CS 441: ENGINEERING DISTRIBUTED OBJECTS FOR CLOUD COMPUTING

REPORT

ON

PROVISIONING STRATEGIES IN CLOUD

SUBMITTED BY:

SURYA SELVARAJ

UIN: 679003533

M.S COMPUTER SCIENCE

Table of Contents

EXECUTIVE SUMMARY

The applications deployed in the cloud environment are allocated appropriate resources based on the load for the application. The cost incurred is based on the usage of the resources. Hence it is very vital to thoroughly investigate the behavior of the application and form provisioning rules for the application that will enable the cloud provider increase and decrease the resources as per the given rules. The current project, a Java based web application by name JPetstore is aimed at understanding the bottlenecks and improving the performance of the application in the cloud environment by forming provisioning strategies.

OVERVIEW: PROJECT BACKGROUND AND SCOPE

The project that is performance tested in this project is called JPetstore. The application is built on Java based web technologies using maven build system. It is an e-commerce application where different pets can be bought which are organized according to their type.

The user has sign in if re-visiting the site or has to sign-up if visiting the site for the first time. A cart, similar to the ones provided in popular on line shopping sites is available, which enables the users to add the orders and continue shopping. Once the customer finishes adding all the necessary products to the cart, the application directs the customer to check out where the payment details and shipping address is filled out to complete the transaction.

The aim of the project is to find the provisioning strategies such that the application hosted in the cloud environment can scale both ways. The performance of the application should not take a hit and the resource usage should be optimal, as the owner of the application is charged based on the resources used.

The project uses the Jmeter performance and stress testing application to simulate the users to create the load and understand if the resources that are provisioned in the cloud environment meet the demand users. This is done in incremental steps so that a proper idea is obtained on the number of users that the hosted application can handle versus the performance of the application. The cloud environment used in the current project is Microsoft Azure and the application is deployed as a cloud service. The measure is taken based on the amount of CPU usage, the number of users, the throughput, response time and the error rate for the requested inputs from the server.

PERFORMANCE REQUIREMENTS

A good performance measure is selected such that the throughout of the application is high for the requests, the response time is low and the error rate is low. In order to provide this, the application server has to monitor the incoming traffic regularly and be flexible and agile in its resource allocation towards the application. The above terms can be understood to be following:

Throughput is the number of requests server per given unit of time. Better the computational power implies that more number of requests can be server resulting in better throughput.

Response time is the time for which the user has to wait before the request is served. This depends on the time taken due to the network latency, the time taken to serve the request by the server and the application architecture.

Error rate suggests the number of erroneous replies sent by the server to the user. This rate increases with the decrease in the availability of the server computational power.

PROJECT REQUIREMENTS

The project requires a computer installed with the Jmeter software for simulating the users and perform stress testing.

The application which is under consideration needs to be built and deployed on the cloud environment. The cloud environment chosen as part of this investigation is Microsoft Azure and the application is deployed as a cloud service.

BUILDING AND DEPLOYING THE APPLICATION

The application is hosted on Github for public usage. It can be obtained by choosing either to download the application or clone the application to the local machine. The method chosen in the present experiment is by downloading the application to the local machine instead of cloning.

The application is based on the maven dependency build procedure, where the dependencies on the project can be added to a POM file and they are added in the project directory.

For easing the build process further, the current project uses Netbeans IDE to build the project. The Netbeans based build requires very less effort and prior knowledge. The process is documented as a video in the recordings folder in the project folder.

TESTING THE APPLICATION IN THE LOCAL DEPLOYMENT

The application used for testing is Jmeter, which helps in creating stress tests by imitating user like behavior. This step will be helpful understanding the application before deploying it in the cloud environment. This will also enable the tester to gain a fair amount of understanding on how the server might respond and how many users can be handled by a single deployment.

DEPLOYING THE APPLICATION IN AZURE CLOUD ENVIRONMENT

The azure cloud environment provides multiple options to deploy the application in the cloud. Deployment as a cloud service and web service are the widely used options. The deployment as the cloud service provides managed services where the user can easily deploy it and the underlying work is done by the service provider. The web service mode of deployment requires the user to create virtual machines, install all the required software for the application to run and then deploy the application on the server. This mode enables customization options on the application and the running environment.

RATIONALE FOR CLOUD SERVICE AS MODE OF DEPLOYMENT

The method chosen in this experiment is the azure cloud service option. This option is chosen as the application is light weight and the database used is Hyperbase, a database tailored for Java eco-system. This is also a light weight application and needs very less customization.

As the system level tweaking required is very less cloud service is chosen. The extra mileage of this method is in the form of managed services meaning less time to spend on deployment of the application deployment and use focus more on performance of the application.

DEPLOYING APPLICATION IN AZURE CLOUD AS CLOUD SERVICE

The quickest and easiest way to deploy a cloud service is by using the eclipse plug-in for azure cloud. The azure plug-in can be installed in the eclipse IDE using the following resource.

WindowsAzurePlugin4EJ - http://dl.msopentech.com/eclipse

After installing this plug-in, right click on the project folder, select the azure option and click on publish to Azure. After providing the subscription details the and selecting the configuration for deployment, the application will be packaged and deployed in the cloud.

PERFORMANCE TESTING APPROACH

Using the Jmeter application each time, a transaction is recorded using the Recorder available in Jmeter application. Using each of the transaction, multiple users are scheduled in the application as threads in a thread group for load testing. Based on the allocated number of users, the throughput, error rate is measured and CPU usage percent is monitored in the Azure cloud metrics.

The following are the experiments performed iteratively to arrive at point where the bottleneck in the application can be observed and the provisioning metrics are clear.

TEST EXECUTION

The testing performed can be split into three phases:

Phase 1:

Testing using heavy load and fetching only a single page by all the users. This approach did not place the CPU under heavy usage.

Phase 2:

Testing using heavy load and all the users requesting for pages in an order as listed in a transaction. This approached placed server CPU under heavy load. Auto scaling and alerting also came into picture.

Phase 3:

Testing is performed using heavy load on the CPU with users placing load in regular intervals using each set of users placing request for different set of transactions. Here the bottle neck of the application, the auto scaling provisioning strategies and alerting become more prominent.

PHASE #1

As per initial understanding the cloud environment in which the application is hosted has to be stress tested by simulating the users who might use the application in real time.

Test parameters:

Number of users: 150

Number of times looped: 250

Test script used: Thread Group_jpetstore2.jmx, Thread Group_jpetstore15000.jmx

Transaction performed: Fetch a single page.

In this test only one web page is demanded by all the users and the resulting graph for CPU usage can be seen from the below snapshot.

Observations after test:

The CPU usage is reaching less than half of the total capacity and it is dipping down even as the test is continued using the same number of users.

After performing the experiment several times, the results are almost the same as per the number of threads used in Jmeter application and number of times the loop has been executed.

The above experiment has been performed using 150 threads and looping it for 250 times.

With almost the similar load the test yielded the same behavior of results.

Understanding Gained:

- The above tests gave an understanding that merely using huge number of users with same transaction for every user may not place a heavy load on the CPU as caching mechanism may come into affect. Also if the size of page to be fetched is small then it is easier to cache the page for repeated requests.
- The error rate is almost constant and negligible and throughput deteriorates linearly as the number of users are increasing and it is not alarming.

PHASE #2

As per the understanding gained in the phase 1 of experimentation different transactions are needed to build considerable load on the CPU in the cloud environment.

Scripts used:

- firstFunnyScript.jmx
- autoscaled_measuring.jmx
- MultipleTransactionsFunnyScript.jmx
- MultipleTransactionsFunnyScript2.jmx

Number of transactions: On an average 60-70 transactions have been used in the phase 2 of experimentation for each thread/user.

Number of users: 1000

Number of loops: 2

It was observed that during the initial stages the throughput is high and the error rate is zero for almost all the application and database oriented transactions. The error rate is 100% for all redirections. The throughput can be counted as a healthy.

The throughput is down as the time passes and the number of users are increased.

There is an increase in the CPU usage and when the CPU utilization crosses the range of 50, the error rate starts increasing.

In order to deal with this, auto scaling has been implemented with increment in the instance of the CPU when the CPU usage is in the range of 40% to 60% for a period of over 5 minutes.

This results in spawning of new CPU as a load persisted continuously for 5 minutes.

Observations from the present phase of experiments:

- All the newly spawned instances are created from the initial instance that is running
- The process of creating new instances takes some time, approx. 5 minutes before all the instances are functional and working.
- Even though the new instances are created the throughput and the error rate are not improving meaning the load balancing for the previously queued requests are not distributed among the newly created instances.
- Therefore, it can be concluded that the increase in the new instances must be done before hitting the bottleneck condition as the time taken to instantiate a new machine is taking some time and decreases the output

PHASE #3

In order to increase the effectiveness of the experiment and the outcomes from them, the load is presented to the server in multiple iterations and checked if the throughput is increasing for different set of requests without decreasing the initial load.

Test scripts used:

- dec9_2015_9532.jmx

- dec9_2015_1156.jmx

Observations from phase 3:

These scripts are run in such a way that the second script is started after the initial CPU instance starts giving erroneous results and a new VM is created. This can help in understanding if the throughput is stagnant or increasing as the time passes.

OBSERVATIONS AND RATIONALE:

1) During the transition, the time when the new VM's are being started and made to run, the throughput in the application is dropping very drastically hinting that there may be a non-availability in the resources. Not sure of the reason for this behavior.
2) It is taking while for the transition to take place, the process of starting the new VM and getting the newly added resource to run and start serving.
3) The replication is not seamless and it is taking some time and output is dropping.

4) The CPU usage during the process is going very low in spite of many threads waiting for completion in Jmeter. This may occur as the processors/VM's are not responding.

The reasons for low throughput in spite of increasing the VM's can be thus:

- The original VM's queue is filled with requests from the previous requests and no new requests are being made. Also need to observe that the worker role that is initially being used is used during transition for purposes of crating new VMs and other activities.
- Also it could be that the DB is already having bottleneck due to which the entire application is not responding.
- The load balancer may not be active or is not configured properly. If the initial queue for worker role is over crowded but there is no way to transfer that queue to the new VM's.