Soduko Cloud
Project Checkpoint 1
Cloud Computing and Virtualization

Gonçalo R. A. Faria
Instituto Superior Técnico
University of Lisbon
Lisbon, Portugal
goncalorafaria@tecnico.ulisboa.pt

Abstract

This report details the design and development of an elastic cluster of web servers that is able to execute a computationally-intensive task to discover the solution of a sudoku puzzle on-demand.

1. Introduction

The goal of this report is to detail the design and development of an elastic cluster of web servers that can execute a computationally-intensive task to discover the solution of a sudoku puzzle on-demand. The system was designed to receive a stream of web requests from clients. Each request contains a description of a sudoku board as well as the name of one of a predefined list of available solution methods. In the end, the solved soduko boards are sent back to each of the clients.

The project was proposed by the faculty of **Cloud Computing and Virtualization** in a statement delivered to students. Accompaning these project's statement, there was an implementation of a single web server in java byte code and an HTML client.

To implement the functionality required for an elastic cluster of web servers, given that I was suposed to use the provided byte code, the original byte code was instrumented using the BIT library(0).

The main functionality implemented by instrumentation was data collection. The continously collected data, serves as the ground truth for the load balancing and resource allocation policies.

The system was designed to run within the Amazon Web Services ecosystem.

The first checkpoint comprises the software infrastructure for the system and a detailed exploration of what metrics are more usefull for the load balancing and resource allocations policies while keeping the ovearhead low.

2. System Architecture

Figure 1 contains a high-level illustrative description, as delivered by the faculty, of the target system architecture. The client interacts with the system by submitting an HTTP request to the load balancer. The load balancer decides which virtual machine will serve the request.

Figure 2 contains simplified class diagram of the project's code base.

2.1. Load Balancers

Since hypothetically, an extremely large number of web-servers could be started to take care of an increasing demand

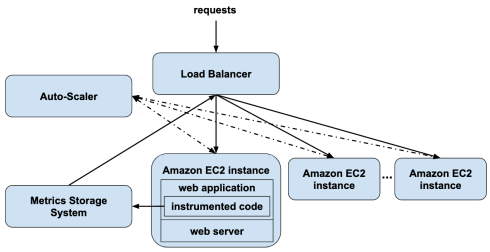

Figure 1: Diagram of the soduko cloud architecture.d

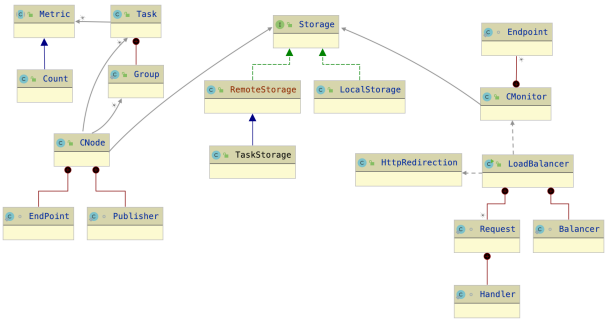

Figure 2: Classe diagram of the codebase of this project.

of requests, I can conclude that the load balancer will be the bottleneck of the entire system.

For this reason, the main design decisions behind the load balancing implementation were motivated to free resources in this crucial component. Hopefully, these freed resources will prove useful when in later phases it is required computational time for the more sophisticated decision making policies and data structures.

Therefore, I decided to use HTTP redirection as the mechanism to avoid directly answering the client from the load balancer. It is important to note that, I refer to temporary HTTP redirection. This means that subsequent requests have to go again through the load balancer. This does not provide client affinity, which is not a particularly useful property for soduko solving. One of its clear costs is the additional round trip time for redoing the HTTP POST to the web servers.

With this approach, rather than keeping 2 connections in the load balancer per request, one for the client and other for im-

itating the client for the selected web server, I'll have neither. The only work in the load balancer is receiving the request, putting it in a queue, and then a 'balancer'(server worker) decides where to re-route the request. In this way, after the re-routing, every responsibility regarding the request is transferred to the web servers. What is left is one TCP connection for each of the web servers. These connections are the medium for continuously maintaining a representation of each of the instances inside the load balancer. This state representation will have valuable information that will be used for the autoscaling and load balancing in conjunction with the metric storage system(queue size, expected time to complete, etc).

2.2. Auto Scaler

The functionality for the autoscalar is already implemented, however, the particular policy implemented in this phase is particularly simple. Essentially, if the number of requests in the entire system is greater than 3 times the number of instances(including the ones starting) then a new instance is created. When the number of requests in the entire system plus one is lower than the number of instances(including the ones starting) then the virtual machine with the lowest load is scheduled for recall.

2.3. Web Server and Code instrumentation

The Webserver was intrumented for two porposes. Firstly, metric collection, to allow for counting the number of branches taken and other metrics that will not be used for the final appllication but that were crucial to study the computational caracteristics of requests with different parameters(particularly because the source code was not given) and secondly to keep a connection with the load balancer.
The first one proved to be challenging, given that to obtain metrics associated with a particular request the code has to intercept when a new thread is created, which parameters the request has and to start a new metric counter for that specific thread.
For this porpose, I extended the functionality of the high-BIT library to allow for the execution of a desired function before or after the execution of particular routine with the arguments of the routine. In this way, it is possible to acess values of variables contained in the running application.
This was used to execute code before the argument parser so as to associate the thread to specific request parameters.

2.4. Persistent Storage

The persistent storage for saving the performance metrics of the request was a table using the Amazon DynamoDB.

3. Metric Selection

3.1. Predictive capacity

To determine what types of metrics would be valuable for determining each request's load(which I considered to be the number of instructions) I tested a set of metrics. Particularly, I experimented with the number of 'basic blocks executed', 'methods invoked', 'branches taken' and 'increments performed'.
Figure 3 contains 4 scatter plots that depict how the number of instructions varies as a function of the metrics for

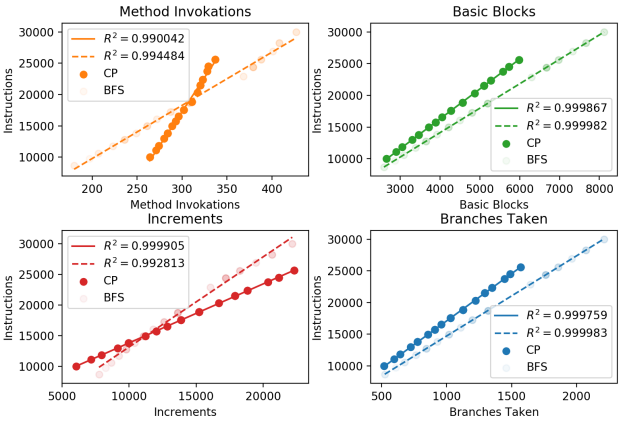

Figure 3: Experiments maid using the provided client for 9x9 soduko board with CP and BFS solvers.

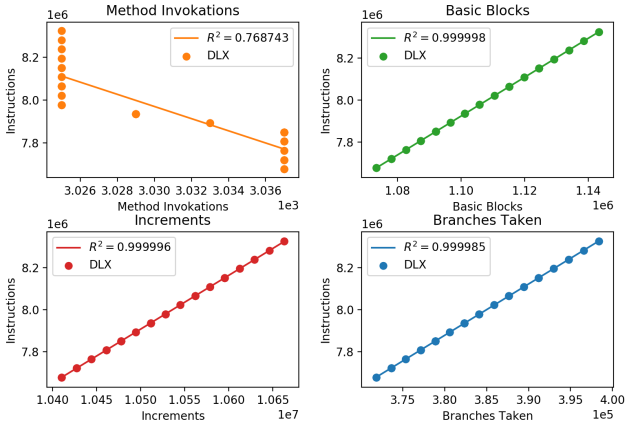

Figure 4: Experiments maid using the provided client for 9x9 soduko board with DLX solvers.

the CP(Constrained Programming) and BFS(Breath-First) solving methods. The data pertains to a 9x9 soduko map, for multiple values of unsigned entries.
As can be seen, all of the metrics correlate extremely well with the number of instructions for the tested solvers. It can also be seen that the slope for the regression line is different for different solvers.
Figure 4 contains the same experiment now performed with the DLX solver, under the same conditions.
Unlike, when using CP or BFS, the metric number of methods invoked, does not correlate well with the number of instructions.

3.2. Metric Overhead

Given that almost all of the tested metrics predict reasonably well the number of instructions to decide which to use for the load balancing I decided to measure the number of additional instructions required for collecting each of them. Figure 5, 6 and 7 present an exploration of the metric overhead for the same 9x9 soduko board for the CP, BFS and DLX, respectively.
What I can conclude is that 'methods invoked' causes the lowest amount of overhead, followed by 'branches taken'. If this metric also correlated well with DLX, it would be the best choice. Since this is not the case, I decided to use

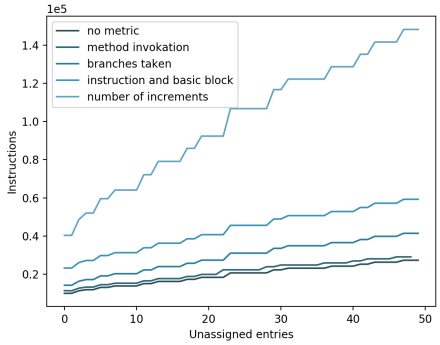

Figure 5: Overhead results from solving multiple configurations of 9x9 soduko board with the CP solver.

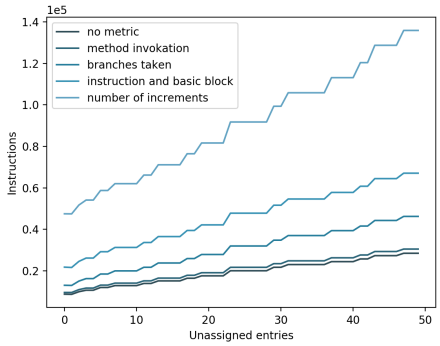

Figure 6: Overhead results from solving multiple configurations of 9x9 soduko board with the BFS solver.

'branches taken'.

4. Discussion

It is clear that for every solver 'branches taken' correlates almost perfectly with instruction number. However, for different solvers, the slope of the linear predictor is different. This means that increasing the "number of branches taken" does not lead to the same increase in the number of instructions. This suggests that, by itself, the number of branches taken can not be used as the criterion for comparing loads of different solvers. To this end, we need a linear predictor

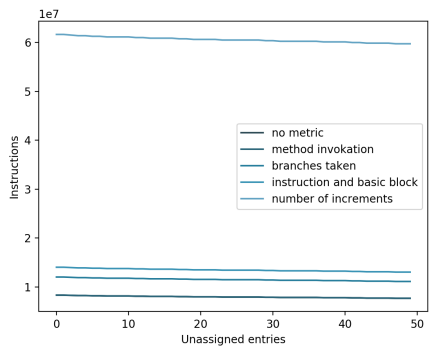

Figure 7: Overhead results from solving multiple configurations of 9x9 soduko board with the DLX solver.

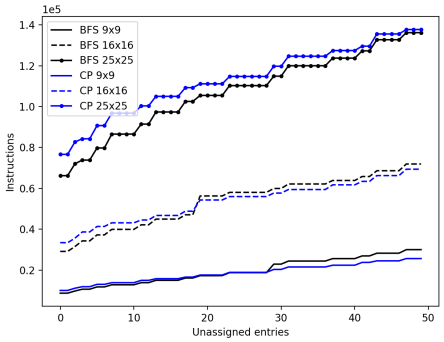

Figure 8: Comparison of number of instructions required for boards with differnet sizes with CP and BFS solving methods.

for each solver which, given the number of branches taken, outputs the expected number of instructions.
Despite being generally known that as the size of the board increases the problem rapidly becomes intractable(due to the exponential growth of possible combinations). However, for a fixed board size, the growth associated as a function of the number of unsigned entries seems close to linear. This can be seen in Figure 8.
Additionally, given that some of the solvers shown to be slightly sensitive to board configuration, it will be useful to save the sample standard deviation along with the metric mean for each of the request parameters in the Metric Storage System.

5. Conclusion and Future Work

In this checkpoint, I developed the software infrastructure for the entire sudoku cloud system. The data collection is operational, the load balancing and the resource allocation are as well despite using simple policies.
For future work, I intend to develop more sophisticated resource allocation policies that use the insights acquired during the experiments made with instrumentation as well as the data collected in the metric storage system. Additionally, I also intended to study and optimize the system to make a compromise between the quality of service and the costs associated with the Amazon services.

6. Bibliography

ZORN, B., AND LEE, H. B. Bit:a tool for instrumenting java bytecodes. Advanced Computing Systems Association.