THE UNIVERSITY OF SYDNEY

SCHOOL OF INFORMATION TECHNOLOGIES

HOW A CONSUMER CAN MEASURE ELASTICITY FOR CLOUD PLATFORMS

TECHNICAL REPORT 680

SADEKA ISLAM, KEVIN LEE, ALAN FEKETE, ANNA LIU

AUGUST, 2011

How A Consumer Can Measure Elasticity for Cloud Platforms

Sadeka Islam[1,2], Kevin Lee[1,2], Alan Fekete[3], Anna Liu[1,2]

[1] National ICT Australia, Sydney
firstname.lastname@nicta.com.au
[2] University of New South Wales, Sydney
[3] University of Sydney, Sydney
firstname.lastname@sydney.edu.au

Abstract. One major benefit claimed for cloud computing is elasticity: the cost to a consumer of computation can grow or shrink with the workload. This paper offers improved ways to quantify the elasticity concept, using data available to the consumer. We define a measure that reflects the financial penalty to the consumer, from under-provisioning (leading to unacceptable latency or other QoS for requests) or over-provisioning (paying more than necessary for the resources). We propose to compute this penalty measure for a suite of different time-varying workloads, and thus define a single figure that is an overall score for the relative elasticity of two platforms. We have applied these workloads to a public cloud; from our experiments we extract insights into the characteristics of a platform that influence its elasticity. We explore the impact of the rules used to increase or decrease capacity. In particular, we show that the default rules of a widely-used platform give poor elasticity, and that other rules do better on our measure.

Keywords: Cloud Computing, Elasticity, Performance Measures

1 Introduction

Cloud computing platforms are attracting a lot of attention. They are already used extensively by some innovative companies with huge and rapidly growing computational needs, and traditional enterprises are looking closely at the cloud as an addition or even an alternative to running their own IT infrastructure. Many features of cloud platforms are attractive; for example, cloud platforms may be low-cost by exploiting economies from vast scale (cheap energy, low rates of system administration labor, etc) and they may achieve high availability by extreme replication and distribution. One feature that is commonly a powerful selling point for cloud platforms is the claim that they are *elastic*; the user can pay only for what they need at a given time. In particular, a startup facing rapid growth in its business might structure its costs so they also start small, and grow as and when the income arrives to match. In contrast, traditional data processing requires a large up-front capital expenditure to buy and install IT

systems. Furthermore, traditionally, the cost must cover enough processing for the anticipated and the hoped-for growth; this leaves the company bearing much risk from uncertainty in the rate of growth. If growth is slower than expected, the revenue won't be available to pay for the infrastructure, while if growth is too fast, the systems may reach capacity and then a very expensive rewrite and rebuild is needed. Also, it is common in web-based companies for demand to be periodic or even bursty. The workload may grow very rapidly when the idea is "hot" (or "cool"), but fads are fickle and so demand can then shrink back to a previous level. Traditional infrastructure must try to provision for the peak, and so it risks wasting resources after the peak has passed. In summary, elasticity can remove risk from a startup or enterprise, by allowing "pay-as-you-grow" computing infrastructure where the costs adjust smoothly to rising (and perhaps falling) workload.

Just as with traditional IT infrastructure, the company seeking to use a cloud platform needs some basis for comparing different offerings, and choosing the one that will be best for its needs. The usual approach is to follow a benchmark, which includes a standardised workload, and defines exactly what and how to measure the behaviour of any system when subjected to this workload. The benchmark gives one (or a few) summary numbers that represent the value to the chooser of the system; it is then reasonable to choose the system with best benchmark results. Organisations like TPC and SPEC have a range of benchmarks for hardware and/or software systems such as databases, CPUs or application servers. This paper is part of an effort in the research community to define appropriate measures that will allow comparing the desirableness of competing cloud platforms. However, the cloud platform works differently than a traditional database or network, and so a different way of measuring is needed. Florescu and Kossmann [6] remark that for cloud platforms, the key question is no longer how much performance can be obtained from a given amount of resources, but instead

> Given performance requirements of an application (peak throughput; maximum tolerable response times), minimise the required hardware resources and maximise the data consistency.

While all the cloud offerings claim elasticity as a virtue that they possess, we can be sure that none are perfect in letting a customer pay for exactly what they need, and no more. There may be a minimum charge (e.g., 1 instance), there may be delays in adapting the platform to sudden increase in workload, and so on. Thus we would want to know *how elastic* is each system. As yet, the literature does not contain any explicit measurement to quantify the amount of elasticity in a platform. This is the gap that our paper addresses.

We draw attention to the difference between elasticity and scalability, since there are existing proposals to measure the latter characteristic of cloud platforms. Scalability is defined as the ability of a system to meet a larger workload requirement by adding a proportional amount of resources. This is part of what elasticity needs, but perfect scalability does not ensure perfect elasticity. Time is a central aspect in elasticity, which depends on the speed of response to changed

workload, while scalability allows the system as long as it needs to meet the changed load. Also, good elasticity requires that the system reduce costs when workload decreases, while scalability just considers growth.

One important feature of our work is that we regard benchmarking as a process done by the consumer of cloud services. We limit ourselves to running a benchmark as a consumer, and taking observations that are available to the consumer through the platforms API (or directly inside the user's application code). This makes our task harder, since we do not have access to arbitrary measurements of the infrastructure itself, but this viewpoint is necessary for our work to give the consumer a reasonable basis for choosing between competing platforms.

A key idea in our proposal to measure elasticity of a platform is to use a mix of workloads, that vary over time in different ways. Some workloads will rise and fall repeatedly, others will rise rapidly and then fall back slowly, etc. For each workload, we examine the way a platform responds to this, and we quantify the impact on the consumer's finances. That is, we use a cost measure in dollars per hour, with a component that captures how much is wasted by paying for resources that are not needed at the time (overprovisioning), and a component to see how much the consumer suffers (opportunity cost) when the system is underprovisioned, that is, the platform is not providing enough resource for a recent surge in workload.

From our measurements, we have discovered several characteristics of a cloud platform that are important influences on the extent of elasticity. Some of these (such as the speed of responding to a request for increased provisioning) have already been discussed by practitioners, but others seem to have escaped attention. For example, we find in Amazon EC2 that there is a large improvement of elasticity from relatively simple changes in the set of rules that the system uses to control provisioning and deprovisioning. There is a set of rules (based on recent utilisation rates) that is widely followed, perhaps because it is done that way in tutorial examples. We find that this leads to rapid deprovisioning when load decreases, which leaves the system underprovisioned if a future upswing occurs. Because the financial impact from poor QoS (when demand can't be handled) is generally much more severe than the cost of running some extra resource for a while, this is a poor strategy. What is worse, on typical platforms one pays for an instance in quanta that represent a significant period of time (say 1 hr), so eagerness to deprovision can leave the consumer paying for a resource without the ability to use it.

The key contributions that this paper makes are (i) a novel framework for measuring elasticity, that can be run by a consumer and which takes account of the consumer's particular business situation (ii) a specific pair of case studies, using one set of workload patterns and financial assumptions, to show that this approach can be carried out in practice, (iii) insights that we gained from the case studies, especially concerning the main internal characteristics of a platform that impact on the elasticity experience of a consumer.

This paper is structured as follows: Section 2 identifies the relevant work from the literature on elasticity and cloud performance measurements. Section 3 presents our proposed benchmark for measuring elasticity, first as a general framework based on penalties that are expressed in monetary units, and then with specific choices for penalty functions, workload curves etc. Section 4 describes details of the experimental setup including the tools and specific cloud technologies used in our case studies. Section 5 presents empirical case studies that show the elasticity of two particular platforms, given by different choices of rulesets that control provisioning decisions in a widely-used public cloud. We close the paper with a discussion of the lessons learnt and a conclusion in Section 6 and Section 7 respectively.

2 Related Work

The main objective of this paper is to provide a way for a consumer to measure how well (or not) each cloud platform delivers the elasticity property. The most relevant prior research is concerned with understanding the elasticity concept, and with measuring cloud platform performance. This work has appeared in formal academic forums, and/or in trade press or practitioner blogs.

2.1 Elasticity : Definition and Characteristics

As in any hype-prone field, it is important to get common usage of marketable terms such as "elastic". Several efforts have tried to explain the meaning of this term, and others that are used about cloud platforms, and along with explanation, they point to aspects of the platform's performance that can be important for elasticity. Unlike our work, these studies do not give explicit measurement proposals.

Armbrust et al.[1] discuss relevant use cases and potential benefits for cloud platforms. Among their points, they draw attention to the value of cloud elasticity as compared to the conventional client-server model in the context of perceived risks due to over- and under- provisioning.

The prestigious National Institute of Standards and Technology (NIST) has attempted to explain and define terms for the cloud[4]. For elasticity, NIST points to rapid provisioning and de-provisioning capability, virtually infinite resource capacity with unconstrained purchasable quantity at any single moment.

Two other discussions from David Chiu[5] and Ricky Ho[6] have pointed to an additional important factor of elastic behaviour, i.e. the granularity of usage accounting. That means that elasticity when load declines is not only a function of the speed to decommission a resource, but also it depends on whether charging for that resource is stopped immediately on decommissioning, or instead is delayed for a while (say till the end of a charging quantum of time).

[4] NIST homepage: http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

[5] Crossroads, Vol. 16, No. 3. (2010), pp. 3-4.

[6] http://horicky.blogspot.com/2009/07/between-elasticity-and-scalability.html

2.2 Cloud Performance Analysis and Evaluation Benchmarks

There has been significant work on developing new ways to conduct performance evaluation of public cloud providers, looking especially at aspects such as price/performance and scalability.

A number of recent research efforts conducted in-depth performance analysis on the virtual machine instances offered by public cloud providers. For example, Stantchev et al.[11] introduce a generic benchmark to evaluate the nonfunctional properties(e.g., response time, transaction rate, availability etc) of individual cloud offerings for web services from cost-benefit perspective. Dejun et al.[4] and Schad et al [10] examine the performance stability and homogeneity aspects of VM instances over time to determine the prospect of dynamic resource provisioning for SLA-aware applications in the cloud. Although these studies are useful to understand the underlying performance characteristics of the cloud infrastructure services for application deployment, they do not consider the responsiveness of the platform during scaling with the variation in workload demand. On the other hand, our elasticity model aims at delving into this problem domain and assigning a single penalty factor to reflect the performance-cost implications of public cloud platforms as they adjust under a variety of time-varying workloads.

Several different performance benchmarks [14, 9] have proposed to quantify the resource spin-up (spin-down) delay in the cloud as one (among many) important performance metrics. For instance, Yigitbasi et al.[14] present a framework to determine the performance overheads associated with the scaling latency of the virtual machine (VM) instances in the cloud. The framework measures the provisioning (deprovisioning) delay of an Amazon EC2 [7] instance and evaluates the corresponding performance overheads (i.e. queue waiting time, response time, bounded slowdown) for jobs across different VM configurations and scheduling algorithms. Likewise, Li et al.[9] developed CloudCmp, a systematic comparator of the performance and cost of cloud providers. They identified a set of metrics (e.g., scaling latency, operation response time, cost per operation etc.) to analyse customer perceived performance and cost effectiveness of public cloud offerings. Their study is comprehensive enough to compare the low level performance metrics and expenses of common cloud offerings and can be used to anticipate the behaviour of an arbitrary application deployed in the cloud. Although these approaches measure several discrete performance metrics, they do not combine their metrics into a macroscopic overview of the platform's adaptability behaviour. In contrast we will propose a single summary measure for elasticity, which will be influenced by several factors that were used in these earlier studies.

Yahoo! Cloud Serving Benchmark (YCSB)[3] is a framework which aims to facilitate the performance comparison across new generation cloud data serving systems (e.g., Cassandra, HBase and PNUTS). This benchmark consists of two tiers; the first one is performance tier which characterises the performance of the databases under load for a variety of workload scenarios while the second one is concerned with scale-up and elastic speed-up measures (that is, they

[7] Amazon EC2 homepage: http://aws.amazon.com/ec2/

consider workloads that grow and grow). Their work is valuable when seeking to analyse the performance implications of large database-intensive applications in the cloud; however, our work differs by considering de-provisioning and resource granularity aspects as well. Furthermore, our elasticity model captures the financial implications as well as traditional performance.

Donald Kossmann's group at ETHZ has a research project on benchmarking cloud platforms. An initial workshop discussion [2] proposed that it would be useful to take the ratio of the throughput achieved by operations with acceptable response time, to the rate of requests, in workloads with successive peaks and troughs. However, in the later conference paper [7] they present an extensive evaluation of the end-to-end scalability aspect of existing cloud database architectures for OLTP workloads. Here they define a set of performance and cost metrics to compare the throughput, performance/cost ratio and cost predictability of existing cloud database systems for larger and larger loads. They look at a much wider variety of measures than we do, but they omit to look at the speed of responding to change in workloads, nor do they consider workloads that can shrink as well as grow.

2.3 Elasticity Measurement Model

Recently, a working paper from Joe Weinman has proposed a numeric measurement of elasticity [13]. This is the closest inspiration for our work, as Weinman considers arbitrary workload patterns, and attempts to construct a numeric score for how a platform handles a workload, reflecting impacts of both over- and under-provisioning.

Weinman starts from a simple conceptual model: consider a resource such as computational capacity, then there is a demand curve that indicates, as a function of time, how much of the resource is needed for the application to work properly. This can be treated as a mathematical function $D(t)$. Similarly, a function $R(t)$ shows how much of the resource is allocated to the application at each time. Perfect elasticity would be shown if $R(t) = D(t)$ for all t. Weinman identifies the situation where $R(t) > D(t)$ as "excess resource" (we say "overprovisioning"), and assigns it a cost which (in the simplest case) is linear in the quantity of resource that is allocated above that needed. Similarly, he considers "unserved demand" (which we call "underprovisioning") when $D(t) < R(t)$, and measures this by opportunity cost which is again linear in the difference, although the constant of proportionality is much higher for unserved demand than for excess resource.

Figure 1 illustrates this for a single resource of CPU capacity; the curves show a hypothetical situation with a sine wave variation in demand (solid blue line), and linearly increasing supply (dotted black line). A value of 150% for demand indicates that the application could use one-and-a-half times the capability of the standard instance, and 150% as the supply indicates that the application has been allocated instruction execution from cycles that were one-and-a-half as frequent as those on a standard instance. Imperfect elasticity is shown by the gap between the curves, and Weinman's proposed measure is a weighted

combination of the areas between the curves (with higher weight for the areas of under-provisioning).

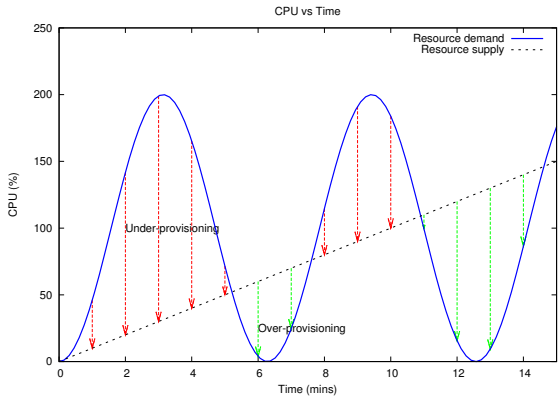

Fig. 1. Elasticity Explained

Our work enhances Weinman's insights in several respects. Most importantly, we use penalties that are more typical in industry SLAs, where, for example, the opportunity cost from high latency is not linear in the delay, but rather depends on whether the latency has breached a threshold, and furthermore we allow a small number of requests to see excessive latency. Also, we distinguish between the resources that are allocated, and those that are charged to the consumer (as we will see, the difference between these can be significant). We considered pragmatic issues needed to produce a figure-of-merit for a platform, by choosing explicit workloads and carrying out measurements, while Weinman's paper is entirely theoretical. Unlike Weinman, we explore the impact of the scaling rules used in the platform, to provision or deprovision instances.

3 Elasticity Measurement

This section defines our proposal, for how to determine a figure that expresses "how elastic is a given cloud platform". We begin by explaining a general framework to measure the cost of imperfect elasticity when running a given workload, with separate penalties for overprovisioning and underprovisioning; the sum of these is the penalty measurement for the workload. By considering a suite of workloads, and combining penalties calculated for each, we can define a figure-of-merit for a cloud platform. Next we discuss choices that we have made that lead to an explicit measurement, taking concrete decisions on the SLA aspects that are evaluated, charging rates, and the particular suite of workloads.

3.1 Penalty model

We present in this section our "penalty model" approach to measuring imperfections in elasticity for a given workload in terms of monetary units. We assume that the system involves a variety of resource types. For example, the capacity of an EC2 instance can be measured by looking at its CPU, memory, network bandwidth, etc. We assume that each resource type can be allocated in units. We assume that the user can monitor the platform, to learn what level of resourcing is allocated, and also the consumer knows the QoS metrics for their requests (such as distribution of latency, how many requests failed, etc). Amazon CloudWatch[8] is an example of the monitoring functionality we expect.

Our elasticity model is composed of two parts: penalty for over-provisioning and penalty for under-provisioning. The former captures the cost of *provisioned* but *unutilised* resources, while the latter measures opportunity cost from the performance degradation that arises with under-provisioning.

Penalty for Over-provisioning In existing cloud platforms, it is usual that resources are temporarily allocated to a consumer from a start time until a finish time. The start time may occur because the consumer directly requests some resource based on its observed or predicted needs, or perhaps the platform may proactively do the allocation following some rules. This is represented by a function we call available supply and denote by $R_i(t)$ for each resource i. In current platforms it can also happen that a resource may be charged to a consumer even without being available. For example, in Amazon EC2, an instance is charged from the time that provisioning is requested (even though there is a delay of several minutes before the instance is actually running for the consumer to utilise). Similarly, charging for an instance is done in one-hour blocks, so even after an instance is deprovisioned, the consumer may continue to be charged for it for a while. Thus we need another function $M_i(t)$ that represents the *chargeable supply* curve; this is what the consumer is actually paying for. These curves can be compared to the demand curve $D_i(t)$.

The basis of our penalty model is that the consumer's detriment in over-provisioning (when $R(t) > D(t)$) is essentially given by the difference between chargeable supply and demand; as well, we charge a penalty even in under-provisioned periods whenever a resource is charged for but not available (and hence not used). These penalties are computed with a constant of proportionality c_i that indicates what the consumer must pay for each resource unit. In real systems, resources of different types are often bundled, and only available in collections. We assume that some weighting is used to partition the actual monetary charge for the bundle between its contained resources. For example, in Amazon EC2, a small instance in US East region costs $0.085 per hour, and this is equivalent to a collection comprising a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor, with 1.7GB memory, 160GB instance storage, etc.

[8] Amazon CloudWatch API homepage: http://aws.amazon.com/cloudwatch/

Formally, we define the overprovision penalty $P_o(t_s, t_e)$ for a period starting at t_s and ending at t_e. We assume a set of resources indexed by i, and we use functions $D_i(t)$, $R_i(t)$, and $M_i(t)$ for the demand, available supply, and charged supply, respectively, of resource i at time t. Our definition aggregates the penalties from each resource, and for each resource we integrate over time.

Definition 1.

$$P_o(t_s, t_e) = \sum_i P_{o,i}(t_s, t_e)$$

$$P_{o,i}(t_s, t_e) = \int_{t_s}^{t_e} c_i \times d_i(t) dt$$

$$d_i(t) = \begin{cases} M_i(t) - D_i(t) & \textit{if } R_i(t) > D_i(t), \\ M_i(t) - R_i(t) & \textit{if } M_i(t) > R_i(t) \textit{ and } D_i(t) \geq R_i(t), \\ 0 & \textit{otherwise.} \end{cases}$$

Our definition meets tests of reasonableness, such as (i) penalty is non-negative, (ii) penalty increases when the chargeable supply increases, against a fixed demand curve.

Penalty for Under-provisioning Next, we turn to the penalty model for under-provisioning. In contrast to over-provisioning, the penalty when resources are insufficient is a measure of the opportunity cost to the consumer, when poor performance is experienced. The penalty is calculated from SLAs that capture how service matters to the consumer.

We assume that the consumer has used their business environment to determine a set of performance or Quality of Service (QoS) objectives, and that each is the foundation for an SLA-style quantification of unsatisfactory behaviour. For example, the platform's failure to meet the objective of availability can be quantified by counting the percentage of requests that are rejected by the system. In many cases, such SLA quantification might reflect a wide variety of causes, not only those that arise from underprovisioning, but also some from eg network outage etc. We assume that the customer also knows how to convert each measurement into an expected financial impact. For example, there might be a dollar value of lost income for each percent of rejected requests. In many cases, the financial impact may be proportional to the measurement, but sometimes there are step functions or other nonlinear effects (for example, word-of-mouth may give a quadratic growth of the damage from inaccurate responses). To provide a proper baseline for the penalties, we also consider the expected optimal value that occurs when resources are unlimited (effectively, we measure with so much overprovision that additional allocation would not change the SLA measurement).

Formally, we let Q be a non-empty set of QoS measures, and for each $q \in Q$, we consider a function $p_q(t)$ that reflects the amount of unsatisfactory behaviour observed on the platform at time t. The consumer provides also, for each QoS

aspect q, a function f_q that takes the observed measurement of unsatisfactory behaviour and maps this to the financial impact on the consumer. Let $p_q^{opt}(t)$ denote the limit (as $K \leftarrow \infty$) of the amount of unsatisfactory behaviour observed in a system that is statically allocated with K resources.

Thus we define the underprovision penalty $P_u(t_s, t_e)$ for a period starting at t_s and ending at t_e

Definition 2.

$$P_u(t_s, t_e) = \sum_{q \in Q} P_{u,q}(t_s, t_e)$$

$$P_{u,q}(t_s, t_e) = \int_{t_s}^{t_e} (f_q(p_q(t)) - f_q(p_q^{opt}(t)))dt$$

Total Penalty Rate for an Execution We calculate the overall penalty score $P_u(t_s, t_e)$ accrued during an execution from t_s till t_e, by taking the sum of the penalties from both over- and under-provisioning; note that both are expressed in units of dollars. We then normalise to calculate the total penalty rate P in dollars per hour. A lower score for P indicates a more elastic response to the given workload.

Definition 3. *The penalty score over a time interval* $[t_s, t_e]$ *is defined as follows:*

$$P(t_s, t_e) = P_o(t_s, t_e) + P_u(t_s, t_e)$$

$$P = \frac{P(t_s, t_e)}{t_e - t_s}$$

3.2 Single Figure of Merit for Elasticity

The definitions above measure the elasticity of the systems response to a single demand workload. Different features of the workload may make elastic response easier or harder to achieve; for example, if the workload grows steadily and slowly, a system may adjust the allocation to match the demand, but a workload with unexpected bursts of activity may lead to more extensive underprovisioning. Thus, to get a measure for the elasticity of the platform, we consider a suite of different workloads, and determine the penalty rate for each of these.

To combine measured penalty rates from several workloads into a single summary number, we follow the approach used by the SPEC family of benchmarks. That is, we choose a reference platform, and measure each workload on that platform as well as on the platform of interest. We take the ratio of the penalty rate on the platform we are measuring, to the rate of the same workload on the reference platform, and then we combine the ratios for the different workloads by the geometric mean. That is, if $P_{x,w}$ is the penalty rate for workload w on platform x, and we have n workloads in our suite, then we measure the elasticity of platform x relative to reference platform x_0 by

$$E = [(P_{x,w_1}/P_{x_0,w_1})(P_{x,w_2}/P_{x_0,w_2}) \dots (P_{x,w_n}/P_{x_0,w_n})]^{1/n}$$

3.3 Concrete Choices for an Elasticity Benchmark

The approach to elasticity described above is flexible, and could be adapted to the needs of each consumer, through choosing appropriate SLA objectives and metrics that reflect the business situation, workloads that are representative of that consumer's patterns of load variation, etc. To actually determine an elasticity score, we need to make one set of choices for all these parameters. For the purpose of this paper, we use the following. Our workload consists of requests that follow the TPC-W application design [12, 5].

To calculate overprovisioning penalty, we deal with a single resource (CPU capacity, relative to a standard small EC2 instance) and measure the financial charge as $0.085 per hour per instance. This reflects the current charging policy of AWS. To calculate underprovisioning penalty, we were inspired by the specific QoS constraints that are enforced in TPC-W. In particular, each user in our workload pattern lands on the homepage first and then searches for newly released books. TPC-W expects at least 90% of these requests generated by the users will see a response within 3 seconds. So we have used the following two QoS aspects with associated penalties. The cost penalty for latency violation is a simplified version of the cost function mentioned in [8]. As e-commerce websites lose more revenue for slower latency issue than that of application down-time[9], we associate a lower cost penalty for unavailability impact.

- (Latency) No penalty as long as 90% of requests have response time up to 3 seconds; otherwise, a cost penalty, 12.5¢ will apply for each 1% of additional requests (beyond the allowed 10%) that exceed the 3 seconds latency .
- (Availability) Cost penalty of 10¢ will apply for each 1% of requests that fail completely (they are dropped or time out).

While these penalty charges are chosen somewhat arbitrarily, we note that the penalty for unmet demand is very high compared to the cost of provisioning; this is accurate for real consumers. As Weinman [13] points out, the cost of resources should be much less than the expected gain from using them (and the latter is what determines the opportunity cost of unmet demand).

In our measurements, we use a set of 10 different workloads, which grow and shrink in a variety of shapes.

- Sinusoidal Workloads: This workload type demonstrates the periodicity often found in web traffic. For example, news sites may see an hourly cycle, while banking may vary daily. These loads can be expressed as $D(t) = A(sin(2\pi t/T + \phi) + 1) + B$, where A is the amplitude, B is the base level, T is the period and ϕ is the phase shift. For the benchmark suite, we use three different examples, whose periods are 30 minutes, 60 minutes and 90 minutes, respectively. All have peak demand of 450 requests per second, and trough at 50 requests/sec. A load of 100 req/s is about what one VM instance can support.

[9] http://blog.alertsite.com/2011/02/online-performance-is-business-performance/

- Sinusoidal Workload with Plateau: This workload type modifies the sinusoidal waveform, by introducing a level (unchanging) demand for a certain time, at each peak and trough. Thus the graph has the upswings and downswings, with flat plateau sections spacing them out. In the suite we have three workloads like this, each starting from the sinewave with period of 30 minutes; in one case the plateau at each peak lasts 10 minutes, in another it lasts 40 minutes, and in the last of this type, the peak plateaus last 70 minutes each. In all cases, the plateaus at troughs last 10 mins (and there is always a plateau at the start of the experiment and at the end).
- Exponentially Bursting Workload: This workload type exhibits extremely rapid buildup in demand (rising U-fold each hour), followed by a decay (declining D-fold each hour). This is inspired by experience of fads (say when a site is slashdotted). The demand of a website whose popularity grows fast with time (e.g., social networking and gaming applications). We provide two workloads of this type, one with $U = 18$ and $D = 2.25$; the other has $U = 24$ and $D = 3$ (so this rises and falls more quickly).
- Linearly Growing Workload: This workload represents a website whose popularity rises consistently. It can be stated as $D(t) = mt + c$, where m is the slope of the straight line and c is the y-axis intercept. We have one example of this type, with workload that starts at 50 req/s (and stays here for 10 mins to warm the system up), then the load rises steadily for 3 hours, each hour increasing the rate by an extra 120 req/s. Thus we finish with 410 req/s.
- Random Workload: This workload type is a random one where the generation of requests is continuous in time and independent from each other. We have one example of this type, with requests produced by Poisson sampling on a set of random values.

We note that the demand curves described above are expressed in terms of the rate requests are generated; in practice, performance variation in identical instances means that this does not lead the utilisation of CPU resources to track the desired demand pattern exactly.

4 Experimental Setup

In the high level view, the architecture of our experimental setup can be seen as a client-server model. The client side is a workload generator implemented using JMeter [10], which is a Java workload generator used for load testing and measuring performance. The sole purpose of JMeter in this experiment is to generate workload based on our predefined workload patterns. These workload patterns are of various types and in the form of XML scripts. To allow generation of extremely large workload, we operate JMeter in master-slave mode, whereby a single master, that contains a workload instance, distributes the workload over a number of slave instances, the slaves then apply the workload to the server side.

[10] JMeter homepage: http://jakarta.apache.org/jmeter/

We chose TPC-W [12, 5] as the application in all our suite of workloads, because it has easy-to-obtain code examples and it is most often used in the literature. It can be substituted with other applications if desired. TPC-W emulates user interactions of a complex e-commerce application (such as an online retail store). In our experiment, we adopt the online bookshop implementation of TPC-W application server and deploy it into EC2 small instances of the web-server farm in the AWS cloud. Instead of having the TPC-W workload generator at the client side, we use JMeter to specify our pre-defined workload patterns.

The server side is setup as an application with a single load-balancer facing the client side, and a number of instances behind the load-balancer. We hosted all of the web servers on EC2 m1.small instances at US-East Virginia region (the cost of each instance is 8.5¢ per hour, matching the penalty we apply for over-provisioning). The number of instances is not fixed, but rather it is controlled by an autoscaling engine which dynamically increases and decreases the number of instance based on the amount of workload. The behaviour of an autoscaling engine follows a set of rules that must be defined. Each ruleset produces a different "platform" for experimental evaluation, with different elasticity behaviour.

An autoscaling rule has the form of pair consisting of an antecedent and a consequence. The antecedent is the condition to trigger the rule (e.g., CPU utilisation is greater than 80%) and a consequence is the action to trigger when the antecedent is satisfied (e.g., create one extra instance). In our experiments we consider two platforms, because we run with two different rulesets. The detailed configurations of the autoscaling engine (configured via AutoScaling[11] library) is shown in Table 1.

Table 1. AutoScaling Engine Configuration

Rule Set	Monitoring Duration	Upper Breach Duration	Lower Breach Duration	Upper Threshold	Lower Threshold	VM Increment	VM Decrement
1	1 minute	2 minutes	2 minutes	70%	30%	1	1
2	1 minute	1 minute	20 minutes	70%	30%	2	1

We measure available supply $R(t)$ by using the reports from CloudWatch showing the number of instances that are allocated to our experiment; we treat k instances as $R(t) = 100 \times k\%$ of supply, so this function moves in discrete jumps. Chargeable supply $M(t)$ is determined from the launch time and termination time of the allocated EC2 instances, given by AWS EC2 API tools For demand, our generator is defined to produce a given number of requests, rather than in the measure of CPU capacity, that is needed for our measurements. Thus we use an approximation; we graph $D(t)$ from what CloudWatch reports as the sum of the utilisation rates for all the allocated instances. As will be seen in the graphs in Section 5, this is quite distorted from the intended shape of the demand function. One distortion is that measured $D(t)$ is capped at the available supply,

[11] AutoScaling API homepage: http://aws.amazon.com/autoscaling/

so underprovisioning does not show up as $D(t) > R(t)$. This inaccuracy is not serious for our measurement of elasticity, since the use of $D(t)$ in measurement is only for cases of overprovisioning; during underprovisioning, the penalty is based on QoS measures of latency and lost requests, and these do reveal the growth of true demand. Another inaccuracy is from the system architecture, where requests that arrive in a peak period may be delayed long enough that they lead to work being done at a later period (and thus measured $D(t)$ may be shifted rightwards from the true peak). As well, there is considerable variation in the performance of the supplied instances [4], so a given rate of request generation with 450 req/s can vary from 350% to 450% when we see the measured demand. Future work will find ways to more accurately measure demand in CPU capacity units.

5 Case Studies

We describe in some detail the observations made when we run our workloads against Amazon EC2. These case studies serve two purposes: (1) as a means of sanity checking the elasticity model in Section 3. That is, we can see that the numerical scores, based on our elasticity model, do in fact align with what is observed in over- or underprovisioning. For example, in reducing the steepness of a workload increase we shall observe that supply tracks more closely to demand, and the penalty calculated is lower. (2) We demonstrate the usefulness of our elasticity benchmark in exposing situations where elasticity fails to occur as expected, and other interesting phenomena can be observed.

5.1 Workload Pattern Case Study

To begin, we applied each of the 10 workload patterns from our elasticity benchmark, in EC2 with a fixed scaling rule set 1 as defined in Table 1. This rule set is common in tutorial examples, and it seems widespread in practice. With this rule set, the number of instances increases by one when average CPU utilisation exceeds 70%, and one instance is deprovisioned when CPU utilisation drops below 30%.

Effect of Over- and Under-provisioning Figure 2 shows behaviour of the platform in response to an input sinusoidal workload with a period of 30 minutes. The CPU graph shows the available supply, chargeable supply and demand curves over a 110-minute interval. Initially, there was only one instance available to serve the incoming requests. As workload demand increases (after 15 minutes), the rule triggers provisioning a new instance, but there is a delay of about 10 minutes until that is available (however it is charged as soon as the launch begins). As workload generation is rising fast during this delay, the system experiences severe effects of underprovisioning: latency spikes and penalties accrue at about 30¢/min. In our implementation, demand is measured on the instances and so the curve shown is capped at the available supply, rather than showing the full upswing of the sinewave. The lag between charging for the instance and

it being available, is reflected in a penalty for overprovisioning of about 0.2¢/min during this period.

Deprovisioning of Resources Our work's inclusion of cases where workload declines is different from most previous proposals for cloud benchmarks. These situations show interesting phenomena. We saw situations where a lag in releasing resources was actually helpful for the consumer. In Figure 2 we see, on the downswing of the demand curve, that most of the resources claimed on the upswing were kept; this meant that the next upswing could utilise these instances, and so the latency problems (and underprovisioning penalty) were much less severe than in the first cycle.

We also observe in downswings that the difference between chargeable supply and available supply is significant, with a deprovisioned instance continuing to attract charge till the end of the hour-long quantum. We see in the upper graph of Figure 5 (with rule set 1) that the chargeable supply is simply not following the demand curve at all, and indeed there are extensive periods when the consumer is paying for 6 instances, even though they never have more than 5 available for use.

Considering the evolution of the supply led us to discover an unexpected inelasticity phenomenon, where the cloud-hosted application is never able to cut back to its initial state after a temporary workload burst. The average utilisation may not drop below 30% which triggers deprovisioning, even though several instances are not needed. To demonstrate this fact, we ran a Sinusoidal workload pattern with peak at 670 req/s and trough at 270 req/s, and 40 minutes plateau at each peak and trough; the resultant graphs are shown in Figure 4. The peak workload triggered the creation of 6 instances. The long-lasting trough workload (about 136% CPU) could easily be served by 2 instances, however, the number of VM instances remained at 4.

Trends In Elasticity Scores Looking at the penalty scores in Table 2, we can see how the calculated penalty varies with the type of workload. In all workloads, the overall penalty is dominated by the loss in revenue due to under-provisioning. This is appropriate to business customers as the opportunity cost, from unmet requests or unsatisfactory response that may annoy users, is much more than the cost of resources.

For pure sinusoidal workload patterns, the overall penalty declines with the increase in waveperiod. This demonstrates EC2 platform is better at adapting to changes that are less steep, where underprovisioning penalties will be less severe as the demand will not have increased too much in the delay from triggering a new instance, until it is available to serve the load.

Turning to the sinusoidal workload with plateaus, we observe that a 10 minutes plateau at peak and trough worsens the overall penalty as compared to the basic sinusoidal workload where the peak is sharp (sine_30). We attribute this rise in penalty to the insertion of a plateau at the trough which wipes out the resource-reuse phenomena in subsequent cycles. With a trough plateau, the

system has time to deprovision and return to its initial state before the next cycle; therefore, each cycle could not take advantage of the resources created in the previous cycle and it pays a similar underprovisioning penalty. See the upper graph of Figure 5. As the length of the plateau at the peak increases (from 10 minutes to 70 minutes), overall penalty gradually moves down. The system has time to adapt to the peak demand, and serve it effectively for longer.

For the exponential burst workloads, we can observe large penalty values in Table 2. In general, under-provisioning penalty tends to rise as the growth rate increases; that means, the underlying cloud platform is not elastic enough to grow rapidly with this type of workload, thus resulting in sluggish performance for these fast-growing workload types as they head towards the peak. Figure 3 explains the performance implications of an exponential workload with growth and decay rates of 24 (increase 24 fold per hour) and 3 (decrease 3 folds per hour) respectively. The CPU utilisation and request count graphs are mostly distorted and the application could not serve the peak workload because of the lag between peak demand and resource supply introduced by the platform. Although EC2 instance increment rule was triggered with the ramping-up of the workload demand, those instances were available long after the peak demand; thus resulted in huge under-provisioning penalty due to high latency, dropped requests and time-outs. The high under-provisioning cost in the penalty graph also confirms EC2 platform's inelasticity in coping up with this fast-paced workload pattern. The under-provisioning penalties incurred were much higher than in the sinusoidal workloads, indicating that EC2 platform is not so adaptive to traffic surges with high acceleration rate. Again, looking at the over-provisioning penalty graph, we observe large over-provisioning cost (around 0.8¢/min) right after the peak is over; as some of the VM instances launched during the peak load were available at the off-peak period, they just accrued more penalty due to over-provisioning with no significant positive impact in improving under-provisioning penalty.

Likewise, EC2 platform turns out to be quite inelastic for random and highly fluctuating workload patterns. Since the bursts are too short-lived and independent from one another, it is likely that the platform's lag in provisioning resources will leave the hosted application out of sync with the peak burst. This will incur over-provisioning cost with no positive impact on improving the under-provisioning penalty. For this reason, we observe high penalty score for random workload pattern in Table 2 indicating platform's lack of agility in resource alignment.

Unlike the above workloads, linear workload yields less overall penalty. This suggests that EC2 platform can easily cope up with workloads with lower and consistent growth. This is not surprising as slowly growing workload pattern is not affected by the provisioning delay of the underlying platform and therefore incur less under-provisioning penalty. However, we expect that as the slope becomes steeper, the overall penalty will show a rising trend, as the resources are not provisioned rapidly enough.

Table 2. Penalty for Benchmarking Workloads - Rule Set 1

	$P_o(t_s, t_e)/hr$	$P_u(t_s, t_e)/hr$	$P(t_s, t_e)/hr$
sine_30	30.81¢	270.76¢	301.57¢
sine_60	28.96¢	259.80¢	288.76¢
sine_90	24.74¢	86.5¢	111.24¢
sine_plateau_10	33.71¢	469.82¢	503.53¢
sine_plateau_40	26.31¢	203.6¢	229.91¢
sine_plateau_70	30.86¢	120.29¢	151.15¢
exp_18_2.25	22.82¢	288.33¢	311.15¢
exp_24_3.0	21.95¢	652.03¢	673.98¢
linear_120	14.63¢	52.60¢	67.23¢
random	29.96¢	502.53¢	532.49¢

Table 3. Penalty for Benchmarking Workloads - Rule 2

	$P_o(t_s, t_e)/hr$	$P_u(t_s, t_e)/hr$	$P(t_s, t_e)/hr$	Ratio to Rule 1
sine_30	35.98¢	122.03¢	158.01¢	0.52
sine_60	40.45¢	55.71¢	96.16¢	0.33
sine_90	34.58¢	49.52¢	84.10¢	0.76
sine_plateau_10	28.95¢	107.73¢	136.68¢	0.27
sine_plateau_40	38.89¢	66.79¢	105.68¢	0.46
sine_plateau_70	39.24¢	66.21¢	105.45¢	0.70
exp_18_2.25	28.39¢	248.90¢	277.29¢	0.89
exp_24_3.0	30.94¢	581.33¢	612.27¢	0.91
linear_120	37.20¢	54.70¢	91.9¢	1.37
random	35.29¢	488.5¢	523.79¢	0.98
Geometric Mean	N/A	N/A	N/A	0.65

5.2 Case Study: Impact of Scaling Rules on Elasticity

In the previous experiments with the widely-used rule set 1, we observe severe under-provisioning penalty dominating the overall score. We saw cases where the system took too long in adjusting to rapid growth in demand. When demand drops, there is a tradeoff: slow response increases the duration of overprovisioning charges, but it can help if a subsequent upswing occurs that might reuse the retained resources. To improve the elasticity, one recommendation would be changing the scaling rules so that they are aggressive in provisioning extra resources and conservative enough in de-provisioning those resources. The initial (Rule Set 1) and adjusted (Rule Set 2) scaling rule sets are shown in Table 1.

Figure 5 shows the CPU utilisation of the two rule sets under the same sine plateau workload pattern.

Rule set 2 performed markedly better than rule set 1. The main reason is that Sinusoidal workload pattern with a period of 30 minutes has a very sharp increase rate. This makes rule set 1 less suitable because it only adds one instance for each rule trigger and there is a cooling period which stops it from immediately creating another instance (even if the condition is met). On the other hand, rule set 2 increases two instances at each trigger thus enables it to respond quicker to sharp workload increase.

On the downswing, we see that rule set 1 responds much quicker to the drop of demand by deprovisioning its resources. Though the trend of available supply follows closely with the demand, the chargeable supply does not follow as well. As a result, resources are being charged but not used. In contrast, as we intended, rule set 2 (with a lower breach duration of 20 minutes rather than 2 minutes) keeps the resources from the previous upswing so that they are reused in the subsequent cycles of the workload demand.

This benefit from resource reuse holds as long as workloads come in periodic bursts and the inter-arrival time of bursts are shorter than the lower breach duration in the rule set. If the inter-arrival time is more than the lower breach duration, subsequent bursts will not be able to enjoy the resource reuse phenomenon as the resources are likely to be released by that time. For this reason, the penalty for exponential workloads could not improve that much with rule set 2 as the duration between the bursts is long enough to set the number of instances back to the initial state (1 EC2 instance); therefore, successive bursts could not make use of the resources from the previous one.

Results for all workloads of our suite are shown in Table 3 which should be compared to Table 2. One clear disadvantage of rule set 2 is that it is likely to overprovision too much in the case where workload does not increase quickly. This is reflected in the experiment with the linear workload pattern. The modest pace of growth in demand here means that scaling rule set 1 was sufficient to align the resource supply with its resource demand. Scaling rule set 2 rather worsened the over-all penalty in this case by doubling the over-provisioning cost.

We computed a single figure of merit based on the SPEC family of benchmarks as defined in Section 3.2. We used the platform with rule set 1 as the reference when evaluating rule set 2. The last column in Table 3 shows the ratio

of the total penalties between the two rules sets for each of the 10 workload patterns. All but one of these ratios are smaller than one, indicating that rule set 2 is generally more elastic than rule set 1 for the benchmark workload patterns. We calculated the geometric mean of these ratios as 0.65, which quantifies the improvement in elasticity. Thus we demonstrated that our single figure for elasticity can be effectively used to compare different rule configurations. It can also used to detect variation in elasticity level between platforms from different cloud providers, as well as variation over time within a cloud provider due to the consistently evolving underlying infrastructures of cloud.

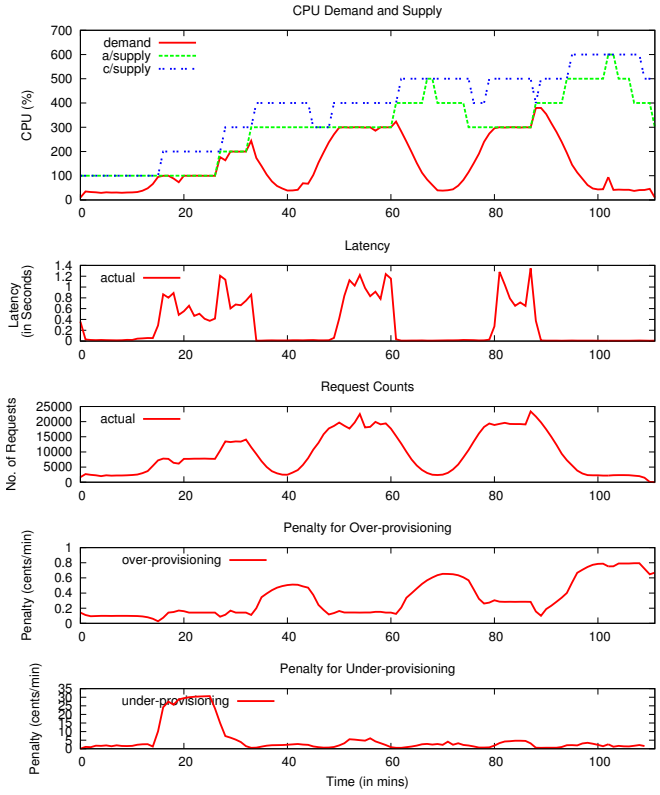

Fig. 2. Results of Sinusoidal Workload with Period 30 Minutes

6 Discussion

The case studies we performed have given us insights into how a cloud platform can be better or worse at elasticity when following a varying workload.

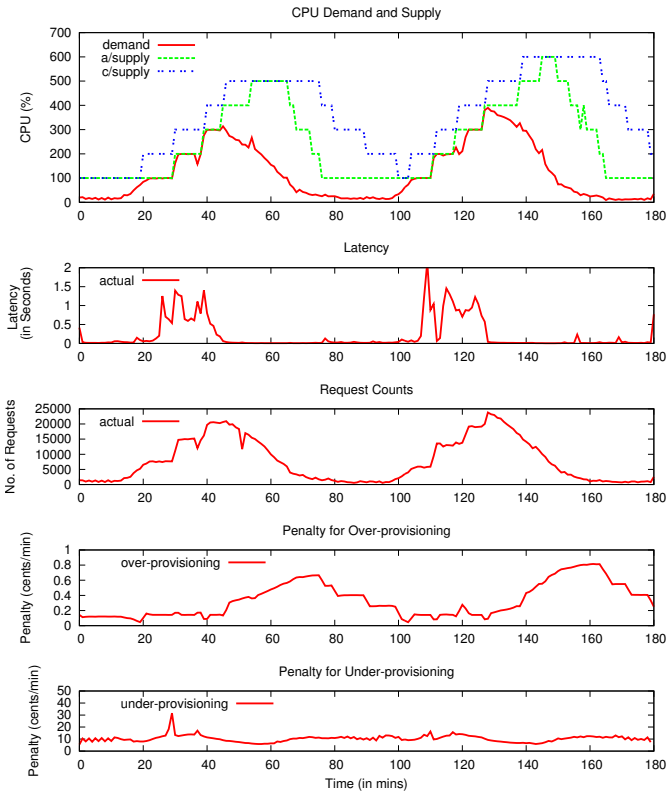

Fig. 3. Results for Exponential Workload with Growth Rate 24/hour and Decay 3/hour

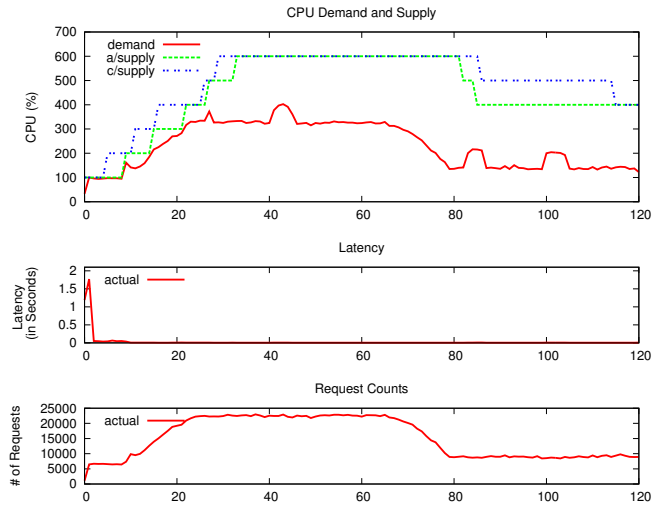

Fig. 4. Results for the Trapping Scenario

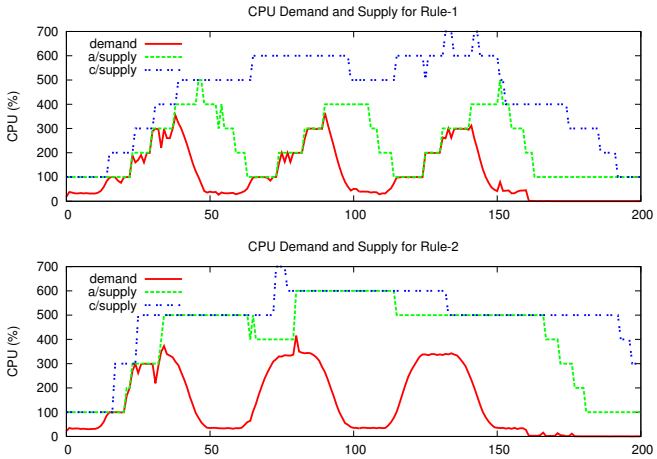

Fig. 5. Comparison of Results between Rule-1 and Rule-2 for Sine Plateau Workload with 10 Minutes Peak Plateau

Identifying the importance of these characteristics of a platform should be of independent value to consumers who want to choose a platform, and they may also help a cloud provider to offer better elasticity in their platform.

It is already well understood that the granularity of instances is important in elasticity. If a consumer gets another substantial PC-like (virtual) machine as the smallest unit of increased resource, this is less elastic than a platform which can share its physical boxes in smaller fragments (e.g., allowing each customer to have whatever percentage of the cycles that they need, as in the case of GAE[12]). Similarly, the time delay between a request for provisioning, until one can actually run on the new instance, can be significant. We observed that this delay varies unpredictably, but can be over 10 minutes. If the workload is increasing fast enough, by the time 10 minutes have elapsed, the previous configuration may have become badly overloaded, leading to poor QoS and thus to financial losses for the consumer. Finally, the delay to decommission an instance is also important. Being too slow to give up resources is wasteful, but being too eager can leave the consumer without resources if/when workload recovers to previous levels.

We have seen how important it is to understand the way the consumer is charged for resources, We have seen that this can be quite different from the actual access to those resources, and the difference is important for the consumer's perception of elastic behaviour. When charging runs till the end of a substantial quantum (e.g., an hour, for EC2), we can see financial losses from too rapid response to changed load. In particular, if the fluctuating load leads a consumer to give up an instance, and then they need to request it back, they may end up paying for it twice over.

Our experiments have shown the way changes to the provisioning and deprovisioning rules can alter the elasticity of the platform. This seems to deserve much more attention from consumers, and we haven't found useful guidelines in research or tutorial literature. In particular, many applications seem to follow sample code, and use a default policy where instances are created or given up based on utilisation levels holding for a fairly short time (e.g., 2 minutes). By being less eager to deprovision, we saw a different ruleset gave about 50% improvement in the elasticity measure.

It is a pleasing sign that running our benchmark has been informative for us. We now reflect directly on the advantages and disadvantages of the decisions we made in proposing this benchmark, that is, how exactly we decided to measure elasticity.

Having a variety of workloads, with diverse patterns of growth and decline in demand, is clearly essential for a measurement of elasticity. Without this, we are just measuring scalability, as in the previous research on cloud performance measures. We found that having workloads that rise rapidly (that is, fast compared to the provisioning delay in the platform) reveals many cases of poor elasticity. It is also vital to let demand decline and then rise again so that we see failure of elasticity from an overlong charging quanta.

[12] Google Apple Engine: http://code.google.com/appengine

Given multiple workloads, we needed some way to combine the observations into a summary number. The approach we used, taking the ratio of each workload against a reference platform, and then forming geometric mean of the ratios, is that used in the SPEC family of benchmarks. It gives consistent relative scores no matter which platform is the reference. It is very robust in that it does not change depending on subtle choice of weights, nor on the scale chosen for each workload (unlike e.g. taking some average of the actual observations).

Our calculated penalty for overprovisioning is based on the charged level of resources, rather than on the resources that are actually allocated (as in Weinman's discussion of elasticity [13]). We have seen that there can be a considerable difference between these quantities. Using charged level has been helpful in making our measurements match the consumer's concerns. By this decision, we properly give a worse score for a system if it keeps charges going for a longer quantum. Our penalty calculation for underprovision is based on observed QoS, and using consumer-supplied functions to convert each observation into the opportunity cost. We do not assume a constant impact of each unmet request. This clearly fits with widespread practice, where SLAs with penalty clauses are enshrined in contracts.

In Weinman's discussion, each time period is penalised for either underprovisioning or overprovisioning, but not for both. We allow both penalties, for example, when utilisation is around 95%, one has 5% of the capacity unused, but one may also see queuing delays that lead to poor QoS. For us, optimal elasticity is not when allocation equals demand, but rather when allocation is the least that still allows as many QoS conditions to be met as possible.

Overall, we feel that our approach fits well with the decision-making of a consumer who is trying to choose the most suitable cloud platform for their needs.

7 Conclusion

Small and medium enterprises are heading towards the cloud for many reasons, including to increase their profit by exploiting elasticity when they face time-varying workload that may be unpredictable. To choose appropriately between platforms, a consumer of cloud services needs to have a way to measure the features that are important, one of which is the amount of elasticity of each platform. This paper offers the first concrete proposal giving a numeric score for elasticity. We adopt some ideas from Weinman's discussion [13], such as representing the cost of wasted resources, and the opportunity cost of unmet demand. We suggested specific new ways to use SLAs to determine penalties for underprovisioning. We have defined a suite of workloads that show a range of patterns over time. With this, we carried out a case study on two platforms. This has shown that our approach is feasible, and that it leads to helpful insights into the elasticity properties of the platform. In particular, we have shown that one gets poor elasticity when following a widespread ruleset for provisioning and deprovisioning.

In future, we will extend our measurements to other platforms with a wider range of features. We hope to consider workloads that grow much further than our current set (which peak at demand for about half-a-dozen instances). We also will try examples with a greater range of SLAs and opportunity cost functions. We would like to make the benchmark running as automatic as possible, so that consumers could download a package and simply run it to get a score for each platform they are considering. We see this paper as an important step towards allowing consumers to make informed choices between cloud platforms.

8 Acknowledgement

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.
2. C. Binnig, D. Kossmann, T. Kraska, and S. Loesing. How is the weather tomorrow?: towards a benchmark for the cloud. In *Proceedings of the Second International Workshop on Testing Database Systems*, DBTest '09, New York, NY, USA, 2009. ACM.
3. B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.
4. J. Dejun, G. Pierre, and C. H. Chi. EC2 performance analysis for resource provisioning of service-oriented applications. In *Proceedings of the 2009 international conference on Service-oriented computing*, ICSOC/ServiceWave'09, pages 197–207, Berlin, Heidelberg, 2009. Springer-Verlag.
5. R. C. Dodge JR, D. A. Menasce, and D. Barbara. Testing E-Commerce Site Scalability With Tpc-W. In *Proc. of 2001 Computer Measurement Group Conference*, 2001.
6. D. Florescu and D. Kossmann. Rethinking cost and performance of database systems. *SIGMOD Record*, 38(1):43–48, 2009.
7. D. Kossmann, T. Kraska, and S. Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 579–590, New York, NY, USA, 2010. ACM.
8. S. Krompass, D. Gmach, A. Scholz, S. Seltzsam, and A. Kemper. Quality of service enabled database applications. In *ICSOC*, pages 215–226, 2006.
9. A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: comparing public cloud providers. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 1–14, New York, NY, USA, 2010. ACM.
10. J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *PVLDB*, 3(1):460–471, 2010.

11. V. Stantchev. Performance Evaluation of Cloud Computing Offerings. In *The Thrid International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 187–192, Oct. 2009.
12. Transaction Processing Performance Council (TPC), San Francisco, CA 94129-0920, USA. *TPC-W Benchmark*, Dec. 2003.
13. J. Weinman. Time is Money: The Value of "On-Demand", Jan. 2011. Working paper (Jan 2011), from `http://www.JoeWeinman.com/Resources/Joe_Weinman_Time_Is_Money.pdf`.
14. N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann. C-Meter: A Framework for Performance Analysis of Computing Clouds. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 472–477, 2009.

ISBN 978-1-74210-249-8

School of Information Technologies
Faculty of Engineering & Information Technologies
Level 2, SIT Building, J12
The University of Sydney
NSW 2006 Australia

T +61 2 9351 3423
F +61 2 9351 3838
E sit.information@sydney.edu.au
sydney.edu.au/it

ABN 15 211 513 464
CRICOS 00026A