Report on checkpoint delivery of Group 17's CCV project

Diogo Vilela, Francisco Barros, Rafael Ribeiro
Instituto Superior Tecnico - ULisboa
Lisbon, Portugal
{diogo.vilela, francisco.t.barros, rafael.m.lucas.ribeiro}@tecnico.ulisboa.pt

Abstract

In this paper we describe the work done for the checkpoint delivery of the Cloud Computing and Virtualization project. We also describe the steps to be done for the final delivery.

1. Introduction

In this year's iteration of the course, the project, named HillClimbing@Cloud, is to design and develop an elastic cluster of web servers to find the maximum value on simplified height-maps using a set of search/exploration algorithms. The functionality of the system serves as a demonstrator of CPU-intensive processing in order to be able to easily use the mainly evaluated parts of the project: instrumentation done over the search algorithms code, load balancer and auto-scaler algorithms and metric storage system design.

The developed system is running within the Amazon Web Services (AWS) ecosystem, and currently using AWS's own load balancer and auto-scaler, but configured by the group, so the system is able to work properly in order to respond to the amount of requests received at any point in time.

2. Work done

The following subsections describe what was done for the checkpoint delivery of the project.

2.1. Refactorization of the given initial code

To avoid students spending time working on non-evaluated parts of the system, the professors made available the code for a web server that handles the requests the system will receive.

Although the available code worked as it should, it was refactored so it would be more modular and ease the development of the intended solution. During the refactorization the modules that would be necessary for the checkpoint delivery were created.

After refactoring, the project had the following modules:

- WebServer, comprising of the code responsible to receive the request, parse the arguments, pass it to the classes in HillClimbing module so it computes the response to the request and finally sending the response to the client.
- HillClimbing, comprising of the code responsible to compute the solution to the problem passed by the WebServer module. It has also the code for generating height-maps.
- Instrumentation, comprising of the code responsible to create instrumented classes and holding instrumentation in-runtime values so it can, currently, be saved into the file system in the AWS instance running the WebServer.
- Database, comprising of the code that defines the information that will be stored in the metric storage system.
- Utils, comprising of the code that is used by the other modules to perform auxiliary functions.

2.2. WebServer

In order for the load balancer test the liveliness of the web server, it was added a new *HttpHandler* to the web server on the page */ping*. Also, in order to collect and store the information needed to the metric storage system, before starting the solver, it uses the Instrumentation module to store the parameters of the request associated with the running thread and after the solution is found, the instrumentation values are stored in memory alongside the parameters of the request, the web server uses a method present in the Instrumentation module to store that information to disk.

2.3. HillClimbing

No changes were done to this module.

2.4. Instrumentation

So that in the final delivery, the load balancer and the auto-scaler have valuable data about the computational cost of each request, a BIT tool was developed which instruments the HillClimbing solver code so it gathers the amount of load type instructions in each run of the solving algorithms. To support the storage of said information, it was also developed a class that holds the two maps between each thread's ID and request parameters, represented with the database entry template, and each thread's ID and load type instructions count. That class is also responsible to store the information to the disk.

2.5. Database

In order to have a system wide representation of the instrumentation data a class that holds the parameters of the request and the load type instructions count was designed. This class was implemented already thinking about the way it would be stored in the metric storage system of the project's final delivery and so, it is also a valid entry to a Amazon DynamoDB table that will exist in the final delivery.

2.6. Utils

No changes were done to this module.

3. Work to be done

The following subsections describes the work to be done for the final delivery of the project.

3.1. Instrumentation

In order for the information gathered to be the most valuable, for the next delivery it will be investigated what type of instruction count is the one that most accurately represents the cost of a request. This will be done, probably, by testing all instructions types and comparing it with the actual of the request.

3.2. Database

In order for the instrumentation data be stored in a central database instead of the file system of each AWS instance, it will be created an Amazon DynamoDB table with the entry template already defined in the checkpoint delivery and a connector will be developed, in order to write to and read from it.

3.3. Load balancer and auto-scale algorithms

Finally, the load balancer and auto-scale algorithms will be implemented. These will use the information stored in the metric storage system, so it can perform better compared to the ones configured in the AWS Console.