CNV Project: Final Report - Group 7

Diogo Neves	Filipe Silva	João Moniz
95554	95585	83480

17 June 2023

1 Introduction

For the final delivery, our objective was to develop an auto scaler and a load balancer, both in the Java programming language and using the Amazon SDK, that could do a work similar to the auto scaler and load balancer encountered in the Amazon Web Services (AWS).

With this objective in mind, we were given a workload (EcoWork@Cloud, in the *webserver* folder) that created a webserver with three scenarios:

- Image Compression
- Foxes and Rabbits
- Insect Wars

Our objective was, then, to develop these applications, alongside scripts to create the images programmatically and deploy them in AWS EC2.

We will then, in this report, show our design decisions for each of the following modules of our work (in the git repo, each one is a folder located in the *src* folder):

- javassist
- lbas
- scripts
- webserver

2 Design

2.1 Javassist

The Javassist module consists of two main parts: the *ICount* tool, modified to report the instruction count per thread (because the workload is multithreaded); and the *AmazonDynamoDBConnector* class, which consists of a connector to the Amazon DynamoDB, a key-value store, in order to keep certain information about the program we will run it with, which we will discuss later.

This module is then ran with the webserver, and allows to know how many instructions were ran for a specific scenario of it, by intercepting each one of the functions that correspond to a scenario:

- *Image Compression* - *process* function, with the following arguments: image, target format and compression quality.
- *Foxes and Rabbits* - we will keep a record of number of instructions per generation, per world (the generation doesn't matter much in this case);
- *Insect Wars* - we will keep a record of number of instructions per round, per number of insects (total) and ratio between the numbers of each army.

2.2 LBAS

2.3 Scripts

2.4 Webserver

For the intermediate delivery, we have implemented the following features:

- Instrumentation of the EcoWork@Cloud workload using a JavassistAgent: in this case we modified the ICount JavassistAgent to be mindful of the multithreaded nature of the workload, saving (locally, for now) the instruction count of every thread in a map;
- Scripts to create the image and deploy/terminate deployment in AWS, along with load balancer and auto scaler provided by AWS.

For the final delivery, the following goals apply:

- Update the (already) modified ICount JavassistAgent to send its statistics to DynamoDB (just periodically, to keep performance), so that multiple runs can be better distributed across nodes;
- Implement a load balancer in Java that uses the information gained from ICount (retrieved from DynamoDB) to distribute the load across nodes;
- Implement an auto scaler in Java that uses the CPU utilizations of the various nodes to determine if a node needs creating/destroying.

Regarding the load balancer, our goal is to distribute the load as evenly as possible across all nodes, in order to keep the number of nodes spawned at a minimum. It can also choose to serve a request using a Lambda instead of a nodes: this option is more expensive, however it makes up for it for the speed in startup. Especially when the system is under load and more requests are being made while the LB spawns more nodes, Lambdas will be a good option to support users while the system is still trying to scale up.

For the metrics used by the load balancer, we will keep in mind the number of instructions executed per request, with different granularity based on the workload that is provided:

- *Image Compression* - there are 3 image formats: PNG, JPEG and BMP. We will keep a record of the number of instructions per image size, per compression factor and format;
- *Foxes and Rabbits* - we will keep a record of number of instructions per generation, per world (the generation doesn't matter much in this case);
- *Insect Wars* - we will keep a record of number of instructions per round, per number of insects (total) and ratio between the numbers of each army.

Regarding the auto scaler, the metric we will keep in mind is the average CPU utilization of each node: if the system detects that more requests are being made and the CPU utilization is above a certain threshold, it will spawn a new node. Likewise, if the CPU utilization is below a certain threshold, it will destroy nodes, in order to keep the cost of running as low as possible. The algorithm will scale up by creating new nodes, and scale down by marking the nodes it wants to destroy for termination, however they can only be terminated once there are no more requests being served by them.