Practical Performance Control for Web Applications in the Cloud

Corina Stratan, Guillaume Pierre, Hector Fernandez, Eliana Tirsa and Valentin Cristea

Vrije Universiteit Amsterdam and University Politehnica of Bucharest

Other versions for the title:

Automatic Scaling for Cloud Web Applications in Practice

A Practical Approach for Automatic Scaling (Resource Provisioning?) of Web Applications in the Cloud

Implementing Automatic Scaling for Cloud Web Applications ;

Introduction

Why is automatic scaling in the cloud important/useful? (give a real-world example)

Current possibilities for users who want to run web applications in the cloud:

1) Systems currently available in clouds: most of them define triggers for adding and removing VMs from the application, based on the monitoring data available in the cloud. Most of these systems are recent and there is a lack of documentation on best practices and performance reports of web applications using these systems.

2) At the other end of the spectrum: several research works on resource provisioning, proposing more sophisticated techniques based on queuing models, prediction and workload analysis. However most of these research techniques are tested with synthetic (or partially synthetic) benchmarks and it is difficult to estimate how they would perform for a real-world application; also not all of them are compatible with the current cloud environments.

Our goal is to bridge the gap between the state-of the art research in resource provisioning and the real-world web application deployments. In order to achieve this goal:

- we built a Platform-as-a-Service system that can be readily used on top of some of today's most popular clouds (Amazon EC2, OpenNebula)
- we integrated and adapted in our system some resource provisioning mechanisms developed in previous research from our group [Jiang]
- we tested our system with a Wikipedia deployment and access traces

Guillaume's version:

1. *The cloud is a great place to run Web application. In particular it opens the door to resource provisioning, since computing resources are available on-demand.*
2. *There are lots of research papers dedicated to sophisticated techniques to handle resource provisioning. However, if we look at real deployments we see that they rely on extremely simple techniques, and completely ignore the results from academic research on the topic.*
3. *There can be two explanations for this discrepancy: (i) the gains of using sophisticated techniques are too low for anyone to bother; (ii) implementing these techniques is a difficult exercise, which is why real cloud systems rely on simpler techniques.*
4. *This paper tries to identify the real cause. We do this by implementing a sophisticated provisioning system in realistic conditions, and reporting on (i) how hard implementation was; and (ii) potential gains from using the better technique as compared to a simple strawman.*

Related work

B. Urgaonkar et al., Agile Dynamic provisioning of Multi-Tier Internet Applications: They use predictive and reactive provisioning. They propose a queuing model for multi-tier applications, but I don't think it's very accurate (they use a G/G/1 queue, and don't take into account processor sharing at all. At the end there is a paragraph to address multi-threaded servers, probably added at the request of a reviewer; but it doesn't match with what's in the previous paragraphs). For testing they use Rubis and Rubbos. To generate workload for Rubis they use some processed access traces from a real-world site. The interesting part: they did have flash crowds in their traces.

Z. Whang et al., AppRAISE: Similarly to the previous paper, they use predictive and reactive provisioning. But they work at the hardware level, adjusting the CPU capacity allocated to VMs. They also use a queuing model for this. For testing they use Rubis, but with processed traces from a real-world application.

[Homogeneous experiments] [No real traces]

H. Zhang et al., Resilient Workload Manager: This paper splits the workload in "base workload" and "tresspassing" (flash crowd) workload. These workloads are served by different groups of servers. They attempt to divide the data items into popular and less popular, and place them in the right group of servers. It's not clear how they do the actual provisioning.

[Heterogeneous experiments] [Real Traces] [No profiling]

N. Kaviani et al., Profiling-as-a-Service: The aim of this paper is to provide an application profiling service. To do this, they substitute at runtime a regular VM from the application with a VM that has the same application, but with profiling instrumentation. They don't have much results and also not much details about how the profiling is done.

[Homogeneous experiments] [Use of profiling techniques] [No Real traces]

http://www.aschroder.com/2012/01/using-aws-auto-scaling-with-an-elastic-load-balancer-cluster-on-ec2/ A blog post with advice on setting up AWS auto-scaling.

ConPaaS overview

Make an overview of the base ConPaaS system, without details on provisioning.

- the ConPaaS manager and agents
- Ganglia integration
- the provisioning / profiling manager

Possibly make a diagram with all these components.

The Wikipedia workload

Describe the application and the workload traces:

- structure of the Mediawiki application
- content and size of the DB
- the access traces and how Wikibench works
- the PHP pages take significantly more time to process than the web pages
- each PHP page needs a lot of DB queries
- the pages vary in complexity so it is difficult to make predictions (maybe compute the standard deviation of the response time from a trace to show this)

Basic resource provisioning

Explain the trigger-based provisioning method available in EC2 (e.g., add new machines whenever the load in the last 10 mins exceeds a threshold). Show an experimental result of it and explain why it doesn't work so well:

- it doesn't take into account application-specific metrics

- it is particularly difficult to decide when to remove VMs and to avoid repeated adding/removing

How can basic provisioning be improved: use previous knowledge about the behaviour of the application, and about how much workload a server can handle; for example: what is the request rate that a server can sustain without becoming overloaded? what is the value of the CPU utilization/load that indicates that a server is overloaded? These values are different from one application to another, and also from one server to another.

Second experiment: improved the basic provisioning with some application knowledge observed empirically (the maximum request rate we can send to a server, maybe also the maximum CPU utilization we can allow). The results show more stability in the provisioning.

Profiling-based resource provisioning

Problems with the basic approach:

- it requires previous knowledge of the application performance behaviour
- the VMs are heterogeneous in performance
- the performance behaviour of the application can also change in time if the type of workload changes

Solutions for these problems, proposed by the previous research in our group:

- a quick offline profiling of new VM instances to have an initial assessment of their performance that can be used for predictions
- an online profiling after the application started, to adjust the weighted load balancing

In addition to these, we propose a mechanism to continuously adapt the online profile, in order to deal with possible changes in workload type or in VM performance. The online profile is used to dynamically adapt the load balancing weights and also our performance predictions.

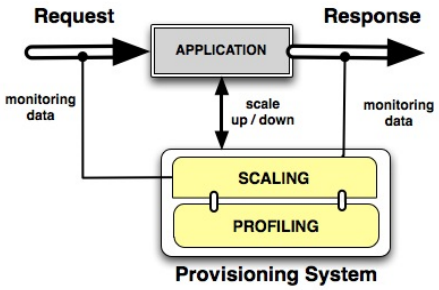

Figure 1: Profiling Resource Provisioning

- experiment with profile-base provisioning
Issues:

- unlike in some synthetic benchmarks, in Wikipedia there are large variations in the complexity of the articles; so the PHP processing time is more difficult to predict

Experimental Evaluation

0.1 Homogeneous Infrastructure

0.1.1 Discussion

0.2 Heterogeneous Infrastructure

0.2.1 Discussion

0.3 Discussion

Conclusion

Excesive reactive algorithms trend to temporally overprovision applications affected by flashcrowds or slashdot effects. This type of algorithms increase the resource consumption and infrastructure costs than other ...

Experiments based on real-traces "Wikipedia" and conducted on heterogeneous and homogeneous cloud infrastructures.

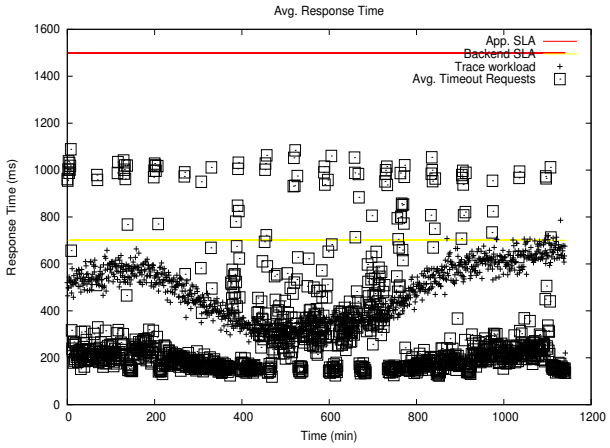

Figure 2: Average of response timeout – Naive Alg.

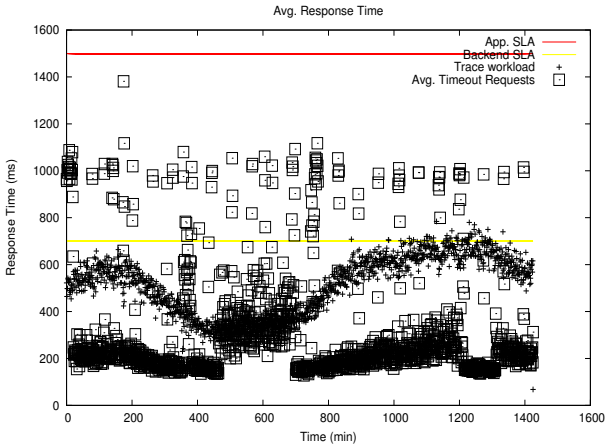

Figure 3: Average of response timeout – Smart Alg.

The use of offline-profiling techniques allow to identify the threshold of the resources of a cloud infrastructure.

The provisioning system remains independent of the infrastructure on which the apps run.

Acknowledgments

This work is partially funded by the FP7 Programme of the European Commission in the context of the Contrail project under Grant Agreement FP7-ICT-257438.

Probably also mention ERRIC here.

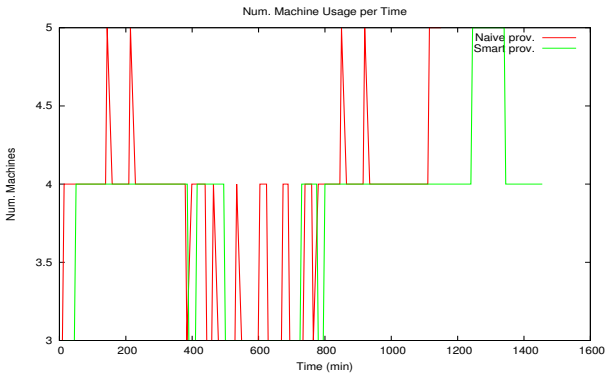

Figure 4: Resource consumption