Automated staging testing framework for Amazon EC2 (Enhancing Elba in to EC2 with MySQL Cluster)

Deepal Jayasinghe, Karsten Schwan and Calton Pu

Georgia Institute of Technology
266 Ferst Drive, Atlanta, GA 30332-0765, USA
`{ijayasin, schwan, calton}@cc.gatech.edu`

Abstract. Increasing complexity of large enterprise and distributed application systems makes the management of the system is an increasingly important and increasing expensive technical challenge. Staging comes as a handy tool when solving those important challenges. The key advantage in staging is, it provides system developers and system administrators to monitor and tune new deployment configuration under simulated work condition before the system goes into production. The goal of Elba project is to verify and test an application system deployment plan in a staging environment before committing it to a production environment. Manual verification of a deployment is cumbersome, time consuming, and error prone. This problem will grow in importance in the deployment of increasingly larger and more sophisticated applications (eg. Data centers). Therefore, it will be increasingly important to have an automatic method for executing a benchmark similar to enterprise applications on the deployment plan to validate the deployment during staging, instead of debugging a deployment during production use. With commencement of cloud computing, industries are moving towards cloud computing environment considering the benefits they gain form using it. In the commercial setting Amazon EC2 (Elastic Compute Cloud) is one of the commonly used cloud computing system, as a result of that executing benchmark and analyzing the results on Amazon EC2 is undoubtedly useful.

1 Introduction

Scalability of N-tier applications in general and database servers and application servers in particular present serious challenges to both academia and industry. This is largely due to the requirements such as non-stationary workloads and dependencies created by requests that are passed between web servers, application servers and database servers. In Elba [5] we attempt to start bridging these gaps with a large-scale experimental evaluation of server scalability using a N-tier application benchmark (RUBBoS [1]). For our study we have collected more than 600GB of data so far. These experiments cover scale-out scenarios with up to nine database servers and three application servers. The configurations were varied using two relational database management systems (MySQL and PostgreSQL), workloads ranging from 1,000 to 13,000 concurrent users, and two different hardware node types. The hardware infrastructure for all those existing data is Emulab network testbed.

Previously, we have been relying on Emulab testbed and we have been using middleware-based replication (i.e., C-JDBC [2]) in our experiments. In this paper we present preliminary results obtained with MySQL clustering [4] technology both in Emulab and Amazon EC2 [3], which has been receiving growing attention recently. MySQL Cluster is a high-availability, high-redundancy version of MySQL, which has been adapted distributed computing environments. It uses the NDBCLUSTER storage engine to enable running several node with MySQL servers in parallel. The presented experiments' data are comprised of a full scale-out mesh with up to four SQL nodes, up to two application servers, and user numbers that range between 1,000 and 13,000 concurrent users. Additionally, the four-tier systems were exposed to two different workload-mixes (browse-only and read/write). The initial analysis of our extensive data produced several interesting findings.

The paper makes two main contributions. First, using an automated experiment creation and management system, to measure the scalability and performance in Amazon EC2. Second the scalability and performance analysis of MySQL cluster.

The remainder of this paper is structured as follows. Section 2 provides a background information about Elba project, Amazon EC2 and MySQL cluster. Implementation details and implementation challenges are discussed in Section3, and Section 4 provides a experiment setup as well as experiments results on both Emulab and Amazon EC2. Problem we encounter during the experiment and the solution we have found are described in section 5. Experiment procedure and how to run experiment is discussed in section 6, in section 7 we discussed about the future direction of the project. And in section 8 we conclude the paper.

2 Application Background

2.1 Elba overview

The goal of the project is to provide a thorough, low-cost, and an automated approach to staging that overcomes the limitations of manual approaches and recaptures the potential value of staging. In Elba processes there are three major components when automating the staging , first the application, second the workload, and third quality of service requirements. One of the main research challenges is the integrated processing of these different specifications through the automated staging steps. In other words we need to merge Service Level Agreement (SLA), Test Based Language (TBL) and workflow into the automating process and come up with a specification for the application. Specification of the production applications and their execution environments consists of a number of research challenges. First, automated re-mapping of deployment locations to staging locations, second creation of consistent staging results across different experiments, third extensibility to many environments and applications.

The overall architectural flow of Elba project for the RUBBoS benchmark is shown in Figure 1, where Elba achieves full automation in system deployment, evaluation, and evolution, by creating code generation tools to link the different steps of deployment, evaluation, reconfiguration, and redesign in the application deployment life cycle. A cyclical view of staging allows feedback from execution to influence design decisions

before going to production. The figure shows how new and existing design tools can be incorporated in the staging process if their specification data can be transformed to support staging.
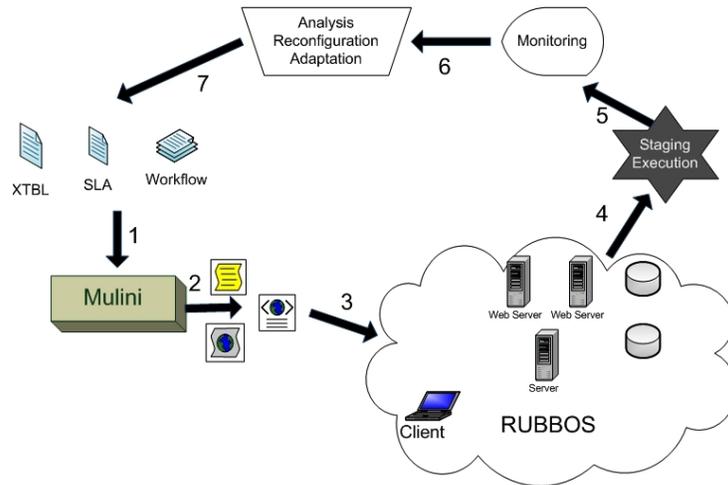

Fig. 1. The architecture of Elba project and Staging process with Mulini code generator (for RUBBoS benchmark)

1. Developers provide design-level specifications , Test plan and service agreement
2. Mulini creates a provisioning and deployment plan for the application (configuration , scripts etc..)
3. Deploy the generated client and server application.
4. Execute the staging
5. Collect and Monitor the execution results
6. Manually analyze the data to figure out what to change to get the expected output.
7. Change the input configurations to meet the expectation.

2.2 Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers. Amazon EC2s simple web service interface allows us to obtain and configure capacity with minimal friction. It provides us with complete control of our computing resources and lets us run on Amazons proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing us to quickly scale capacity, both up and down, as our computing requirements change. Amazon EC2 changes the economics of computing by allowing us to pay only

for capacity that we actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios.

Amazon EC2 presents a true virtual computing environment which runs on highly modified and optimized version of XEN hypervisor, allowing us to use web service interfaces to launch instances with a variety of operating systems, load them with our custom application environment, manage our networks access permissions, and run our image using as many or few systems as we desire

Features of Amazon EC2

- **Elastic** : Amazon EC2 enables you to increase or decrease capacity within minutes, not hours or days. You can commission one, hundreds or even thousands of server instances simultaneously. Of course, because this is all controlled with web service APIs, your application can automatically scale itself up and down depending on its needs.
- **Completely Controlled** : You have complete control of your instances. You have root access to each one, and you can interact with them as you would any machine. Instances can be rebooted remotely using web service APIs. You also have access to console output of your instances.
- **Flexible** : You have the choice of several instance types, allowing you to select a configuration of memory, CPU, operating system, and instance storage that is optimal for your application.
- **Designed for use with other Amazon Web Services** : Amazon EC2 works in conjunction with Amazon Simple Storage Service (Amazon S3), Amazon SimpleDB and Amazon Simple Queue Service (Amazon SQS) to provide a complete solution for computing, query processing and storage across a wide range of applications.
- **Reliable** : Amazon EC2 offers a highly reliable environment where replacement instances can be rapidly and reliably commissioned. The service runs within Amazons proven network infrastructure and data centers.
- **Secure** : Amazon EC2 provides web service interfaces to configure firewall settings that control network access to and between groups of instances.
- **Inexpensive** : Amazon EC2 passes on to you the financial benefits of Amazons scale. You pay a very low rate for the compute capacity you actually consume. Compare this with the significant up-front expenditures traditionally required to purchase and maintain hardware, either in-house or hosted. This frees you from many of the complexities of capacity planning, transforms what are commonly large fixed costs into much smaller variable costs, and removes the need to over-buy "safety net" capacity to handle periodic traffic spikes.

Amazon EC2 Flow .
The process of using Amazon EC2 consists of following key steps;

- Create an Amazon Machine Image (AMI) containing our applications, libraries, data and associated configuration settings, keys and etc...(Apache, Tomcat, MySQL, JDK, knowhosts etc)

- Upload the AMI into Amazon S3. Amazon EC2 provides tools that make storing the AMI simple. Amazon S3 provides a safe, reliable and fast repository to store our images.
- Use Amazon EC2 web service to configure security and network access.
- Choose which instance type(s) and operating system we want, then start, terminate, and monitor as many instances of our AMI as needed, using the web service APIs or the variety of management tools provided.
- Pay only for the resources that we actually consume, like instance-hours or data transfer.

2.3   MySQL Cluster

The information in this section is directly obtained from the manual pages of the MySQL website [6]. Please refer to this source for comprehensive and complete documentation. Note that we omit further explicit referencing in the following for ease of reading and simplicity.

High availability is a core requirement for all organizations that depend on real time online enterprise systems to sell their products, service their customers, and do business with partners. Consequently, system downtime immediately translates into lost revenue and increased costs such as lost orders and increase call center load. Using MySQL Cluster, organizations can make their core IT systems highly available. The core advantage of MySQL Cluster compared to other industrial solutions is its ability to run on commodity hardware, which leverages hardware, network, application, and system failures.

Why MySQL Cluster? .
MySQL Cluster is a real-time open source transactional database designed for fast, always-on access to data under high throughput conditions. MySQL Cluster utilizes a *shared nothing* architecture, which does not require any additional infrastructure and provides 99.999% data availability with no single point of failure. MySQL Cluster is most often used as an in-memory database, but can be configured to use disk-based data as well. More concretely, MySQL Cluster is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. It uses the NDBCLUSTER storage engine to enable running several nodes with MySQL servers in parallel.

MySQL Architecture .
In order to achieve fault tolerance a number of distributed components are used to run the application, access the database, and manage the distributed environment are required. These components are typically distributed across machines and continuously share the latest changes to the database with one another. If a node or machine fails, there are other nodes that contain the same information so an application can continue to run without interruption. The system must also ensure that transactions are successfully executed and database information remains consistent throughout the recovery process. MySQL Cluster consists of three kinds of nodes which replicate data, access

the database, as well as manage the system. The overall architecture of MySQL cluster is shown in Figure2.
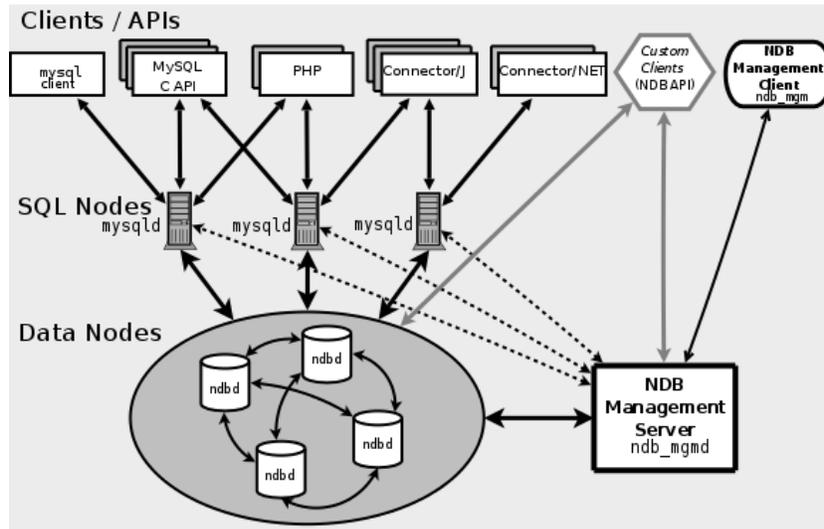

Fig. 2. MySQL cluster overall architecture

- *Management node (MGM node)*: The role of this type of node is to manage the other nodes within the MySQL Cluster, performing such functions as providing configuration data, starting and stopping nodes, running backup, and so forth. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node. A MGM node is started with the command ndbmgmd.
- *Data node*: This type of node stores cluster data. There are as many data nodes as there are replicas, times the number of fragments. For example, with two replicas, each having two fragments, we need four data nodes. One replica is sufficient for data storage, but provides no redundancy; therefore, it is recommended to have 2 (or more) replicas to provide redundancy, and thus high availability. A data node is started with the command ndbd.
- *SQL node*: This is a node that accesses the cluster data. In the case of MySQL Cluster, an SQL node is a traditional MySQL server that uses the NDBCLUSTER storage engine.

3 Implementation

In this section we will discuss the different types of implementation details on both Amazon EC2 and MySQL cluster. As we discussed in section1, previous version of Elba

was only focus on Emulab testbed, as a results all the templates which used to generate deployment and monitoring scripts were only compatible with Emulab testbed. Though we use same operating system (Fedora Core) both in Emulab and Amazon EC2, those two infrastructures have significant differences. Those differences are mainly due to the user privileges, underlying virtual machine and security restrictions. As the first step of template changes was to do a sami-automation (both manual and scripts) of deploying and running the experiment and then find the differences. After finding those differences then make the necessary code changes.

3.1 Amazon machine Image for Elba

The main focus of this section is to explain the process of creating AMI for Elba project, however the process involves a number of additional steps which are not discussed here. Amazon provides a detailed documentation on how to complete those steps. The first step of moving Elba into Amazon EC2 is to create AMI consisting all the required softwares and other configurations files. When creating images one important thing to keep in mind is on which platform we are going to run and which type of instances we are going to use. Depending on the platform and type of the instances we have to create 32-bit images or 64-bit images. Since one of the goal of moving Elba into EC2 is to experiment with different types of instances, it is required to create two different types of AMIs. One for 32-bit and one for 64-bit, both of the instances. Considering performance and usability my selection was to use Fedora Core 8 as the starting point of creating OS images. Once we create the images we can store them in Amazon S3 and use them whenever we want to run the experiment.

3.2 Axis2 Web service client for EC2

Amazon EC2 has a set of tools which we can use to start, configure and monitor the instances, and most of those tools driven by Web service API. Those tools are very convenient and useful, provided that if we are going to use them manually, however when it comes to automation those are not that useful. Therefore when moving Elba project into Amazon EC2, one of the main issue was to find a way to automate the process of instance starting, configuring and monitoring. As a result of that we had to implement a Web service client for EC2 focusing only the API that Elba project is going to use. As a part of the project I have implemented a Web service client using famous open source framework, Apache Axis2[7]. In this implementation I have only implemented the following APIs;

- runInstances
- describeInstances
- terminateInstances

Writing a Web service client itself is not a mean task considering the amount of work we need to allocate to get the client working properly. One of the major challenges was documentation issues with Amazon, most of the security requirements were not clearly mentioned in the service WSDL. As a result configuring public key and certificate was one of the most time consuming task.

3.3 Subversion as an evolution mechanism

Evolution of software is inevitable, it may be due to a number of reasons, adding new features, improvement of existing features and fixing bugs. One of the key challenge of building softwares and systems is to cope with changes in convenient manner. Even in Amazon EC2 one of the key challenge is to build AMI with all those softwares at the first time. The main reason is, building an AMI is not an easy task, it takes considerable amount of time as well money. Therefore it is a important to address the software evolution into AMI itself. In this regrades I found subversion (SVN) [8] as a perfect solution, where when I build the AMI I have set up SVN access for the instance and configure it for a SVN repository [9]. With this approach, downloading latest softwares and sources is just a matter of doing a svn update. With SVN approach it simplifies the initial setup, we can build the AMI with the software packages which are not going to be changed frequently and rest (templates and source code) configured to download from SVN. Next we can create a SVN repository and keep changing that, when starting an instance it can configured to do a svn update to get the latest code. Figure 3 illustrates how subversion works in Amazon EC2.

3.4 Experiment auto termination

Amazon EC2 is a commercial cloud, so we have to pay for each hour we use the service, as a result time management is very critical factor in EC2. When we run experiment on Emulab we do not need to worry too much about experiment terminate, because when we create the experiment we can set the idle time and after that Emulab will automatically swap out the experience. It is not possible to implement such a mechanisms in EC2, therefore one of the solution we found is to automatically shutdown all the instances we allocated for the experiment. Once we do the shutdown EC2 will automatically terminate the experiment. We need such a termination because our experiment takes 14-16 hours so it is extremely difficult to keep an eye on the experiment and terminate the experiment when it finishes.

4 Experimental Setting

4.1 Benchmark Applications

Among N-tier application benchmarks, RUBBoS has been used in numerous research efforts due to its real production system significance. RUBBoS is a N-tier e-commerce system modeled on bulletin board news sites similar to Slashdot. The benchmark can be implemented as three-tier (web server, application server, and database server) or four-tier (addition of clustering middle-ware such as C-JDBC) system. The benchmark places high load on the database tier. The workload consists of 24 different interactions (involving all tiers) such as register user, view story, and post comments. The benchmark includes two kinds of workload modes: browse-only and read/write interaction mixes. In this paper we solely use the browse-only workload for our experiments. Typically, the performance of benchmark application systems depends on a number of configurable settings (including software and hardware). To facilitate the interpretation
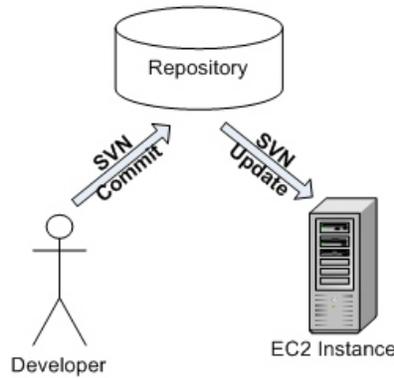
Fig. 3. Subversion in Amazon EC2

of experimental results, we chose configurations close to default values, if possible. Deviations from standard hardware or software settings are spelled out when used. Each experiment trial consists of three periods: ramp-up, run, and ramp-down. In our experiments, the trials consist of an 8-minute ramp-up, a 12-minute run period, and a 30-second ramp-down. Performance measurements (e.g., CPU or network bandwidth utilization) are taken during the run period using Linux accounting log utilities (i.e., Sysstat) with a granularity of one second.

4.2 Experiment results on Emulab

The first set of graphs demonstrates distribution of response time and throughput of various configurations on Emulab testbed. Figure 4 illustrates the throughput comparison with two tomcat servers and different number of MySQL servers for browse-only workloads, and Figure 5 illustrates the same results for read/write workloads. The throughput variation for browse-only workloads for varying tomcat servers and MySQL servers are illustrated in Figure6 , and in Figure 7 same variation for read/write work loads. The machine configuration for Emulab is as follows;

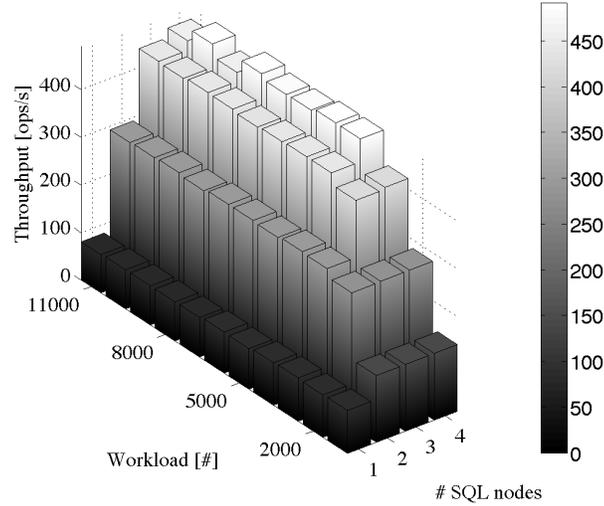| Type	Components		
Normal	Processor	Xeon 3GHz	64-bit
	Memory	2GB	
	Network	6 x 1Gbps	
	OS	2.6.12-1.1390_FC4	
	Disk	2 x 146GB	10,000rpm

Table 1. Hardware node setup for Emulab.

Fig. 4. Throughput for RUBBoS with two application servers, MySQL Cluster, and browse-only

4.3 Experiment results on Amazon EC2

Amazon EC2 is commercial cloud, so running experiment on EC2 is costly, as a results of that I have only run limited number of experiments proof the status of the project. However we are planing to conduct a set of different experiment with different configuration of both n-tier configuration as well as different type of instances. Following results demonstrate the differences in Emulab and EC2, both of them assumed to be having the same hardware configuration (though we get virtualize environment in EC2) and same experiment setup. As the figures illustrate both the response time and throughput is considerably slow in EC2. Response time comparison for browse only running the RUBBoS on Emulab and Amazon EC2 is shown in Figure 8, and response time comparison for read/write is shown in Figure 9. Throughput comparison for both browse only and read/write for both Emulab and Amazon EC2 is shown in Figure 10. In Figure 11 we compare throughput distribution for both read/write and browse only work loads for Amazon EC2 small instances and Amazon EC2 large instance. In Figure 12 we compare response time distribution for both read/write and browse only work loads for Amazon EC2 small instances and Amazon EC2 large instance. The configuration for Amazon Small instance and Large instances as shown below;

5 Discussion

In this section we discussed the main issues and obstacles encountered during the project and solution we found and used. Amazon EC2 is still not so matured and it still have some problems with management. We will not find those issues as significant
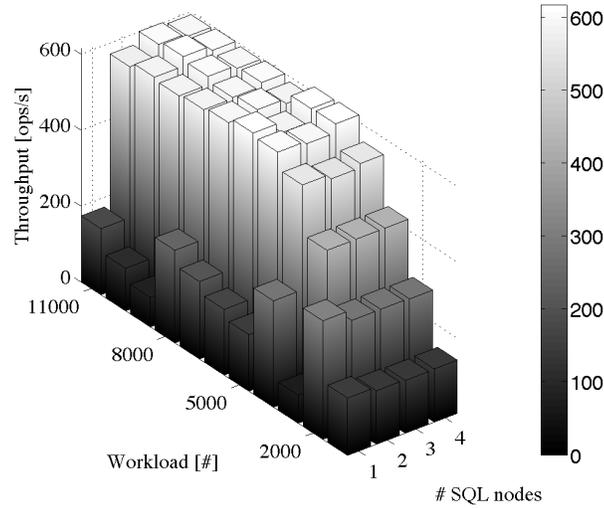
Fig. 5. Throughput for RUBBoS with two application servers, MySQL Cluster, and read/write workload.

Type	Components	
Small	Processor	Dual-Core AMD Opteron(tm) Processor 2218 HE 32-bit
	Memory	1.7 GB
	OS	2.6.21.7-2.fc8xen (32-bit)
Large	Processor	2 * Dual Core AMD Opteron(tm) Processor 270 64-bit
	Memory	7.5 GB
	OS	2.6.21.7-2.fc8xen (64-bit)

Table 2. Hardware node setup for Amazon EC2.

issues if we are using Amazon EC2 in manually, but when it comes to tool automations those management issues are really matters.

5.1   Issues with SCP and SSH on EC2

Amazon EC2 uses higher security mechanisms, as a result system automating has become a challenging task. Unfortunately even when we start two instances with our credential they can not communicate each other. In other words if we start two instances then we can not do SSH or SCP between those two instances. Unfortunately Elba really need to have support for SCP and SSH. Amazon does not provide any documentation for solving those problems, so finding a solution to this particular problem was indeed a time consuming task. However I was able to find a acceptable solution for that, and which consists of following steps;
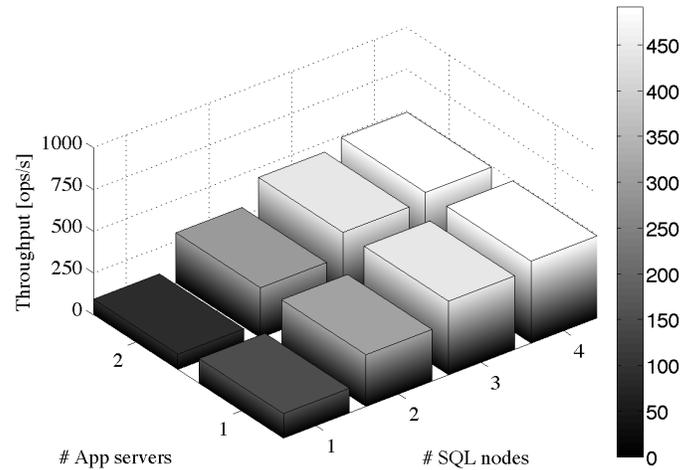
Fig. 6. Throughput for RUBBoS- Application servers and SQL nodes for browse-only workload.

- First create an Amazon Image
- Then generate a ssh key using ssh-keygen -t rsa
- Then SSH to some other public machine (not an EC2, a PC in CoC)
  - ssh b@B mkdir -p .ssh
- Then do cat .ssh/id_rsa.pub |ssh b@B 'cat >>.ssh/authorized_keys'
- Next go to b@B and try to do ssh to EC2 instance root@EC2-public-DNS you will not be able to do that, and you will get an error.
- Next open know_hots file, and copy the part with ec2*
- Next paste that into know_host in EC2 and duplicate the entry number of times
  - One for 127.0.0.1 key
  - localhost -key
  - ec2* -key
  - amazon.com -key
- Next re-create the Amazon EC2 image
- Run two instances using that image, then you will be able to do SSH and SCP without having any problem.

5.2 Issues with building Apache Web server in EC2

Apache is one of the commonly used Web server, and building that is not such a difficult task, and specially in Linux operating system. However building Apache Web server on Fedora Core 8 public images available in Amazon was troublesome work due to a number of reasons. In addition to that it takes considerable amount higher time compared to that of in Emulab. However finding the reason and analysis time break down is outside the scope of this paper.
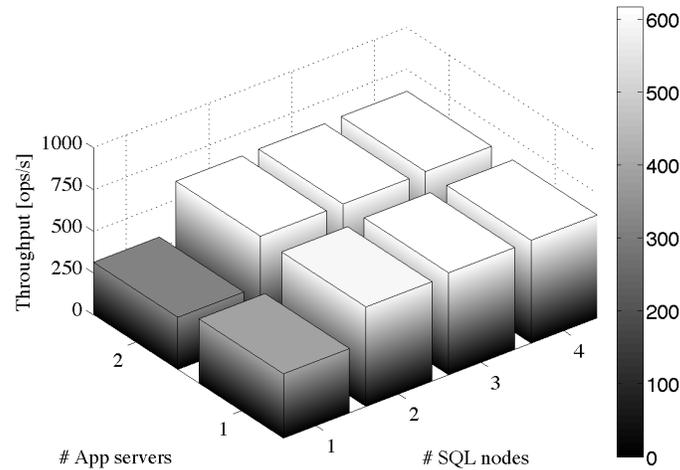
Fig. 7. Throughput for RUBBoS- Application servers and SQL nodes for read/write workload.

– First missing most of the essential libraries such as gcc, gc++
– Missing OS header files, the Fedora Core 8 public instance was missing a number of OS headers files. So I had to copy them from some other Fedora and re-create the image.

5.3   Issues with MySQL cluster

In the current implementation of Elba has support for Rubbis and RUBBoS benchmarks, in the first step of the project I only looked into moving RUBBoS benchmark into EC2. Before doing that I tried setting up MySQL cluster on Emulab. Setting up the cluster consists of following steps;

– Create a fresh instances of MySQL cluster with two data nodes and two MySQL servers
– Then change the original RUBBoS database scripts to use NDBCLUSTER database engine
– Then populate the cluster with original dataset
– Then create a data tar ball from data nodes

Original size of the dataset was 150 MB (archived) however after loading that to MySQL cluster the size of dataset became 3.5 GB. That is mainly due to in the case of MySQL cluster they store one row in one page. So larger the number of rows then larger the size of the data set.

Considering the cost, I first setup MySQL cluster in Emulab and performed a set of experiments, while doing those I encounter a number of issues. Those issues are described below in detail.
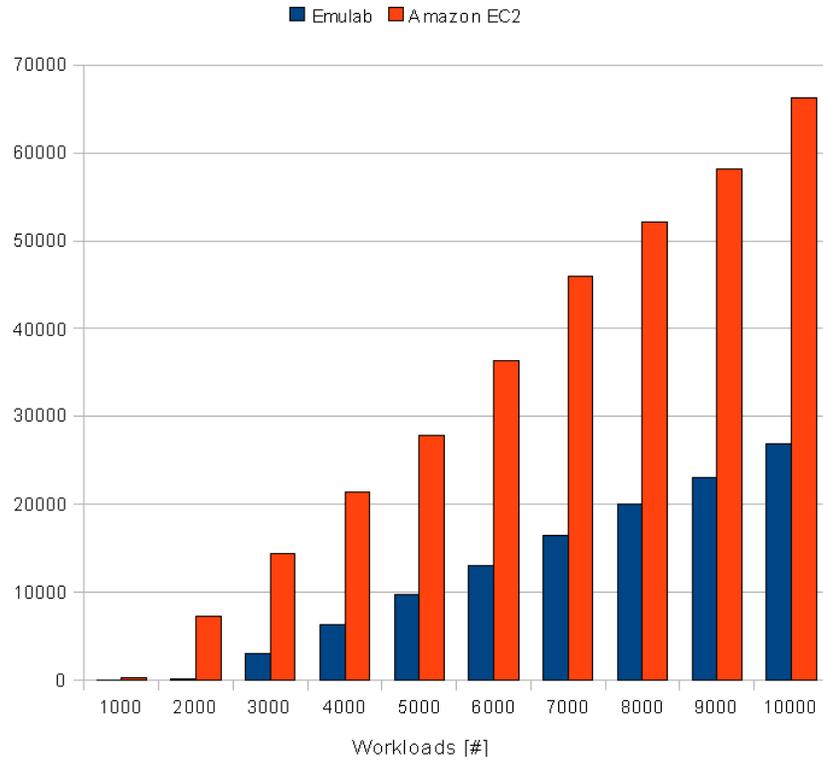
Fig. 8. Response Time comparsion for RUBBoS- Emulab vs Amazon EC2 for browse-only workload.

5.4 Low response time due to Database joints

First I set up MySQL cluster and used existing Elba code for RUBBoS and run the experiment with following configuration (with 1000, 2000,.... 13000 concurrent clients);

- Clients - 5
- Tomcat servers - 2
- Apache server - 1
- MySQL servers - 2
- Datanodes - 2

Then I compared the response time and throughput with the previous experiment with similar configuration, which was run against CDJBC and MYSQL. Then I found the response time is significantly different and MySQL cluster has very poor response time. Then I closely analyzed the RUBBoS source code to identify the reason behind that, then I found out that the low response time is mainly due to Elba was using modified version of RUBBoS. The modified codes heavily used Database joins and
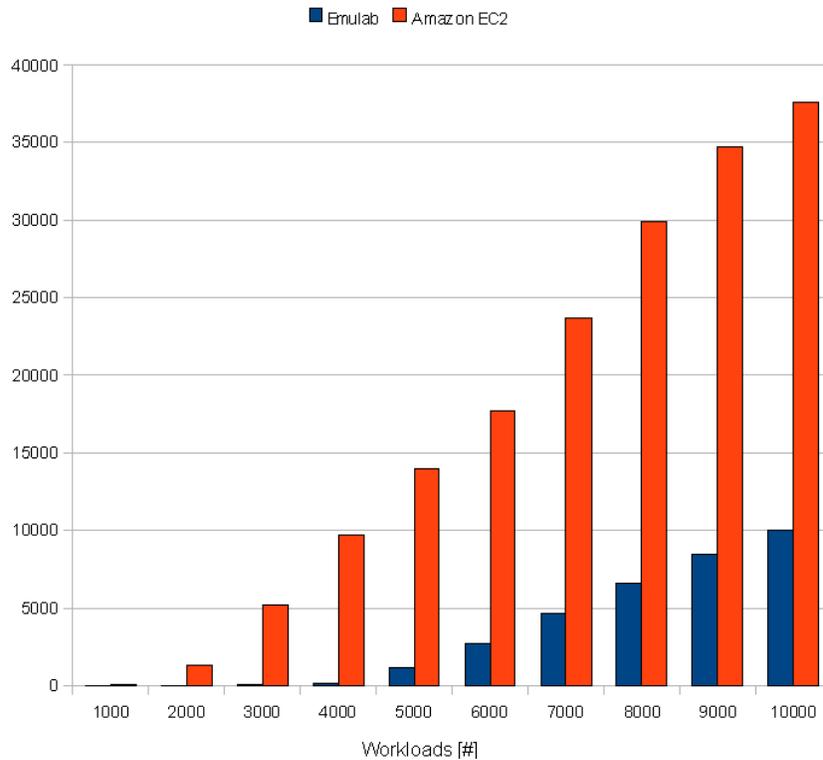
Fig. 9. Response Time comparsion for RUBBoS- Emulab vs Amazon EC2 for read/write workload.

which causes the performance problem in MySQL cluster. However those modified code works fine for other types of MySQL. After removing those database joints statements from the code I was able to get the acceptable results.

"Database joints are slow in MySQL cluster"

5.5 Memory leaks due to prepared statements

Next I replaced original RUBBoS code downloaded from RUBBoS web site and run the experiment, and while running the experiment I monitor the system status of Application server and observed that the memory usage going up unexpectedly. Then when it reaches the allocated heap limit Java garbage collection comes and consume all the CPU cycles, making CPU 99.9% utilized. As a result clients started to get connection timed out. After analyzing the code closely I found that is due to RUBBoS creates prepared statements inside a heavily used while loop. And they never resume them, so all of them get cached in the JDBC driver. As a result memory goes up.
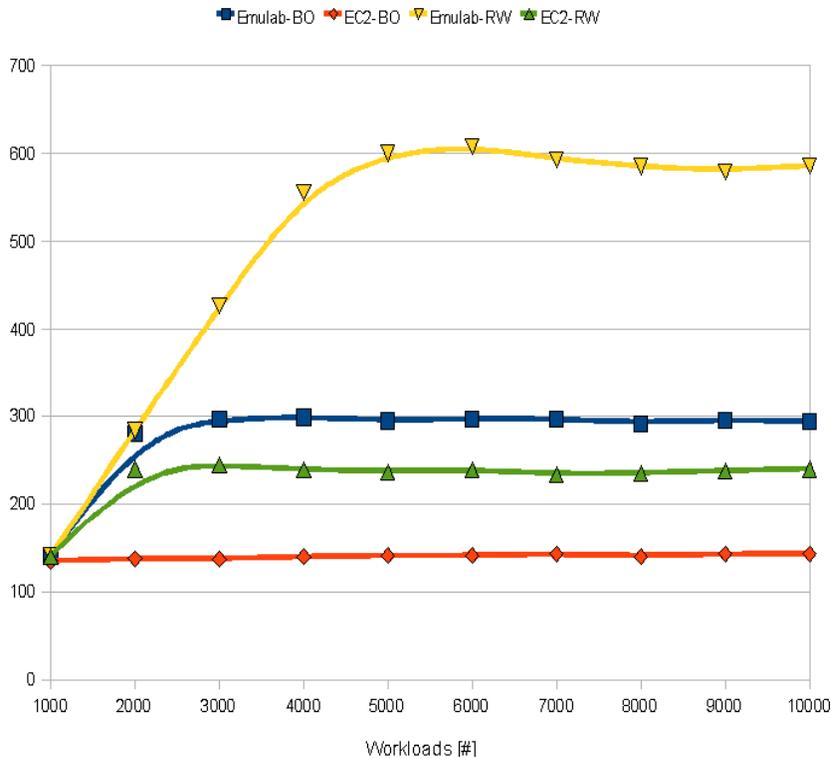
Fig. 10. Throughput comparsion for RUBBoS- Emulab vs Amazon EC2 for both browse-only and read/write workloads

"Do not use prepared statements unless you use them properly"

Java profiling with Jprofiler

Even after fixing the above mentioned memory issues for the higher workload I observed memory leaks in Application server. Then I did a detailed analysis of the code using Java profiling tools. Then I found that was mainly due to un-closed statements, there were a number of places they have created statements and never close them. After fixing all those it was possible to run the experiment and get some acceptable results.

6   Running Experiment on EC2

The first step of running experiment on EC2 is to create an EC2 account, and then create Amazon Machine Image(s). Then we can use following EC2 configuration files which was designed for Elba to start EC2 instances depending on the experiment setup (number of tomcat, number of mysql etc..) we are going to run. Once we have the
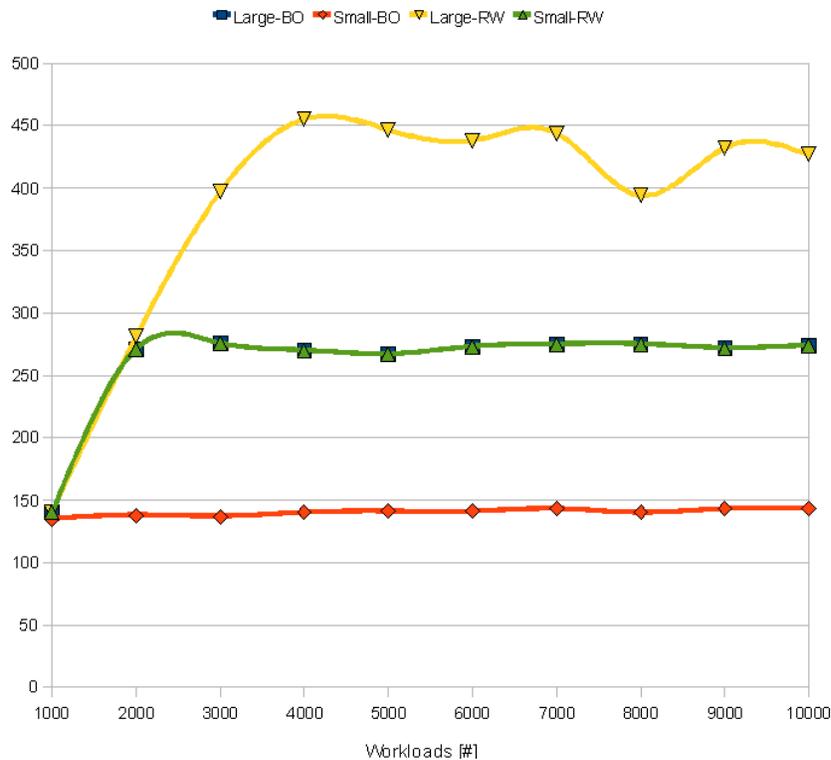
Fig. 11. Throughput comparsion for RUBBoS- Amazon EC2 (large) vs Amazon EC2 (small) for both browse-only and read/write workloads

configuration file in place we can call Axis2 Web service client to start instances. Then it will start instances and create a instance information file, which we can next use to generate the experiment setup description file.

Once we start the instance it will take about 10 to 15 minutes to come to run state, we have to wait till they come to running state to generate the test configuration file. For that we can use the AWS Management console. Once they start to run we can call Axis2 Web service client tool to generate the test configuration file. Once it is generated it we can upload that to control node (control node for the particular experiment we are going to run) of EC2 as follows.

"scp -i id_rsa-gsg-keypair ec2xtbl.xml
root@ec2-174-129-86-59.compute-1.amazonaws.com:/opt/rubbos/tools"

Now login to control node and generate the code, for that
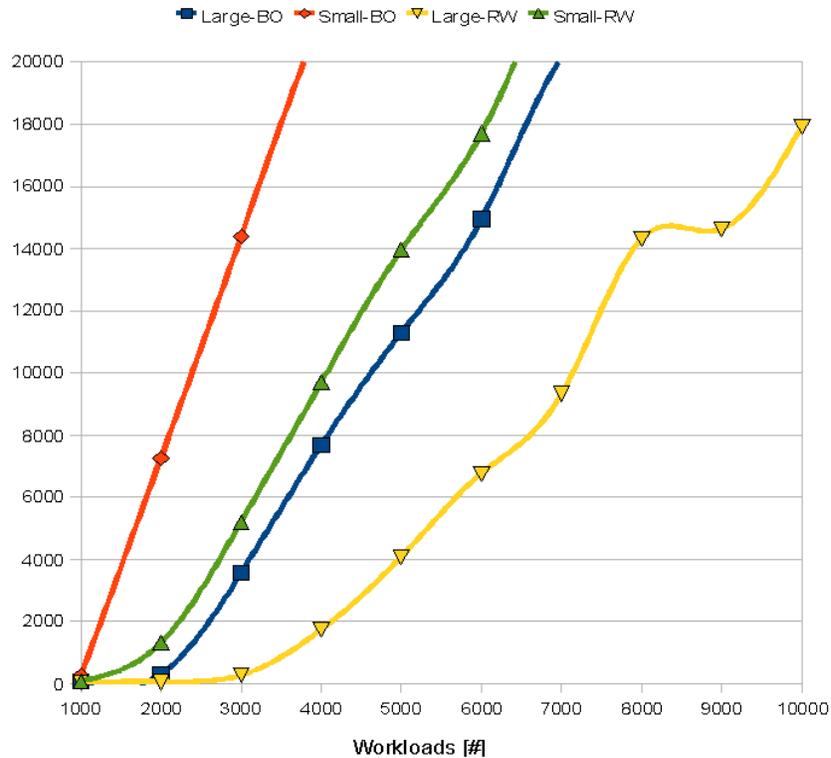
– go to /opt/rubbos/tools

Fig. 12. Response time comparsion for RUBBoS- Amazon EC2 (large) vs Amazon EC2 (small) for both browse-only and read/write workloads

- And run ./codegen
  - Then it will do the SVN update and start to generate the code
- Once it generates the code go to /opt/rubbos/output/scripts
- And run the experiment in background
  - ./run.sh >& run.log &

Then it will run the experiment and upload the results into the configured server once the experiennt is completed.

7 Future Work

Cloud computing is becoming the buzz word in today's IT industry, Amazon EC2 is only one of them. The next step of the project is to move Elba into other Cloud computing platforms including IBM Blue Could, Vmware Vcloud and Open Cirrus. In addition to that continuing the experiment on Amazon EC2 to find the reasons for low response time is also a challenging future work. In this project we have only focused on only one

type of MySQL cluster, and next immediate step of MySQL cluster is to use MYSQL cluster database partition technique and scale out the number of data nodes and do the same set of experiments.

8 Conclusion

We used techniques and tools developed for automated experiment management, including automatically generated scripts for running experiments and collecting measurement data, to study the RUBBoS N-tier application benchmark with read-intensive and read-write intensive workload. One of the major contribution is scaling staging testing into commercial Cloud and do a detailed comparison of response time and throughput for various work loads. In addition to that we have compared the throughput and response time between different types of Amazon EC2 instances.

In addition, we have discussed some of the issues and challenges encountered during the process, specially with security constraints of Amazon EC2, use of subversion as a technique to do the software evaluation management and writing a Web service client for EC2. Apart from Amazon EC2 we have also discussed about the problem we faced and the solution we have found for moving RUBBoS dataset into MySQL cluster.

9 References

1. RUBBoS: Bulletin board benchmark. http://jmob.objectweb.org/rubbos. html
2. Cecchet, E., Marguerite, J., Zwaenepoel, W.: C-jdbc: Flexible database clustering middleware. In: In Proceedings of the USENIX 2004 Annual Technical Conference. (2004) 918
3. Amazon EC2. http://aws.amazon.com/ec2/
4. MySQL Cluster. http://www.mysql.com/products/database/cluster
5. Elba. http://www.cc.gatech.edu/systems/projects/Elba/
6. MySQL manuals. http://dev.mysql.com/doc/#manual
7. Apache Axis2 . http://ws.apache.org/axis2
8. Subversion : http://subversion.tigris.org/
9. Elba SVN repository. http://svn.cc.gatech.edu/Elba/module/ec2