Performance and Cost Assessment of Cloud Services

Paul Brebner[1] and Anna Liu[2]

[1] (NICTA/ANU)
[2] (NICTA/UNSW)
{Paul.Brebner,Anna.Liu}@nicta.com.au

Abstract. Architecting applications for the Cloud is challenging due to significant differences between traditional hosting and Cloud infrastructure setup, unknown and unproven Cloud performance and scalability characteristics, as well as variable quota limitations. Building workable cloud applications therefore requires in-depth insight into the architectural and performance characteristics of each cloud offering, and the ability to reason about tradeoffs and alternatives of application designs and deployments. NICTA has developed a Service Oriented Performance Modeling technology for modeling the performance and scalability of Service Oriented applications architected for a variety of platforms. Using a suite of cloud testing applications we conducted in-depth empirical evaluations of a variety of real cloud infrastructures, including Google App Engine, Amazon EC2, and Microsoft Azure. The insights from these experimental evaluations, and other public/published data, were combined with the modeling technology to predict the resource requirements in terms of cost, application performance, and limitations of a realistic application for different deployment scenarios.

Keywords: Cloud performance, scalability, cost, limits, quotas, service-oriented performance modeling (SOPM), Amazon EC2, Google AppEngine, Microsoft Azure.

1 Introduction

Since 2007 NICTA has developed Service-Oriented Performance Modeling (SOPM), a technology for modeling the performance and scalability of Service Oriented Architecture (SOA) applications. SOPM has been trialed and proven in both laboratory settings and with government and commercial collaborators [1-4]. Recently we have examined a sub-class of SOA applications deployed on Virtualized or Cloud platforms. Some initial results focusing on power costs and carbon emissions were published in [5]. In this earlier work we assumed that the performance of in-house and Cloud deployments of an application were identical. In this paper we revisit this assumption based on a the results of running a suite of empirical tests on Amazon, Google and Microsoft cloud platforms, combined with other published and public insights into cloud performance and scalability.

E.M. Maximilien et al. (Eds): ICSOC 2010 Workshops, LNCS 6568, pp. 39–50, 2011.

2 Service-Oriented Performance Modelling Clouds

SOPM is a method with tool support for modeling SOA application performance and scalability. It supports the direct modeling of software, usage, and resourcing in terms of services (externally and internally visible, 3rd party; compositions and simple services), workloads (external systems which consume the services), and the deployment of services on physical resources (servers, networks). Models are parameterized with workload, performance, and server data, and a model-driven approach automatically generates a run-time version which is solved by a discrete event simulator to compute workload, service, and server metrics to assist with answering performance questions and identify areas of performance risk. We have found that modeling SOA performance is valuable and complements testing as modeling can often be done cheaper, before a complete system is built, when testing is impractical, and can be used to rapidly investigate architectural alternatives.

The results of this paper are based on modeling a realistic SOA application based on a real system that was previously modeled and validated (a whole-of-government common e-Business service used across many different agencies). We modeled two different workloads over a typical 24 hour period. For a number of different hosting scenarios our goal was to model and compare resource requirements for different in-house and cloud platforms (Section 3), performance differences between platforms, and variability in performance within a platform (Section 4). Resource requirements were used to estimate power and billing costs (Section 3), and check if any limits/quotas are exceeded (Section 5).

The alternative hosting scenarios for comparison are: In-house hosting on bare hardware, in-house hosting with Virtualization, Amazon EC2 (a variety of instance types), Google AppEngine, and Microsoft Azure. There are many similarities (architectural, technological, billing, etc) but also significant differences between platforms. We have developed performance models for each cloud platform using data from a variety of sources. Insights and the results from a suite of test applications we ran ourselves were combined and confirmed with data from the public domain including other papers, public blogs and websites, cloud platform support blogs, benchmarks, and real-time cloud monitoring sites.

3 Resources and Cost

Scenario 1: In-house hosting, bare hardware. The first scenario is the default hosting environment from which the performance measurements were obtained to parameterize and validate the initial model. The services from each of the four deployment zones are deployed to four dedicated physical servers (clustered if necessary). The servers are identical multi-core Intel 2.33GHz CPU rack servers with sufficient memory and LAN bandwidth to ensure that there are no bottlenecks, and "perfect" hardware-based load balancing across clustered servers in the same zone. A fundamental assumption is that the application has been architected and the infrastructure configured to be linearly scalable by increasing CPUs and servers (scale-up and scale-out). We also assume that the performance measurements used to parameterize the model have been obtained from an unloaded system which does not have variable speed CPUs (or the results have been scaled for the difference in CPU speed between

light and heavy loads). Running the simulation model with the expected load predicted the maximum number of CPUs required (without any resource saturation or degradation in service response times) for each zone as follows: Web Server 10 CPUs, Client Server 4 CPUs, Security Server 3 CPUs, and Application Server 26 CPUs, a total of 43 CPUs.

Scenario 2: In-house hosting, virtualization. The second scenario is in-house hosting but on virtualized hardware. The performance model is identical to scenario 1, except that each application zone is deployed to a virtual machine, with the virtual machines sharing a pool of CPUs. We assume that the servers are identical to scenario 1, but that the use of virtualization incurs a performance overhead. Obviously the value of the overhead depends on a combination of factors including CPU type, virtual machine type, application, and load. For this experiment we assumed a CPU overhead of 70% for both workloads on Intel CPUs due to IO virtualization [6]. The model predicted a maximum of 70 CPUs in the pool which is slightly less than 1.7 times the maximum CPUs from scenario 1, as CPU pooling is more efficient overall and ensures maximum utilization. The introduction of virtualization in theory enables easier sharing of the CPU pool with other applications during off-peak periods, although care needs to be taken that there are sufficient CPUs for this application during peak times.

Scenario 3: Cloud Hosting, Amazon EC2. This scenario explores hosting the complete application on the Amazon Elastic Compute Cloud (EC2) infrastructure [9]. For simplification we make the following assumptions about the EC2 hosting: All services are deployed to each instance (in practice different services may need to be deployed to different instances. We look at the performance implications of distributed deployment, but not the resource implication, in Section 4); all instances are of the same type (this is not a requirement of the infrastructure, but simplifies modeling and may be a reasonable simplification for managing clouds in practice); Elastic Load Balancing (ELB) is used, but not modeled (as we cannot determine how many instances are required as load balancing instances scale automatically [7]); auto-scaling of EC2 instances is enabled and there is no delay in spinning up new instances [8] (in practice it takes time to start new instances).

We assume that Amazon Auto Scaling is used to start and terminate instances sufficiently in advance to be available when the load requires them (but only just in time, and that they are terminated immediately after the load drops). Elastic Load Balancing and Autoscaling are relatively complex, and need to be setup (and tuned) for optimal scalability. A significant assumption is that Elastic Load Balancing instances dynamically scale fast enough to cope with load spikes (otherwise clients may experience time outs). Other considerations include ensuring that the number of instances in each availability zone is equivalent, clients are from a variety of IP addresses rather than just one (clients from the same IP address tend to connect to the same instance), and the use of persistent HTTP connections to improve performance. In this section we focus on modeling resource usage, rather than general performance/scalability issues (Section 4).

Amazon EC2 instances come in different types [9]. Each type offers a total number of standardized EC2 Compute Units per instance, consisting of a number of cores per instance, number of compute units per core, and at different prices. A core is a virtual core, but virtual cores have exclusive use of a physical core (except standard small instances where two virtual cores share a physical core). Our model must therefore take

into account the difference in performance between EC2 instance types. However, we also observe that not all EC2 instances of the same type are equal, as there are performance differences between instances of the same type. EC2 instances of the same type vary in performance depending on region. We augment the initial SOA performance model with a scaling factor to capture the difference in performance between bare hardware and the target EC2 instance type. We rely on observations that there is a difference in speed between Intel and AMD CPUs, that CPUs have a different speed, and that EC2 instances of the same type are made up of a proportion of both Intel and AMD CPUs. [11-15]. Including the virtualization overhead scale factor from scenario 2 (EC2 uses Xen virtualization) we introduced scale factors for each instance type into the model and obtained estimates of resource requirements in terms of maximum number of EC2 instances (Figure 1), and total EC2 instance hours per day (not shown).

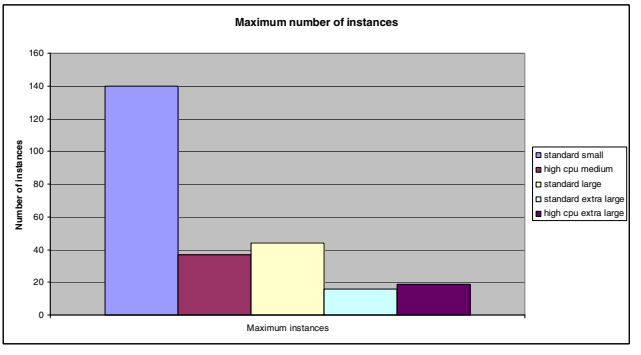

Fig. 1. Maximum EC2 Instances

Amazon offers three pricing options [16]: On-demand, reserved, and spot prices. Billing is always per instance hour (rounded up). We computed and compared two different options. Option 1 is all on-demand pricing. Option 2 is a mixture of reserved instances (for base load) and spot instances (for peak loads). As spot instances can be terminated without warning [19] the application needs to be re-designed to be fault tolerant. On-demand prices were based on an average of windows instances for

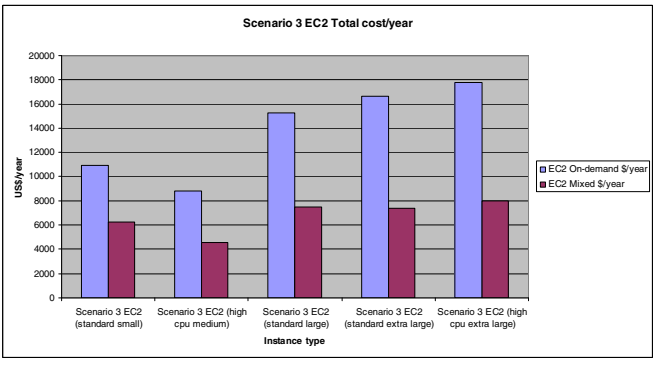

Fig. 2. EC2 cost per year

all regions, reserve prices were California (Linux only available), and spot prices were based on an average price over several months [17]. Including network and CloudWatch [18] (necessary for Auto Scaling) prices (a total of $1,300/year) the total estimated costs per year are graphed in Figure 2. There are significant price differences between instance types, and between pricing options. The most expensive is on-demand EC2 high cpu extra large instances (US$18,000) and the cheapest is mixed price high cpu medium instances (US$4,500).

Scenario 3: Cloud Hosting, Google AppEngine. The Google AppEngine [20] uses a different approach to EC2 for resourcing as the infrastructure automatically scales. It doesn't use virtualisation (although Java applications run in a Java Virtual Machine giving application isolation but not resource isolation). Consequently there are limitations to what can be done by an application to ensure isolation (e.g. no threads, only a subset of the Java APIs are permitted), and there is less visibility into the infrastructure for modelling. Billing is in terms of a reference CPU (1.2GHz Intel), but as we could not determine what the physical hardware speed was we assumed it was the same speed and type of CPU as the in-house model. We assumed that automatic scale-up is instantaneous (which is unlikely, and there is no mechanism available to ensure resources are available in advance of load spikes [23]). CPU billing is finer grained as it is measured cycles and reported in seconds. There are also more complex limits/quotas, and free and billable thresholds [21, 22].

The model estimated 44.8 physical (not reference) CPU hours/day, and the total price per year including data costs ($3,000) of US$4,723/year which is comparable to the cheapest EC2 option. The main benefit with Google AppEngine pricing is that it is very fine grained, so you really do only pay for what you consume (per CPU cycle), and there's no overhead (cost or effort) to scale and manage it.

Scenario 3: Cloud Hosting, Microsoft Azure. The Microsoft Azure platform provides a similar resource model to Amazon EC2. It uses virtualization, has different instance sizes (with 1, 2, 4, or 8 cores), and is hosted on 1.9GHz AMD CPU's. Billing is per instance hour (rounded up) for 1.6GHz reference CPUs (for simplicity we assume billing is really done for 1.9GHz CPUs as this was the speed of CPUs used). Azure offers different instance roles: Web (external facing clients), worker (back-end), and AppFabric (orchestration, security, service bus etc). For resource estimation we assume that only Web instances and workers are used, but for pricing we include

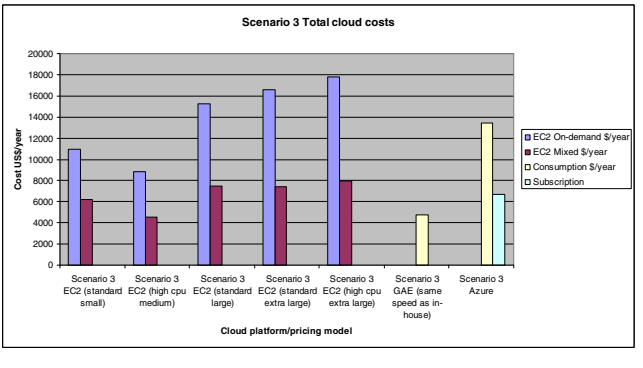

Fig. 3. Cloud scenarios cost comparison

AppFabric access control (transaction) and service bus (connection) costs, and data costs ($4,500), giving a total price of US$13,417/year. Data cost was based on Asian area costs, but ignored difference in peak/off-peak times. Microsoft has a number of promotional offers such as a 6 month subscription which may give up to a 50% saving compared with consumption rates, reducing the yearly cost to $6700, on par with some of the EC2 mixed pricing options (Figure 3).

4 Performance

For performance modeling we wanted to capture some aspects of the difference and variability in performance between and within cloud platforms and in-house hosting. Rather than modeling these directly as a consequence of resource limitations, we only attempted to model them in terms of increases in times based on assumptions about times and distributions. We will examine these in terms of Server, LAN, and WAN times.

Server performance variability. The majority of cloud platforms use virtualization to ensure that applications are isolated from each other, and to guarantee a minimum dedicated allocation of resources to each virtual core. EC2 and Azure use virtualization, and we assume that each virtual core is given a dedicated physical core (except EC2 small instances which has half a physical core). However, this only guarantees CPU. Virtual machines on the same server compete for other resources including memory and IO bandwidth. We assume that for EC2 and Azure the virtualization is "good enough" to ensure reasonably consistent performance. However, for Google AppEngine, which does not use virtualization but some other resource scheduler, we introduced an intermittent (1% chance) performance overhead of up to 5 times to capture observations about longer time due to competition for resources and as an artifact of the CPU scheduler [25]. This may be an underestimate of performance variability as some results suggest up to 50 times longer times in 20% of cases [24].

LAN latency and bandwidth. For resource modeling (Section 3) we assumed that the entire application is deployed in each virtual machine instance. However, in practice this is unlikely to be feasible in all situations, and a distributed application will be necessary [32]. This may impact on resource usage (but minimally, as the peak usage is the dominant factor), but will definitely impact performance due to inter-component communication latencies. Studies have reported good (but not as high as theoretically available) LAN bandwidth between EC2 machines [27], but the latency is significant and increases when multiple demanding applications are sharing the same LAN segment or server, when instances aren't in the same geographical location, and when there is high packet loss [26, 28, 29]. The typical inter-instance latency (in the same zone) is between 1ms and 1s (average 50ms). We introduced this latency distribution to the model for every interaction between services on different instances, and a delay for the amount of data transferred between instances (assuming 500Mbps bandwidth). This is only an attempt to model the average delays, as in the worst case the LAN may be heavily congested with latencies increasing dramatically for extended periods of times. Users have reported that they have to kill instances and start new instances in an attempt to find servers that work better [30]. The latency also probably depends on the inter-instance protocol used. For example, AWS Queue

service latency is reported at 1.5s [31], making this a poor choice of inter-instance protocol for systems where service times are sub-second.

Figure 4 shows the minimum and maximum response times predicted (across all three external services) for selected scenarios. The in-house hosting has the best times of between 613ms and 16.9s, with Cloud hosting options showing an increase in minimum response times (1.9s seconds for EC2 small instances) and maximum response times (Google AppEngine 68.5 seconds). Assuming the system is required to support a SLA with a maximum response time for any service of 30s, none of the cloud options are compliant.

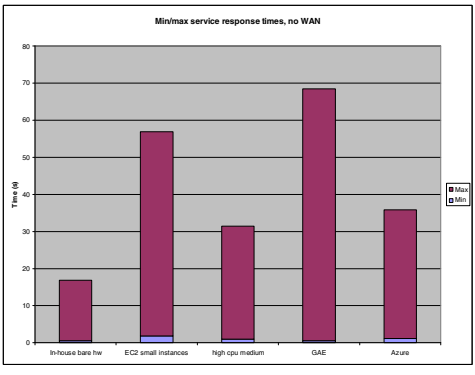

Fig. 4. Minimum and maximum service response times

WAN latency and bandwidth. The above performance results ignored WAN latency and bandwidth aspects. We added latency and transfer speeds to the model, initially assuming 100Mbps WAN bandwidth for data transfers for both continental (in-house), and inter-continental data transfers (for cloud scenarios), and identical transfer rates in both directions (Figure 5).

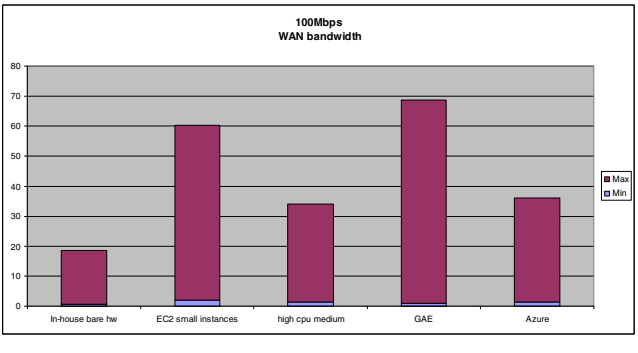

Fig. 5. Response times with 100Mbps WAN

However, based on our own measured results and 3rd party monitoring information we quickly concluded that because of large inter-continental latencies, high theoretical bandwidths, non-zero packet loss, and asymmetries in upload and download

speeds, a more achievable upload speed is around 0.5Mbps with download speed only marginally better at 1Mbps. The predicted maximum response time for in-house hosting with 100Mbps bandwidth WAN (continental) is 18s, but a whopping 271s (almost 5 minutes) for USA hosted EC2 (inter-continental). For the slowest service (external service "3" with 10Mb file) the WAN transfer time makes up the bulk of the response time (88%).

This is a stark demonstration of the so-called "Skinny Straw" problem with Clouds [33] – the fact that high latency, high bandwidth networks don't achieve anywhere near their maximum theoretical speed with standard TCP protocols and configurations.

5 Platform Limits and Quotas

In this section we explore which scenarios would actually work correctly given published platform limits and quotas. EC2 has a limit of 20 concurrent on-demand or reserved instances, or 100 spot instances. For the on-demand price simulations, only the standard extra large and high cpu extra large instances are under the 20 on-demand instance limit. Using mixed (reserved and spot-instances) pricing model, all instance types except small instances are under the 100 spot instance limit (Figure 1).

Because Google AppEngine doesn't use virtualization to isolate applications/users there is a risk that some applications will act as resource hogs and attempt to use more than their "fair" share of resources. The approach AppEngine takes to limit this behaviour is to impose multiple limits/quotas on each user. These are relatively complex to understand, and include total, daily, minute and instantaneous limits. Users can request an increase in limits/quotas, but to do so they need to be able to estimate which limits will be exceeded and the new limits to request as follows:

- AppEngine permits a maximum of 30 concurrent requests. The model predicts a maximum of 59 concurrent users (for all services), exceeding the limit.
- The maximum CPU consumption rate is 72 CPU minutes/minute (in terms of reference CPU times), and the model predicts 40 CPUs (physical), so multiplying by 2 gives 80 reference CPU minutes/minute, exceeding the limit.

For Azure subscription accounts there is a maximum of 20 hosted service projects, 5 roles per hosted service, and 20 VM CPU Cores [35]. The limit of 20 VM CPU Cores is exceeded as 72 Cores are predicted by the model for the peak load.

6 Observations and Conclusions

Our modeling approach, and many of the cloud platforms, give good visibility into the operational costs of an application, and can capture costs for various resource types and load such as the cost of CPU (total, and due to different workloads and base and peak loads), network/data, security operations, transactions, orchestration, connections, etc.

Different cloud charging and billing models allows choice of hosting options between and even within cloud platforms. However, this adds both flexibility and complexity, which modeling may assist with.

At face value a consumption-based charging model can reduce costs. However, it may also focus too much attention on cost as an architectural driver. Optimizing applications for very specific costing models may result in vendor lock in and lack of flexibility and maintainability.

Is cloud hosting cost effective? Our predictions are that cloud costs are comparable to the cost of power alone for in-house hosting. However, our pricing was not exhaustive, and there are likely to be other hidden costs. The final decision about cost/risk benefits must include the cost of migrating applications to the cloud, added maintenance and management complexity due to remote hosting, increased chance of outages due to failures in cloud and network infrastructures (and therefore loss of customer confidence and possibly financial penalties for SLA violations), and importance and economic return of the services offered.

Does performance matter? From experiments and modeling it is obvious that cloud performance is likely to be worse and substantially more variable than in-house hosting. But is it "good enough"? This obviously depends on the type of application, if there are hard response time requirements for services (e.g. due to constraints in the physical world or software of the consumers), or if SLAs must be satisfied. For user facing applications there are maximum response times above when users are impacted and user behavior and satisfaction changes, resulting in abandonment or complaints. Depending on the importance of the task, prior expectations, and availability of alternatives, these times can range from seconds to tens of seconds. Some of these issues can be reduced by changing the way users interact with the services (e.g. allowing asynchronous interactions, providing constant feedback about time to completion, allowing concurrent interactions, etc). For machine-to-machine interactions there are issues such as client-side timeout settings and retry behavior (e.g. which may result in increased load on the services and duplicate transactions).

In most situations it is valuable to offer explicit SLAs per service, and the modeling approach assists with this. Another approach is to offer tailored SLAs for different classes of users of a set of services. This could be enabled by the use of virtualization in most cloud platforms. A set of services could be deployed on a virtual machine and configured and resourced in such a way that a SLA can be guaranteed for a sub-class of users. Fractional and elastic resourcing would assist with ensuring the SLA would be met for a variety of loads.

It may be non-trivial to architect distributed applications for performance and scalability, as the interactions between different components may be complex (and inter-instance latency combined with the number of interactions can be a significant overhead), and resource requirements for each component may differ substantially as loads change. There may be complications with load-balancing across multiple instances of the same component type, and applications will need to be architected to be reliable with lower availability than normal. There may also be components (e.g. databases, 3rd party services, legacy services) that just do not scale well, or have fixed maximum throughputs. Hybrid applications (hosted on both in-house and external clouds) present challenges, particularly due to the cost and low speed of data transfer between distributed components in and out of the cloud (e.g. data will need to be moved to the cloud once and then used in the cloud as much as possible before moving back to the enterprise), and deciding if (and which) components can sensibly be hosted on the cloud (e.g. due to dependencies and inter-connectively to other

systems, security and privacy, difference in performance between enterprise and cloud hosting, etc).

SLAs offered by cloud providers are weak, limited in scope, and don't guarantee availability of all resource types. For EC2 and Azure a dedicated number of physical CPUs (or part therefore for EC2 small instances) is all that is guaranteed. Better SLAs would be valuable for network and IO resources to deal with problems of resource contention in cloud platform infrastructure and applications sharing the same servers. For Google AppEngine, which does not use virtualization, there is no guarantee even of CPU (particularly if the application is CPU intensive as it's priority may be reduced), response times may blow out (due to competing applications, if an application is only intermittently used and is put to sleep, and the first time a new instance of an application is used (e.g. due to JVM startup, lazy loading of application code and data). On the other hand, multiple (complex) limits are imposed in order to attempt to reduce resource contention and limit the impact of greedy applications. Unfortunately, for enterprise applications, the risk of un-predictable performance and exceeding limits (many of which are difficult to relate to physical values, such as CPU time, resulting in application exceptions and periods of unavailability until resources are replenished or can be raised upon request) are likely to be too restrictive. The Google AppEngine programming model may also be too restrictive for enterprise applications as only a subset of Java APIs and enterprise Java Edition technologies are supported [36]. Enterprise Java Beans (EJBs) are not supported due to lack of SQL support, although JDO/JPA (which maps data objects to datastore entities [37]) are supported, and may provide an alternative and scalable persistence model for the cloud.

The ability of cloud platforms to ramp-up with increased load was not tested or modeled. If a cloud platform supports the ability to control the number of instances in use and start/stop instances on demand, then this adds complexity for monitoring and managing to ensure that instances are available when needed. If elasticity is automatic (E.g. Google AppEngine, or Amazon Elastic Load Balancing) then it is important to know what the limitations are through SLAs with the cloud provider and performance evaluations and modeling, so that SLAs with clients can be managed and approaches to limit load rate increases considered.

Modeling is a potentially powerful tool to understand and compare performance, scalability, cost benefits, and risks of various cloud platforms, and hosting and deployment options. If parameterized with performance data obtained from real cloud platforms the models will provide more accurate insights into performance, costs, benefits and risks of critical architectural options than existing Cost of Ownership calculators (e.g. provided by Azure). Further experimentation and research is required to determine if cloud platforms are as scalable as assumed, and if elastic infrastructures can scale quickly enough to cope with load spikes. Unpredictable periods of reduced availability also need to be reckoned with. We plan to enhance our modeling environment to enable direct modeling and simulation of more dynamic properties of clouds platforms.

Acknowledgements

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. Brebner, P., O'Brien, L., Gray, J.: Performance Modeling for e-Government Service Oriented Architectures. In: Experience Report Proceedings, ASWEC 2008, pp. 130–138 (2008)
2. Brebner, P.: Performance modeling for service oriented architectures. In: ICSE Companion 2008, pp. 953–954 (2008), doi: http://doi.acm.org/10.1145/1370175.1370204
3. Brebner, P., O'Brien, L., Gray, J.: Performance Modeling Evolving Enterprise Service Oriented Architectures. In: WISA/ECSA 2009, pp. 71–80 (2009), doi: 10.1109/WICSA.2009.5290793
4. Brebner, P.: Service-Oriented Performance Modeling the MULE Enterprise Service Bus Loan Broker Application. In: SEAA 2009, pp. 404–411 (2009), doi:10.1109/SEAA.2009.57
5. Brebner, P., O'Brien, L., Gray, J.: Performance Modeling Power Consumption and Carbon Emissions for Server Virtualization of Service Oriented Architectures. In: Proceedings of the IEEE EDOC 2009 Workshops, Middleware for Web Services Workshop 2009, pp. 92–99 (2009), doi:10.1109/EDOCW.2009.5332010
6. Wood, T., Cherkasova, L., Ozonat, K., Shenoy, P.: Predicting Application Resource Requirements in Virtual Environments. HP Laboratories, Technical Report HPL-2008-122 (2008), http://www.hpl.hp.com/techreports/2008/HPL-2008-122.html
7. Amazon Elastic Load Balancing, http://docs.amazonwebservices.com/ElasticLoadBalancing/latest/DeveloperGuide/
8. Amazon Auto Scaling, http://aws.amazon.com/autoscaling/
9. Amazon EC2, http://aws.amazon.com/ec2/
10. EC2 Instance Types, http://aws.amazon.com/ec2/instance-types/
11. Amazon EC2 benchmark, http://www.mnxsolutions.com/blog/linux/amazon-ec2-benchmark-pystone.html
12. Testing Cloud Computing Performance with PRTG: Performance Comparison of Amazon EC2 Instance Types, http://www.paessler.com/blog/2009/04/03/prtg-7/testing-cloud-computing-performance-with-prtg-performance-comparison-of-amazon-ec2-instance-types/
13. Lamp performance on the elastic compute cloud: benchmarking drupal on amazon ec2, http://www.johnandcailin.com/blog/john/lamp-performance-elastic-compute-cloud:-benchmarking-drupal-amazon-ec2
14. MySQL on EC2 Part 1: are all instances equal?, http://www.infibase.com/blog/2009/07/mysql-on-amazon-ec2-part-1/
15. Walker, E.: Benchmarking Amazon EC2 for high-performance scientific computing, http://www.usenix.org/publications/login/2008-10/openpdfs/walker.pdf
16. EC2 Pricing, http://aws.amazon.com/ec2/pricing/
17. EC2 spot price history, http://cloudexchange.org/
18. EC2 Cloudwatch, http://aws.amazon.com/cloudwatch/
19. EC2 Spot instances, http://aws.amazon.com/ec2/spot-instances/
20. Google AppEngine, http://code.google.com/appengine/
21. Google AppEngine billing, http://code.google.com/appengine/docs/billing.html
22. Google AppEngine quotas and limits, http://code.google.com/appengine/docs/quotas.html

23. Google AppEngine - a second look, http://highscalability.com/google-appengine-second-look
24. Twitmark ported OFF Google AppEngine, http://mrblog.org/2009/10/16/twitmart-ported-off-of-google-app-engine/
25. CPU (Fibonacci) Latency, http://code.google.com/status/appengine/detail/serving-java/2010/01/30#ae-trust-detail-cpu-fibonacci-java-latency
26. Barker, S., Shenoy, P.: Empirical Evaluation of Latency-sensitive Applications Performance in the Cloud, http://none.cs.umass.edu/papers/pdf/mmsys10-latency.pdf
27. Alexandru-Dorin, G.: Network performance in virtual infrastructures – A closer look at Amazon EC2, http://staff.science.uva.nl/~delaat/sne-2009-2010/p29/presentation.pdf
28. Measuring EC2 system performance, http://tech.mangot.com/roller/dave/entry/ec2_variability_the_numbers_revealed
29. EC2 Benchmarks, http://www.wafflegrid.com/wiki/index.php?title=Ec2_Benchmarks
30. http://alan.blog-city.com/has_amazon_ec2_become_over_subscribed.htm
31. http://cloudstatus.com/
32. Are Clouds ready for large distributed applications?, http://www.cs.cornell.edu/projects/ladis2009/papers/sripanidkulchai-ladis2009.pdf
33. Bolden, B.: The Skinny Straw: Cloud Computing's Bottleneck and How to Address it, http://www.cio.com/article/499137/The_Skinny_Straw_Cloud_Computing_s_Bottleneck_and_How_to_Address_It
34. Aspera on-demand, http://www.asperasoft.com/en/products/on_demand_17/aspera_on_demand_for_AWS_17
35. Azure instance limits, http://blog.toddysm.com/2010/01/windows-azure-role-instance-limits-explained.html
36. Google AppEngine support for Java, http://groups.google.com/group/google-appengine-java/web/will-it-play-in-app-engine
37. Google AppEngine datastore, http://code.google.com/appengine/docs/java/datastore/overview.html