Cloud Computing and Virtualization 2015-2016

Project Checkpoint

Group 27
52467 Tiago Simão, 69769 Nuno Cameira, 70993 Vasco Loureiro

Source code

The source code can be found inside the submitted project's zip in the following locations.

Webserver

Inside cloudprime/webserver/ folder, there's a maven project containing all the webserver source code. The actual .java files are therefore located at cloudprime/webserver/src/main/java/

Factorization

The factorization class is part of the webserver project. It is a single .java file located at cloudprime/webserver/src/main/java/org/irenical/ist/cnv/cloudprime/logic/IntFactorization.java

Instrumentation Tool

Inside cloudprime/instrumentator/ folder, there's a maven project containing the instrumentation tool's source code. It is a single .java file located at cloudprime/instrumentator/src/main/java/org/irenical/ist/cnv/cloudprime/Instrumentator.java

Binary Code

The binary code is available after building the whole project. This can be achieved with maven by running *mvn package* command at the project's root, cloudprime/.
For convenience the project was zipped after build, so this step is not mandatory.

Webserver

The webserver's class files are located (after build) at cloudprime/webserver/target/classes

Factorization (instrumented)

The instrumented factorization class is located alongside the webserver classes (after build) at
cloudprime/webserver/target/classes/org/irenical/ist/cnv/cloudprime/logic/IntFactorization.class

Instrumentation Tool

The instrumentation tool's class is located (after build) at
cloudprime/instrumentator/target/classes/org/irenical/ist/cnv/cloudprime/Instrumentator.class

Application Bundle

The bundled class files and resources are located (after build) at
cloudprime/webserver/target/webserver.jar, as an executable jar. It can be run by calling
java -jar cloudprime/webserver/target/webserver.jar

System Architecture

EC2 Instances

Each cloudprime computing node is an EC2 instance (default Amazon Linux image) with the userdata as described in this project's cloudprime/ec2/userdata file.
On boot each EC2 instance fetches and executes the file at cloudprime/ec2/boot.sh, by a wget to Amazon S3. This bash script is responsible for configuring the machine, installing needed packages, fetching the webserver's application bundle, also located at Amazon S3, and spinning up said webserver.

Auto-scale

To the best of our knowledge, the auto-scale policies provided by Amazon are limited to reacting over CPU, disk and network usage metrics. With this premise and considering the highly CPU bound cloudprime's workload, we chose CPU average usage as the metric for both up and down-scaling. We're using one cores per instance, so if the CPU usage stays **above** 50% for more than one minute we trigger a new EC2 instance provisioning, since this means that a new job cannot be accommodated without degrading the overall node performance. The reciprocate rationale is applied to EC2 nodes teardown.
For high availability reasons, the minimum number of nodes is 2. As of now the maximum number of instances is not an issue, other than economical. This might not hold as true in the final project submissions, for the custom load balancing we will be developing might have scalability limitations and therefore a maximum number of nodes it can relay work to.

Health check

A node's health is determined by it's HTTP REST endpoint at <ec2-instance-dns-name>:8080/status. A 200 OK response code means the node considers itself healthy and available for more requests. The health report implementation currently consists of the following: a node will consider itself healthy if it still has available threads to receive further computation work.

Load Balancing

Since the final load balancing is out of scope of the checkpoint submission, we're currently using Amazon ELB to round-robin HTTP requests across all healthy EC2 nodes.

Code deployment

Running the cloudprime/deploy.sh bash script will build and copy to Amazon S3 both the boot.sh script and the webserver.jar executable Java application. Due to the EC2 instance configuration, booting new instances or rebooting existing ones will be enough to start using the new code and/or boot configuration.

Collected metrics

Balancing a request will take in consideration both the state of each node and the weight of the incoming workload.
With that in mind we chose to we chose to collect two main metrics

1. Occupied threads per EC2 node
   Knowing if a node can accommodate more work will be a key information in the load balancing logic.
2. Instructions executed per number (total operation cost)
   This metric will be measured by each node (with the same outcome) and reported to a central storage. This is how Cloudprime system will learn the complexity of calculating a number in future requests.

We also collect methods executed per number (operation depth). This metric might be useful if want to shard a number across multiple nodes, but this is out of scope of this project.

Example logs

At the beginning and end of each request, we update the number of active threads.
INFO: Factoring: 1337 (1/15 active threads)
At the end of each request, we report the total operation cost and depth
INFO: Factorization of: 1337 took 11100 instructions and 8 methods

Running the application

Since the project's zip file already contains the compiled bundle, running the webserver can be done by executing the **run.sh** bash script located at cloudprime/run.sh. Otherwise, a build is required first (by running the *mvn package* command).

Example:
$ ./run.sh &
INFO: Started listener bound to [0.0.0.0:8080]
Mar 31, 2016 11:44:52 PM org.glassfish.grizzly.http.server.HttpServer start
INFO: [HttpServer] Started.
$ curl http://localhost:8080/f.html?n=1337
Mar 31, 2016 11:45:08 PM org.irenical.ist.cnv.cloudprime.stats.Metric start
INFO: Factoring: 1337 (1/15 active threads)
Mar 31, 2016 11:45:08 PM org.irenical.ist.cnv.cloudprime.stats.Metric end
INFO: Factorization of: 1337 took 11100 instructions and 8 methods
[7,191]

The last line in the console is the JSON formatted HTTP response (the curl output) with the prime factors.