IN4392: Cloud Computing, Lab Report

A. Kučera*, J. van Touw†, A. Iosup‡, D.H.J. Epema§ and B.I. Ghit¶
Parallel and Distributed Systems Group,
Technical University of Delft
Email: *A.Kucera@student.tudelft.nl, †J.vanTouw@tudelft.nl, ‡A.Iosup@tudelft.nl,
§D.H.J.Epema@tudelft.nl, ¶B.I.Ghit@tudelft.nl

Abstract—In this report we evaluated the use of t2.micro instances leased from Amazon.

1 Introduction

1.1 Current Situation

WantCloud BV wants to develop a new image processing client/server application, which would allow users to upload set of images and receive them back resized to 4 different sizes. Such an application might be useful e.g. for web galleries, which requires to have images in different sizes to show image previews etc. WantCloud BV wants to try the cloud computing approach for this application to allow for scaling of the application.

1.2 Related Work

Amazon EC2 t2.micro instances were chosen to host the desired image processing application. The application itself is developed in JAVA and is using several external libraries to support its functions. The JCommander library was used to allow users to specify custom parameters for resized images. The Sigar library was used to collect performance data from the machines. The JSch library was used for SSH communication between Amazon instances. The ZIP Directory library was used for packing resized images into one ZIP file.

1.3 Proposed solution

The application implemented in this report is leveraging Amazon cloud services to provision several users. One Amazon micro instance represents a master server, to which clients can connect to. The master server is responsible for aiming the clients to another Amazon micro instances (called slaves), which process required images. The application can use several slave machines, depending on the clients load.

1.4 Report overview

The report is structured as follows, first we will go into the background of the application with its requirements in Section 2, after which the actual system design is explained in Section 3. This is followed by Section 4 which describes the different experiments run. Next, a discussion about what we think should be done is in Section **??**. Finally, we conclude in Section 6.

2 Background on application

The application receives the ZIP file with images to be resized and also custom user parameters, if specified. Images in JPG and PNG format are supported. The application then extracts the images and resizes them to 4 different sizes. These resized images are then put into another ZIP file and sent back to the user. The image processing itself is performed on slave virtual machines, which are chosen by master machine according to their CPU load.

2.1 Application requirements

WantCloud requires the application to fulfill basic reqirements laid on all cloud computing

applications. In particular, it means that application should satisfy these requirements:

1) System should automatically decide whether it should start more Virtual Machines and the incoming jobs should be automatically distributed among these machines.
2) System should offer such functionalities to start and stop rented machines.
3) System should allocate new jobs to least utilized machines.
4) System should offer policies for sudden system failures.
5) System should keep usage statistics and performance metrics.

3 System Design

3.1 Overview

The application is designed as a typical client/server application. Client can specify in CLI[1] parameters which file to upload and what sizes of images to request. The client first contacts the master server, which will send the user the address of a slave machine, which will process the job. The client then uploads the file to the slave machine, where it is processed and sent back to the client.

Master instance of the server consists of these main parts:

1) *Listener*. Listens to clients through TCP socket, when a client connects, it asks VM Manager for the least utilized slave Virtual Machine and returns its address to the client.
2) *VM Manager*. This component is responsible for starting and stopping slave instances. When a utilization of a machine is bellow certain treshold, this machine is stopped. On the other hand, when a machine is highly utilized and other machine is available, it is started.
3) *Monitor*. Monitor communicates via TCP socket with slave machines and updates information about their performance, so their utilization can be easily resolved.

Slave instances of the server are composed of these main parts:

1) *Connection Handler*. This component is responsible for accepting ZIP file from the clients and sending ZIP files with resized images back to them.
2) *Job Processor*. Main component responsible for image processing itself. It unpacks the ZIP file, resizes images to all 4 different sizes and creates new ZIP from them.
3) *Monitor*. Monitor collects the information about usage of the machine and its performance. It also generates log reports and sends these information to master Monitor.

You can see the design of the application on figure TODO.

3.2 Resource Management Architecture

In this section we will describe several system features to give a better insight into the whole resource management architecture.

3.2.1 Master Virtual Machine

Master Virtual Machine is responsible for managing slave Virtual Machines and pointing incoming client to these machines. The master server is running three main threads:

- Listener thread, which is responsible for accepting clients and pointing them to the right slave machine.
- VM Manager thread, which constantly leverages Monitor to watch over the state of Virtual Machines and which decides whether they should be started, stopped or kept running.
- Monitor thread, which receives information about performance from the slave machines and updates them in VM Manager.

The address of the master server is the only address, which the client requires to know. Addresses of slave machines responsible for image processing itself are dynamically sent to the client.

3.2.2 Slave Virtual Machines

Slave Virtual Machines are performing the image processing itself. After the client receives the address of a slave machine, it starts uploading ZIP file with images to process. After

1. Command line

the upload, the ZIP file is given to the queue, from where the Job Processor starts resizing the images. There is a single thread for each job and therefore a slave machine can process multiple jobs simultaneously. However, a maximum number of jobs can be set. After the resizing, the images are put into new ZIP file, which is sent back via socket connection to the user.

Each slave machine is running three main threads:

- Connection thread, which receives the ZIP file from the client, puts it to the queue and after the completion sends the ZIP file back.
- Job Processor thread, which dequeues the jobs from the queue and processes them.
- Monitor thread, which monitors system state and performance every second and regularly sends these information to master server. The thread is also responsible for logs generation.

3.2.3 Summary

This resource management architecture, as mentioned above, suffices all requirements laid on the system:

1) System is fully automated, the client only needs to specify file to be processed. System will automatically decide on which machine the job should be processed and it will start and stop virtual machines to achieve more efficient utilization.
2) System is able to start and stop Amazon micro machines, according to the given policies.
3) System is able to schedule client on different slave machines, according to the given policies.
4) In case of a failure of a machine a new machine will be started and the job can be restarted on this machine.
5) System keeps track of its usage (number of users) and performance (CPU and Memory utilization) in the logs.

3.3 System policies

3.3.1 Allocation policies

The job is allocated to the machine with lowest CPU utilization at the current moment. However, the system offers methods accessing other usage and performance metrics, which allows easy modification of this policy in the future.

3.3.2 Provisioning policies

New machines are provisioned when the normalized load is above a certain threshold. This normalized load is calculated by taking the CPU load of every machine and dividing it by the number of machines, it does the same for the memory load. Both loads are measured in percentage, in the range of 0-100. If one of the loads is above 75%, the system will provision one new machine. Otherwise if both loads are under 35%, it will send a signal to stop the machine with the lowest load. It will first let it finish its current task if it has one and then actually stop the machine.

3.4 Implementation

The system is implemented in Java and it leverages the following libraries:

- JCommander for easier parsing of parameters.
- Jsch for establishing SSH connection between master and slave machines.
- Sigar for getting the access to performance data.
- AWS API, to manage instances from Amazon Elastic Cloud Computing
- ZIP Directory for creating new ZIP from a folder.

4 Experiments

4.1 Set-up

All of our experiment were run on Amazon EC2 t2.micro machines, these machines have only 1 CPU and 1GB of memory. We had rented 5 of them and had used them for all our experiments. However, it is very easy to extend our application to support unlimited number of Virtual Machines. We were running Amazon Linux with Java installation on all of our instances.

4.2 Experiment analysis

4.2.1 Automation

4.2.2 Elasticity

without provisioning vs with, show that it isn't much slower than just having the jobs all on

4.2.3 Load balancing

(job max on a slave) turn off load balancing vs on

4.2.4 Reliability

kill job, show that user can submit again

4.2.5 Monitoring

4.2.6 MultiTenancy

run x clients with the same job, show that variance isn't too high

5 Discussion

??

6 Conclusion

Appendix

.1 Time Sheet

	Adam	Jaap
the total-time	0	0
the think-time	0	0
the dev-time	0	0
the xp-time	0	0
the analysis-time	0	0
the write-time	0	0
the wasted-time	0	0