior. Similar observations are also reported for different application fields and performance metrics [8].

Efficient dynamic resource provisioning in these conditions is very difficult. To provision a Web application dynamically and efficiently, we need to predict the performance the application would have if it was given a new machine, such that one can choose the minimum number of machines required to maintain the SLO. However, because of resource heterogeneity it is impossible to predict the performance profile of a new machine instance at the time we request it from the Cloud. We therefore cannot accurately predict the performance the application would have if it was using this new machine in one of its tiers. It is therefore necessary to *profile* the performance of the new machine before deciding how we can make the best use of it.

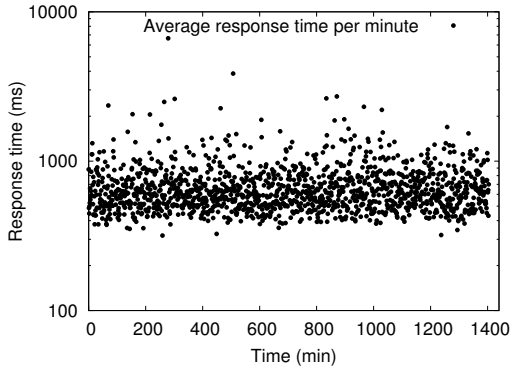
One simple profiling method would consist of sequentially adding the new machine instance to each tier of the application and measure the performance gain it can provide on each tier. However, this approach is extremely inefficient and time-consuming as profiling requires lots of time. For instance, adding a machine instance to a database tier may cost tens of minutes or more, which is not acceptable for dynamic resource provisioning.

In this paper we show how one can efficiently profile new machines using real application workloads to achieve accurate performance prediction in the heterogeneous Cloud. By studying the correlation of demands that different tiers put on the same machine, we can derive the performance that a given tier would have on a new machine instance, without needing to actually run this tier on this machine instance. This per-tier, per-instance performance prediction is crucial to take two important decisions. First, it allows us to balance the request load between multiple heterogeneous instances running the same tier so as to use each machine instance according to its capabilities. Second, when the application needs to expand its capacity it allows us to correctly select which tier of the application should be reprovisioned with a newly obtained instance.

We evaluate our provisioning algorithm in the Amazon EC2 platform. We first demonstrate the importance of adaptive load balancing in Cloud to achieve homogeneous performance from heterogeneous instances. We then use our performance prediction algorithm to drive the dynamic resource provisioning of the TPC-W e-commerce benchmark. We show that our system effectively provisions TPC-W in the heterogeneous Cloud and achieve higher throughput compared with current provision techniques.



(a) EC2 Cloud performance heterogeneity



(b) Consistent performance of individual instance over time

Figure 1: Heterogeneous Cloud performance

The rest of this paper is organized as follows: Section 2 introduces research efforts related to our work. Section 3 shows an example of scaling application on EC2 to highlight the motivation of our work. Section 4 presents the design of our resource provisioning system. Section 5 evaluates our system using both single-tier and multi-tier Web applications. Finally, Section 6 concludes.

2 Related work

A number of research efforts address dynamic resource provisioning of Web applications and model the interactions between tiers of a multi-tier Web application through analytical queueing models [9, 12, 13]. These algorithms can drive the decision to reprovision one tier rather than another for best performance. They also incorporate the effect of provisioning techniques such as caching and master-slave database replication into their provisioning models. We previously extended these works for the dynamic resource provisioning of multi-service Web applications, where Web applications are not only constructed as a sequence of tiers but can also consist of multiple services interacting with each other

in a directed acyclic graph [3]. These works however assume that the underlying provisioning machines are homogeneous. This is a reasonable assumption in medium-scale environments such as cluster computers. However, in Cloud computing platforms resources are heterogeneous so these systems do not apply.

A few research works address the problem of provisioning Web applications in heterogeneous resource environments. Stewart et al. predict the performance of Internet Services across various server platforms with different hardware capacities such as processor speeds and processor cache sizes [10]. Similarly, Marin et al. use detailed hardware models to predict the performance of scientific applications across heterogeneous architectures [6]. These approaches rely on detailed hardware metrics to parametrize the performance model. However, in the Cloud such low-level metrics are hidden by the virtualization layer. In addition, Cloud hardware resources are typically shared by virtual instances, which makes it much harder for hardware-based performance models to capture the performance features of consolidated virtual instances. These works therefore cannot be easily extended to predict Web application performance in the Cloud.

JustRunIt is a sandbox environment for profiling new machines in heterogeneous environments using real workloads and real system states [14]. When one needs to decide on the usage of new machines, this work clones an online system to new machines, and duplicate online workload to them. This approach can effectively capture performance characteristics of new virtual instances in the Cloud. However, it requires to clone online environment to new instances at each adaptation, which can be very time-consuming. On the other hand, our work can predict the performance of new instances for running Web applications without actually executing the application on new instances.

Elnikety et al. address the problem of predicting replicated database performance using standalone database profiling [4]. This work inspired us to realize performance prediction through online profiling techniques. However, it addresses a different problem than ours: here the problem is to predict the performance of different database replication techniques rather than accounting for the performance heterogeneity of the database servers themselves. Our work focuses on the latter.

Finally, instead of predicting performance, Kalyvianaki et al. use control theory to allocate CPU resources to multi-tier Web applications hosting across various virtual instances in a virtualized data center [5]. This work mainly focuses on the problem of hardware resource assignment for composing different virtual instances with different capabilities. It does not address performance impact of resource heterogeneity caused by the virtualization. Nathuji et al. also focus on providing Quality-of-Service guarantees to applications running in the Cloud [7]. However, this work aims at dynamically adapting the hardware resource allocation among consolidated virtual instances to mitigate performance interference of different applications, rather than making the best possible use of heterogeneous machine instances.

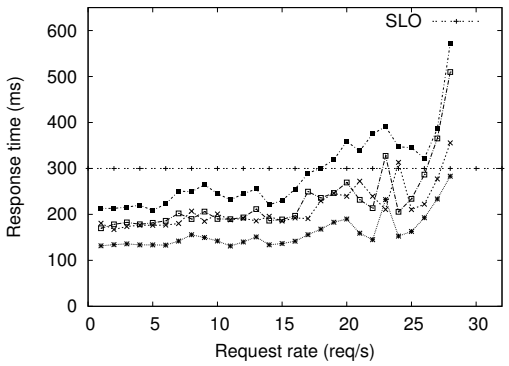


Figure 2: Web application performance heterogeneity under Auto-Scale on EC2

3 Motivating example

The performance heterogeneity of Cloud resources depicted in Figure 1 has strong consequences on Web application hosting. Figure 2 shows the response time of a single-tier CPU-intensive Web application deployed on four 'identical' virtual instances in the Amazon EC2 Cloud, using Amazon's own Elastic Load Balancer (which addresses equal numbers of requests to all instances). As we can see, the four instances exhibit significantly different performance. The response time of the first instance exceeds 300 ms around 17 req/s while the fastest instances can sustain up to 28 req/s before violating the same SLO. As a result, it becomes necessary to re-provision the application at 17 req/s, while the same virtual instances could sustain a much higher workload if they were load balanced according to their individual capabilities.

As we shall see in the next sections, the problem becomes more difficult when considering multi-tier Web applications. When re-provisioning a multi-tier Web application, one must decide which tier a new virtual instance should be added to, such that the overall application performance is maximized. However, this choice largely depends on the individual performance of the new virtual instance: an instance with fast I/O is more likely than another to be useful as a database replica, while an instance with fast CPU may be better used as an extra application server.

4 Dynamic resource provisioning

Dynamic resource provisioning for web applications in the Cloud requires one to predict the performance that heterogeneous machine instances would have when executing a variety of tiers which all have different demands for the machine instances. This performance prediction allows us to choose which tier(s) should ideally benefit from this instance for optimal performance gains of this entire application.

4.1 Solution outline

We address the problem in four steps. First, when using multiple heterogeneous machines to run a single tier, one must carefully balance the load between them to use each machine according to its capacity such that each provisioned instance features with equal response time. We discuss Web application hosting techniques and load balancing in section 4.2.

Second, we need to measure the individual performance profile of new machine instances for running specific application tiers. Benchmarking a machine instance for one tier does not generate a single measurement value, but an estimation of the response time as a function of the request rate. Profiling a tier in a machine requires some care when the tier is already used in production: we need to measure the response time of the tier under a number of specific request rates, but at the same time we must be careful so that the instance does not violate the application's SLO. We discuss profiling techniques in section 4.3.

Third, every time the application needs to provision a new machine instance, it is very inefficient to successively profile each of the application tiers on the new instance. Instead, we calibrate the respective hardware demands of different tiers of the Web application using a single 'calibration' machine instance. We also include two synthetic reference applications in the calibration. After this step, each new instance is benchmarked using the reference applications only. Thanks to the initial calibration, we can predict the performance that this particular machine instance would have if it was executing any tier of the real Web application. We discuss performance prediction in section 4.4.

Finally, knowing the performance that a new machine instance would have if we added it to any tier of the application, we can choose the tier where it would generate the greatest performance improvement for the overall application. We choose the targeted tier by modeling the whole Web application as a queueing network where each tier acts as a separate queue. We discuss the queueing model for provisioning tiers in section 4.5.

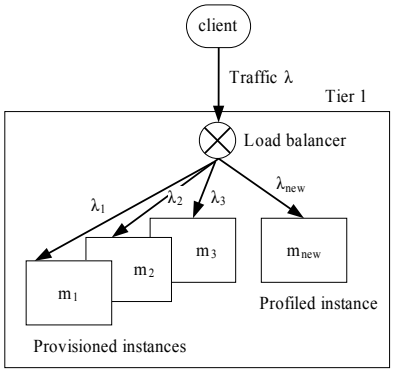


Figure 3: Web application hosting in the Cloud

4.2 Web application hosting

Figure 3 shows the typical hosting architecture of a single tier of the Web application. The provisioned virtual instances m_1, m_2 and m_3 host either the replicated application code if this is an application server tier, or a database replica if this is a database tier. As the performance of provisioned instances largely differs from each other, it would be a bad idea to address the same request rate to all instances. A much better option is to carefully control the respective load of each instance such that they all exhibit the same response time. In this scenario, fast machine instances get to process more requests than slower ones.

We control the workload by deploying a custom load balancer in front of the provisioned instances. To guarantee backend instances to serve with equal response time, the load balancer calculates the weighted workload distribution according to their performance profiles by solving the following set of equations:

$$\begin{cases} \lambda = \lambda_1 + \cdots + \lambda_n \\ r = \mathit{perf}(\mathit{instance}_1, \lambda_1) \\ \cdots \\ r = \mathit{perf}(\mathit{instance}_n, \lambda_n) \end{cases} \quad (1)$$

where λ is the total request rate seen by the load balancer, $\lambda_1, \ldots, \lambda_n$ are the request rates addressed to each provisioned instance respectively and r is the uniform response time of all provisioned instances. The $\mathit{perf}()$ functions are typically defined as a set of measured points, with linear interpolation between each consecutive pair of points.

This set of *n+1* equations can be easily solved to find the values of *r*, $\lambda_1, \ldots, \lambda_n$. The load balancer uses these values as weights of its weighted Round-Robin strategy.

When adding a new instance m_{new} into this tier, the load balancer thus needs to know the performance profile of this new instance such that it can balance the workload

accordingly. This is the goal of instance profiling that we discuss in next.

4.3 Online profiling

Coming up with a machine instance's own performance profile when provisioning a given tier can be done in two different ways: either we measure the actual profile using real request traffic, or we derive the profile from other measured profiles. This section discusses the former.

Profiling a machine instance with a given tier workload consists in deploying the tier service on the machine instance, then addressing traffic with various load intensities to measure the corresponding response times.

We approximate the actual profile of a new instance by measuring performance at carefully selected workload intensities, and using linear interpolation between each consecutive pair of measured points. The output of the online profiling of a new instance is therefore a set of *n* linear functions which cover consecutive workload ranges as follows:

$$r_i = a_i \times \lambda_i + b_i \qquad (1 \leq i \leq n) \tag{2}$$

where *n* is the total number of the consecutive workload ranges, and *r*, λ, *a* and *b* respectively represent average response time, request rate and linear parameters within the workload range *i*.

Generating a performance profile for a synthetic application is relatively easy: one only needs to deploy the synthetic application on the tested machine, and use a separate machine instance to generate a standardized workload and measure the tested instance's performance profile.

Generating a similar performance profile for a real tier of the Web application is harder. We want to address traffic that is as realistic as possible to increase the accuracy of the performance profile. Metrics which make a traffic workload realistic include the respective proportions of simple and complex requests, read/write ratio, popularity distribution of different data items, and so on. All these properties have a significant impact on performance. Instead of trying to synthesize a realistic workload, we prefer to provision the new instance in the tier to profile, and address real live traffic to it (which is realistic by definition).

Profiling a machine instance using live traffic however requires caution. First, we must make sure that profiling this instance will not create an SLO violation for the end users whose requests are processed by the profiled machine instance. For instance, one could simply replace one of the current instances used in the tier with the instance to profile. However, if the new instance is slower than the previous one, the application may violate its SLO. Second, we want to test specific workload intensities regardless of the actual workload received by the tier at the time of the profiling. Profiling a live tier therefore requires careful load balancing where we control the request rate addressed to the profiled instance.

We first need a rough estimation of the variety of performance profiles from one instance to another. Such variety is specific to one Cloud provider, as it largely depends on the consolidation strategies and virtualized performance isolation that the Cloud implements. We calculate the performance variety rate *N* as follows.

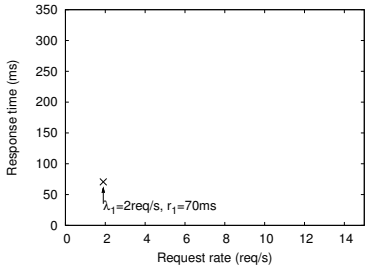
$$N = \frac{T_{max}}{T_{min}} \tag{3}$$

where T_{max} and T_{min} respectively represent the throughput of the fastest and slowest instances in the Cloud when running a given Web application. We set a SLO defining the maximum response time to this Web application. We measure the throughput of this Web application when it violates the SLO. The tested application exhibits either CPU-intensive or I/O-intensive workload for estimating the CPU and IO performance variety separately. For instance, in Amazon EC2 Cloud we observed $N_{CPU} \approx 4$ for CPU-intensive tiers such as application servers and $N_{I/O} \approx 2$ for I/O-intensive tiers such as database servers [2]. Similarly, in Rackspace we observed $N_{CPU} \approx 1.5$ and $N_{I/O} \approx 4$. In a new Cloud platform, one would need to sample a sufficient number of instances to evaluate these numbers.

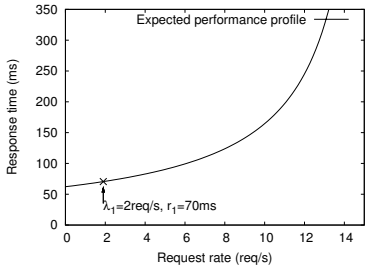
Second, we carefully choose different workload intensities to address to the new machine. One needs to choose the key performance points (λ, r) that represent significant features of the performance profile. For instance, the performance profile of a new instance under low workload can be approximated as a constant value regardless of the load. We ideally want a number of points varying from underload to overload situations, preferably at the inflection points and close to the SLO. The accuracy of the approximated curve increases with the number of measured points. However, this also increases the profiling time.

Figure 4 illustrates our strategy to select the request rate for profiling the new instance. We first address the new instance with request rate $\lambda_1 = \frac{\lambda_{max}}{N}$, where λ_{max} is the current request rate of the fastest instance currently used in this tier. Assuming our estimation of performance variety N is correct, the profiled instance cannot violate the SLO even if it happens to be very slow. This gives us the first performance point (λ_1, r_1) as illustrated in Figure 4(a)

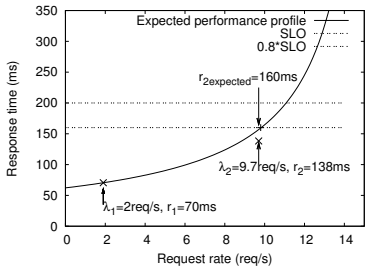
Using this first measurement, we can use queueing theory to generate a first estimate of the entire performance profile of this instance. If we model the tier as an M/M/n queue, then the instance's service time can be computed as:



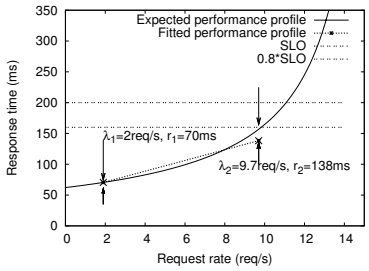
(a) First measurement point selected such that the instance will not violate its SLO



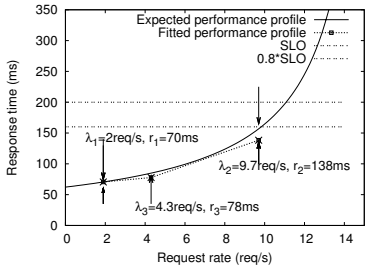
(b) First estimation of the instance's profile thanks to queueing theory



(c) Selection of a second measurement point close to the SLO



(d) Fit performance profile of the new instance



(e) Correction of the performance profile of the new instance

Figure 4: Online profiling process

$$s = \frac{r_1}{1 + \frac{\lambda_1 \times r_1}{n}} \tag{4}$$

where n is the number of CPU cores of this machine (as announced by the Cloud provider). The service time is the response time of the instance under low workload where the effects of request concurrency are negligible. It indicates the capability of the instance to serve incoming requests. We can then use this service time to derive a first performance profile of the new instance as follows:

$$r(\lambda) = \frac{s}{1 - \frac{\lambda \times s}{n}} \tag{5}$$

Figure 4(b) shows the first performance profile of the new instance derived based on the calculated service time. One should however note that this profile built out of a single performance value is very approximate. For a more precise profile, one needs to measure more data points.

Using this profile, we can now calculate a second workload intensity which should bring the instance close to the SLO. We select an expected response time r_2, then derive the workload intensity which should produce this response time.

$$r_2^{expected} = 0.8 \times SLO \tag{6}$$

$$\lambda_2 = \frac{n \times (r_2^{expected} - s)}{r_2^{expected} \times s} \tag{7}$$

Here we set the target response time to 80% of the SLO to avoid violating the SLO of the profiled instance even though the initial performance profile will feature relatively large error margins. We can then address this workload intensity to the new instance and measure its real performance value (λ_2, r_2). As shown in Figure 4(c), the real performance of the second point is somewhat different from the expected 80% of the SLO.

We apply linear regression between the two measured points (λ_1, r_1), (λ_2, r_2) and get the fitted performance profile of the new instance as shown in Figure 4(d). We then let the load balancer calculate the weighted workload distribution between the provisioned instance and the new one.

By addressing the weighted workload intensities to the two instances, we can measure the real response time of the new instance. However, as shown in Figure 4(e), the real performance of the new instance differs slightly from the expected one in Figure 4(d) due to the approximation error of its initial profile. We then correct the performance profile of the new instance by interpolating the third performance point(λ_3, r_3). We show that the above strategy is effective to profile heterogeneous virtual machines and provision single services in Section 5.

Although expressing performance profiles as a function of request rate is useful for load balancing, for performance prediction we need to express performance profiles as a function of CPU utilization (for application server tiers) or I/O utilization (for database server tiers). When profiling a machine instance, we also measure the relevant metrics of resource utilization, and use the exact same technique to build performance profiles that are suitable for performance prediction.

4.4 Performance prediction

To efficiently select the tier in which a new instance will be most valuable to the application as a whole, we first need to know the performance profile of this instance when running each of the application's tiers. A naive approach would be to successively measure this profile with each tier one by one before taking a decision. However, this strategy would be too slow to be of any practical use. Indeed, profiling a new machine with a real application tier requires to first replicate the hosted service to the new machine. For instance, when profiling a new machine for a database tier, one needs to first replicate the entire database to the new machine before starting the profiling process. Replicating a medium-sized database can easily take tens of minutes, and this duration increases linearly with the database size. We therefore need to be able to quickly *predict* the performance profiles, without needing to actually replicate the database.

We found that the most characteristic feature of a virtual instance to predict the performance profile of a given tier in this instance is its resource utilization. Although the absolute response time of two different tiers in the same machine under the same CPU or I/O utilization are not identical, they are highly correlated.

We illustrate this in Figure 5. Each point in this graph represents the response times of two different application server tiers running in the same machine instance, and having the same CPU utilization (respectively 15%, 25%, 65% and 80%). The request rates necessary to reach a given CPU utilization varies from one application to the next. We however observe that the points form an almost perfect straight line. This allows us to derive new performance profiles from already known ones. The same observation is also true for database server tiers, taking the disk I/O bandwidth consumption as the resource utilization metric.

Given the response time and resource utilization of one tier in a given machine instance, we can infer the response time of the second tier in the same machine instance under the same resource utilization. Figure 6 illustrates the input and output of this prediction: we predict the performance of tier 1 and tier 2 on a new machine by correlating their performance and the reference application performance on a calibration machine.

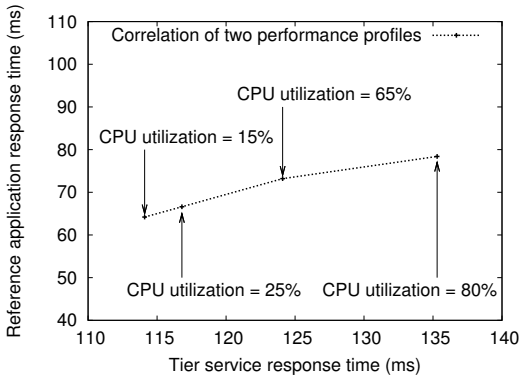


Figure 5: Performance correlation between reference application and tier service

Input:
$perf(machine_{calibration},\ app_{ref}) = f(load)$
$perf(machine_{calibration},\ app_{tier1}) = f(load)$
$perf(machine_{calibration},\ app_{tier2}) = f(load)$
$perf(machine_{new},\ app_{ref}) = f(load)$

Output:
$perf(machine_{new},\ app_{tier1}) = f(load)$
$perf(machine_{new},\ app_{tier2}) = f(load)$

Figure 6: Input and output of the performance profile prediction

First, we need to measure the application-specific demands of each tier of the application. This has to be done only once per application. This profiling should be done on a single calibration machine, which can be any particular virtual instance in the Cloud. To predict the performance of any particular tier on a new instance quickly, we also benchmark the calibration machine using two synthetic reference applications which respectively exhibit CPU-intensive features characteristic of application servers, and I/O-intensive features characteristic of database servers. The Ref_{CPU} application receives customer names and generates detailed personal information through CPU-intensive XML transformation. The $Ref_{I/O}$ application searches for items related to a customer's previously ordered items from a large set of items. The operations of the reference applications introduce typical CPU-intensive and disk I/O-intensive workloads. The reference applications can be deployed very quickly on any new machine instance, for example by including it to the operating system image loaded by the virtual machine instances. We use Ref_{CPU} as a reference point to predict the performance profiles of application server

tiers, and $Ref_{I/O}$ as a reference point to predict the performance profiles of database tiers.

Profiling the same calibration machine instance with one of the Web application's tiers and the corresponding reference application allows us to learn the relationship between the demands that these two applications put on the hardware:

$$perf(app_{tier}, utilization) = \alpha \times perf(app_{ref}, utilization) + \beta$$

The same relationship between the response times of the two applications, captured by the values of α and β, remains true on other machine instances. Knowing the performance profile of the reference application on a newly obtained virtual machine instance from the cloud, we can thus derive the predicted performance profile of *tier1* on the new instance, even though we never even installed this particular tier on this particular instance.

4.5 Resource provisioning

When provisioning a multi-tier Web application, upon a violation of the service-level objective, one needs to decide which tier to re-provision within the whole application. Once a new instance is acquired and profiled, one needs to perform a simple what-if analysis to predict the performance improvement that the whole application would observe if this instance was added in one of the tiers. For simplicity, in this paper we apply our Cloud instance profiling methods to simple two-tier Web applications only. Other performance models of composite Web applications can be used to extend this work to more complex setups [3, 13].

The response time of a two-tier Web application can be computed as follows.

$$R_{app} = R_1 + N_{1,2} \times R_2 \quad (8)$$

where R_{app} is the response time of the whole application, R_1, R_2 are response time of the application server tier and the database tier respectively, $N_{1,2}$ is the request ratio that equals to the request number seen by the second (database) tier caused by one request from the first (application server) tier.

Given the performance profiles of the new instance for each of the application's tiers, we can issue a simple "what-if" analysis: we first use the performance profiles to compute the new application performance if the new instance was added to the first tier, then if it was added to the second tier. The best usage of the new instance is defined as the one which maximizes the application's performance.

5 Experimental evaluation

In this section we evaluate the effectiveness and efficiency of our resource provisioning algorithm for provisioning Web applications on the Amazon EC2 platform.

5.1 Experiment setup

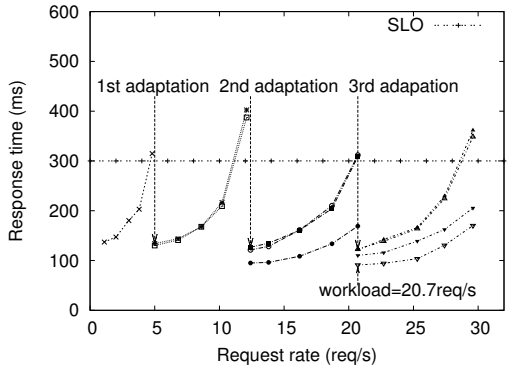
The bulk of our system implementation lies in our custom layer-4 load balancer. In addition to distributing requests to backend servers, the load balancer also profiles new machines when we obtain them from the cloud. By deploying the load balancer in front of the tiers of Web applications, our system can provision Web applications over heterogeneous instances in the Cloud.

We evaluate our resource provisioning algorithm using three Web applications. The first two are the reference applications Ref_{CPU} and $Ref_{I/O}$. The last one is the TPC-W Web application benchmark. This benchmark is structured as a two-tiered application which models an online bookshop like Amazon.com [11]. We run all our experiments in Amazon EC2 platform using small instances.

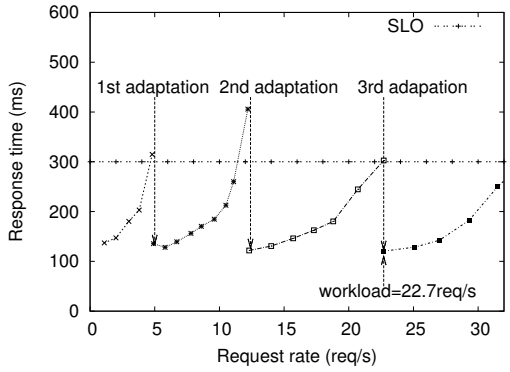
5.2 Importance of adaptive load balancing

We first demonstrate the importance of adaptive load balancing in Cloud using Ref_{CPU} and $Ref_{I/O}$. We deploy each application on a single machine instance, and increase the workload gradually. We set the SLO of the response time of Ref_{CPU} and $Ref_{I/O}$ to 300 ms and 500 ms respectively. We run each experiment using two different setups. First, we use Amazon's Elastic Load Balancer to distribute the traffic between the instances, and Amazon's AutoScale to provision new virtual machine instances when the SLO is violated. Second, we run the same experiment using the exact same instances with our system. Both applications are single-tiered, so here we exercise only the capability of load balancing to adapt to heterogeneous resources.

Figure 7 shows the response time per machine instance running the Ref_{CPU} application. When using the Elastic Load Balancer (ELB), at 5 req/s the system violates the SLO and therefore provisions a new instance. By coincidence, the second instance has a performance profile very close to the first one so they exhibit extremely similar performance. However, after the second and third adaptation we see that different instances exhibit different performance. On the other hand, our system balances the workload such that all instances always exhibit the same performance. This has important consequences in terms of resource usage: when using ELB, the one of the application instances violates its SLO at 20.7 req/s, triggering a request for a fourth instance. When using

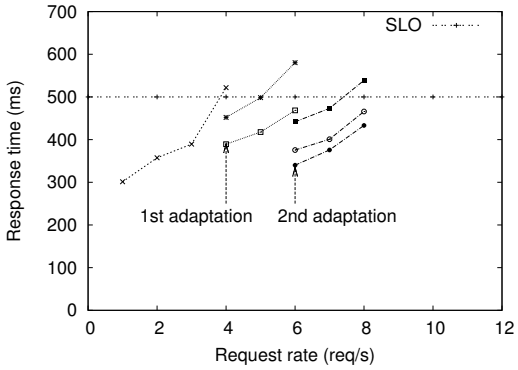


(a) Using Amazon's Elastic Load Balancer

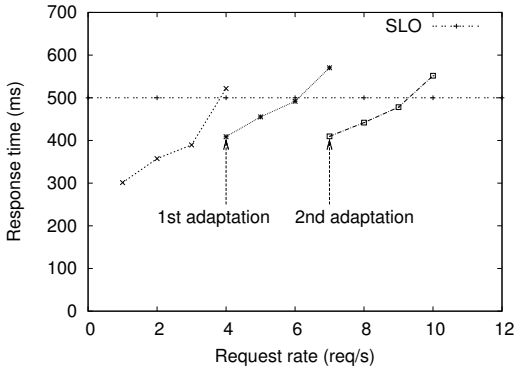


(b) Using our system

Figure 7: Provisioning Ref_{CPU} under increasing workload



(a) Using Amazon's Elastic Load Balancer



(b) Using our system

Figure 8: Provisioning $Ref_{I/O}$ under increasing workload

our system, the third instance (a very fast one) is given a higher workload than the others so the system requires a fourth instance only above 22.7 req/s.

Figure 8 shows similar results for the $Ref_{I/O}$ application. Here as well, our system balances traffic between instances such that they exhibit identical performance, whereas ELB creates significant performance differences between the instances. Our system can sustain up to 9 req/s when using three instances, while ELB can sustain only 7 req/s.

These results show that one should employ adaptive load balancing to correctly assign weights to forwarding instances when distributing traffics in Cloud. By doing so, one can achieve homogeneous performance from heterogeneous instances and make more efficient usage of these instances.

5.3 Effectiveness of Performance Prediction and Resource Provisioning

We now demonstrate the effectiveness of our system to provision multi-tier Web applications. In this scenario, in addition to using our load balancer, we also need to predict the performance that each new machine would have if it was added to the application server or database server tiers to decide which tier a new instance should be assigned to. The Amazon cloud does not have standard automatic mechanisms for driving such choices so we do not compare our approach with Amazon's resource provisioning service.

We use our system to provision the TPC-W e-commerce benchmark using the "shopping mix" workload. This standard workload generates 80% of read-only interactions, and 20% of read-write interactions. We set the SLO of the response time of TPC-W to be 500 ms. We increase the workload by creating corresponding numbers of Emulated Browsers (EBs). Each EB simulates a single user who browses the application. Whenever an EB leaves the application, a new EB is automatically create to maintain a constant load.

When the overall response time of the application violates the SLO, we request a new instance from the Cloud and profile it using the reference application. Thanks to the performance correlations between the tiers of TPC-W and the reference application, we use the performance profile of the new instance to predict the performance of any tier of TPC-W if it was using the new instance.

Table 1: Prediction accuracy during the first experiment run

	Adapt at 90 EBs			Adapt at 160 EBs			Adapt at 210 EBs			Adapt at 270 EBs		
	Real	Predicted	Error	Real	Predicted	Error	Real	Predicted	Error	Real	Predicted	Error
Provision the AS tier	554.6 ms	596.7 ms	+7.6%	578.3 ms	625.1 ms	+8.1%	458.3 ms	490.1 ms	+6.9%	**232.5 ms**	**248.3 ms**	+6.8%
Provision the DB tier	**165.4 ms**	**188.1 ms**	+13.7%	**189.7 ms**	**203.4 ms**	+7.2%	**156.2 ms**	**166.4 ms**	+6.5%	313.4 ms	329.1 ms	+5.0%

Table 2: Prediction accuracy during the second experiment run

	Adapt at 60 EBs			Adapt at 130 EBs			Adapt at 220 EBs			Adapt at 300 EBs		
	Real	Predicted	Error	Real	Predicted	Error	Real	Predicted	Error	Real	Predicted	Error
Provision the AS tier	511.7 ms	567.3 ms	+10.9%	427.9 ms	453.7 ms	+6.0%	**177.5 ms**	**192.3 ms**	+6.9%	541.9 ms	579.2 ms	+6.9%
Provision the DB tier	**152.4 ms**	**168.2 ms**	+10.4%	**218.2 ms**	**230.7 ms**	+5.7%	281.4 ms	302.7 ms	+7.6%	**151.2 ms**	**163.2 ms**	+7.9%

Finally, we compare the performance gains if the new instance was assigned to different tiers of TPC-W and select the tier which gives most performance benefit. We run the entire experiment twice: our provisioning system takes different decisions depending on the characteristics of the machine instances it gets from the Cloud.

Figures 9(a) illustrates the response time of TPC-W in the first run of the experiment. The application violates its SLO around a workload of 90 EBs. We request a new instance from the Cloud, profile it, and predict that it would be most useful if it was assigned to the database tier. When we push the workload further, it adds another database server at 160 EBs, then yet another database server at 210 EBs, then finally an application server at 270 EBs.

Figure 9(b) shows that, if we run the exact same experiment a second time, the machine instances we obtain from the Cloud have different performances. This leads the resource provisioning to take different decisions. It adds two database servers respectively at 60 and 130 EBs, then an application server at 220 EBs, then another database server at 300 EBs.

We can see here that SLO violations occur at different workloads, depending on the performance of the machine instances running the application. We also see that our resource provisioning effectively distinguishes different performance profiles, and takes provisioning decisions accordingly. In particular, at the third adaptation, the first run decides to use the new machine instance as a database server while the second run decides to use its own new machine instance as an application server.

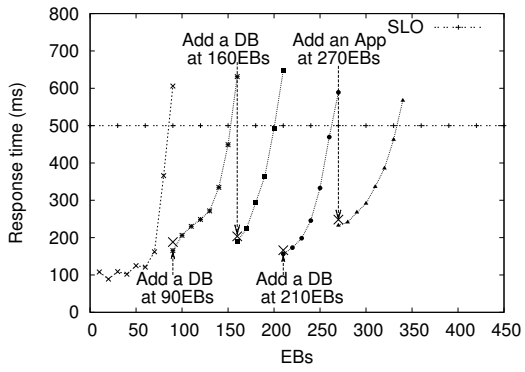
At each adaptation point, the resource provisioning system issues two predictions: it predicts what the new response time of the overall application would be if we assigned the new machine instance to be an application server or a database server. At each adaptation point we also tested the accuracy of these two predictions by deploying each of the two tiers in the new instances and measuring the actual application performance. Tables 1 and 2 show the measured and predicted response times of the whole application at each adaptation point. We can see that all predictions remain within 14% of the measured response times. This level of accuracy is sufficient to take correct provisioning decisions: in this set of experiments, the provisioning always identifies the best use it can make of the new machine instance it received (written in bold text in the table).

5.4 Comparison with other provision techniques

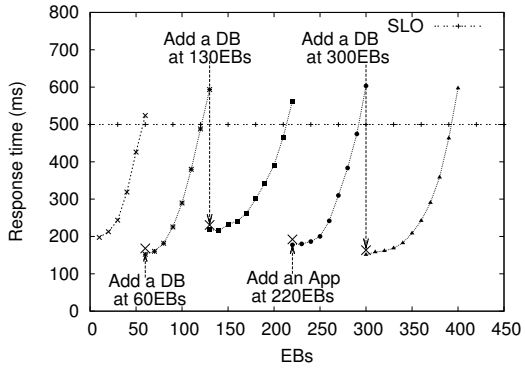
So far we showed the effectiveness of our system to provisioning TPC-W on EC2 by assigning heterogeneous instances to the tier where it gives maximum performance gain. We now demonstrate that our system can improve the throughput of TPC-W running on EC2 compared with two other provisioning techniques: "Homogeneous Provisioning" and "Worst-case Provisioning".

"Homogeneous Provisioning" provisions instances assuming that the performance of these instances is homogeneous. "Homogeneous Provisioning" first profiles the performance of the first two virtual instances hosting TPC-W. At each adaptation, "Homogeneous Provisioning" predicts the performance gains of new instances at each tier using the initial performance profiles, and assigns a new instance to the tier which receives maximum performance gain. "Homogeneous Provisioning" dispatches requests between instances using the round-robin policy. "Worst-case Provisioning" employs our algorithm to first figure out the tier to which a new instance should be assigned. However, "Worst-case Provisioning" systematically adopts the worst possible option. For instance, "Worst-case Provisioning" assigns a new instance to the application server tier if our system decides to assign this instance to the database tier. "Worst-case Provisioning" employs the same load balancing in our system. For comparison, we name our system as "Adaptive Provisioning".

We first use the three techniques to provision TPC-W on EC2 with increasing workload separately. We set the SLO to 500 ms and measure the maximum throughput that a system configuration can sustain before violat-

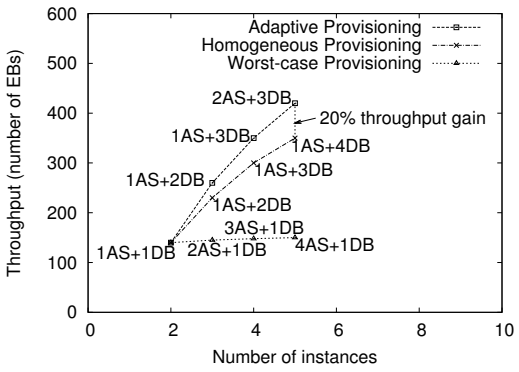


(a) First group

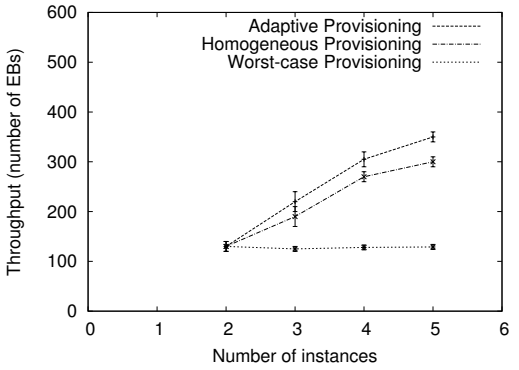


(b) Second group

Figure 9: Provisioning TPC-W under increasing workload



(a) Throughput comparison of single round



(b) Statistical comparison of throughput over multiple rounds

Figure 10: Throughput comparison of three provisioning techniques

ing the SLO. We also record the instance configurations at each adaptation. Figure 10(a) shows the throughputs achieved by each provisioning technique during a single run of the system under increasing workload and the corresponding instance configurations at each adaptation. The three provisioning systems use the exact same instances in the same order so their respective performance can be compared.

"Worst-case Provisioning" decides to provision the application server tier at each adaptation and finally supports around 150 EBs with 5 instances. "Homogeneous Provisioning" and "Adaptive Provisioning" both decide to provision new instances to the database server tier at the first and second adaptation. However, they achieve different throughput at the first two adaptations. The throughput difference is caused by the different load balancing capability of adapting to heterogeneous instances used in each provision technique. At the third adaptation, "Adaptive Provisioning" decides to assign the new instance to the application server tier while "Homogeneous Provisioning" decides to assign the same new instance to the database server tier. After the third adaptation, "Adaptive Provisioning" supports around 420 EBs while "Homogeneous Provisioning" supports around 350 EBs. This represents a 20% gain in throughput.

We then run the same experiment 5 rounds, each with a different set of EC2 small instances. Within each round, we measure the throughput achieved by each provisioning technique using a certain number of instances. The throughputs achieved in different rounds are different due to the performance heterogeneity of small instances. Figure 10(b) shows the average and standard deviation of the throughput achieved by each provisioning technique across multiple rounds. As previously, the "Worst-case Provisioning" behaves as the statistical lower bound of the achievable throughput of TPC-W on EC2. When taking the first adaptation, "Adaptive Provisioning" and "Homogeneous Provisioning" behave similar in terms of achieved throughput. However, when taking more adaptations, "Adaptive Provisioning" supports 17% higher throughput than "Homogeneous Provisioning". This demonstrates that our system makes more efficient use of heterogeneous instances in Cloud and achieves higher throughput using the same resources.

6 Conclusion

Cloud computing provides Web application providers with an attracting paradigm to dynamically vary the number of resources used by their application according to the current workload. However, Cloud computing platforms also have important limitations. In particular, dynamic resource provisioning is made difficult by the fact that each virtual instance has its own individual performance characteristics. Standard resource provisioning techniques provided by Cloud platforms do not take this performance heterogeneity into account, and therefore end up wasting resources.

We demonstrated in this paper that taking performance heterogeneity into account in a resource provisioning system can be practical and bring significant resource savings. One must first capture the performance relationships between different tiers of an application. When the application's workload makes it necessary to provision a new instance, we can efficiently capture its own performance profile, and use this information to drive the resource provisioning decisions: first, it allows us to decide to which tier this new machine instance should be assigned. Second, it allows us to adjust load balancing to make better use of the processing resources of each machine instance.

We hope that these results will allow the creation of new Cloud products such as automated performance monitoring and prediction as a service, and performance-aware load balancers. Providing Cloud users with such tools would allow them to make more efficient use of Cloud resources, and would thereby further increase the attractiveness of Cloud technologies for Web application providers.

7 Acknowledgments

Chi-Hung Chi is supported by the National Natural Science Foundation of China, Project Number 61033006.

References

[1] Amazon EC2: Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[2] DEJUN, J., PIERRE, G., AND CHI-HUNG, C. EC2 performance analysis for resource provisioning of service-oriented applications. In *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing* (Nov. 2009). LNCS 6275.

[3] DEJUN, J., PIERRE, G., AND CHI-HUNG, C. Autonomous resource provisioning for multi-service web applications. In *Proceedings of the 19th Intl. World Wide Web Conference* (Apr. 2010).

[4] ELNIKETY, S., DROPSHO, S., CECCHET, E., AND ZWAENEPOEL, W. Predicting replicated database scalability from standalone database profiling. In *Proceedings of the 4th EuroSys Conference* (Apr. 2009).

[5] KALYVIANAKI, E., CHARALAMBOUS, T., AND HAND, S. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using Kalman filters. In *Proceedings of the ICAC Conference* (June 2009).

[6] MARIN, G., AND MELLOR-CRUMMEY, J. Cross-architecture performance predictions for scientific applications using parameterized models. *Proceedings of the SIGMETRICS Conference* (June 2004).

[7] NATHUJI, R., KANSAL, A., AND GHAFFARKHAH, A. Q-Clouds: Managing performance interference effects for QoS-aware clouds. In *Proceedings of the 5th EuroSys conference* (Apr. 2010).

[8] OSTERMANN, S., IOSUP, A., YIGITBASI, N., PRODAN, R., FAHRINGER, T., AND EPEMA, D. A performance analysis of EC2 cloud computing services for scientific computing. In *Proceedings of the CloudComp conference* (Oct. 2010).

[9] SIVASUBRAMANIAN, S. *Scalable hosting of web applications*. PhD thesis, Vrije Universiteit Amsterdam, the Netherlands, Apr. 2007.

[10] STEWART, C., KELLY, T., ZHANG, A., AND SHEN, K. A dollar from 15 cents: Cross-platform management for internet services. In *Proceedings of the USENIX Annual Technical Conference* (June 2008).

[11] TPC-W: A transactional web e-commerce benchmark. http://www.tpc.org/tpcw.

[12] URGAONKAR, B., PACIFICI, G., SHENOY, P., SPREITZER, M., AND TANTAWI, A. An analytical model for multi-tier internet services and its applications. In *Proceedings of the SIGMETRICS Conference* (June 2005).

[13] URGAONKAR, B., PRASHANT, S., ABHISHEK, C., PAWAN, G., AND TIMOTHY, W. Agile dynamic provisioning of multi-tier internet applications. *ACM Transaction on Autonomous Adaptive System* (Mar. 2008).

[14] ZHENG, W., BIANCHINI, R., JANAKIRAMAN, G. J., SANTOS, J. R., AND TURNER, Y. JustRunIt: Experiment-based management of virtualized data centers. In *Proc. of the USENIX Annual Technical Conference* (June 2009).