Sudoku@Cloud - Checkpoint

Cloud Computing and Virtualization - 2019/2020

MEIC - IST - ULisboa

Samuel Vicente - 87704
Instituto Superior Técnico
samuel.vicente@tecnico.ulisboa.pt

João Marques - 90865
Instituto Superior Técnico
joao.a.marques@tecnico.ulisboa.pt

ABSTRACT

In this report we demonstrate the core decisions and results obtained during the planing and structuring of our Cloud Computing and Virtualization Project.

1 IMPLEMENTATION

At this moment we are able to handle several request at the same time in the same WebServer instance and across an Auto Scaling group. The server will solve the puzzle requested using instrumented Solver classes. Metrics are gathered and the WebServer periodically processes the information about the finished requests and stores the values locally.

1.1 Architecture

At this stage we make use of the AWS Classic Load Balancer and AWS Auto-scaling group. Most of our work revolved around the WebServer and Metrics Storage System. We made sure to have a thread-safe handling of the metrics by using ThreadLocal variables to access the RequestInfo specific to each request thread. The instrumented code will be run by the thread and update this static thread-local metrics. After the puzzle is solved the RequestInfo object will be added to a ConcurrentLinkedDeque. A single thread is scheduled to process this Deque periodically and store the metrics data locally.

1.1.1 ***WebServer***.

Relevant Classes:

- ThreadLocalRequestInfo: Holds a thread local static variable of class RequestInfo that gathers all info about the request and its metrics, if the solver class is instrumented. This variable is accessible by the classes in the solver package.
- ICountCNV: Instrumentation tool based on the ICount sample. The instrumented code updates the ThreadLocalRequestInfo to avoid sync problems between threads.
- WebServer: ConcurrentLinkedDeque "requestsInfo" stores RequestInfo of handled requests waiting to have their metrics processed.

1.1.2 ***Load Balancer***.

We used the AWS Classic Load Balancer configured to fit our needs. We continued to use the same ports for forwarding as in the labs. For the health check we decided to use the same handle but with the quickest call possible, an already completed board with 0 unassigned entries. This allows us to have a reliable check of the server without too much overhead. We decided to keep the Stickiness disabled for now, because we have no guarantee that each user will make several requests, and session wise we do not require it. Some puzzles may take a long time to run in the free tier machines, and we also have data stored locally which might need processing, so we have left the connection draining at 300 seconds.

- Port Configuration: 80 (HTTP) forwarding to 8000 (HTTP)
- Health Check: Puzzle with 0 unassigned entries and a board already completed.
  - Timeout: 10 seconds
  - Interval: 30 seconds
  - Unhealthy threshold: 2
  - Healthy threshold: 10
- Connection Draining: Enabled, 300 seconds
- Stickiness: Disabled

1.1.3 ***Auto-Scaler***.

We used the AWS Auto-scaling group configured to our needs.

The machine images used will run our auto-run.sh script on boot using /etc/rc.local. This script will instrument the classes and start the server on the port 8000. The machines will have the advanced monitoring enabled. The Health Check Grace Period was left very high for the moment, because we noticed that the launch may take long in some cases, but we will have another inspection on it later. The current group cap of 10 instances was set to avoid unnecessary costs while testing.

Auto Scaling Group

- Group size 1
- Minimum Group Size 1
- Maximum Group Size 10
- Health Check Type ELB
- Health Check Grace Period 300
- Detailed Monitoring Yes
- Instance Protection None

We used percentages for the increments to allow for scalability. We also used 2 steps to provide scalability speed when needed. Instances take some time to start and terminate, so we required that the the high cpu was noticed in 2 consecutive periods, to be sure the need for the extra instances would still be there after it finishes launching.

Scaling Policies

- Increase Group Size
  - alarm threshold: CPUUtilization >= 60 for 2 consecutive periods of 60 seconds
  - Add 10 percent of group when 60 <= CPUUtilization < 80
  - Add 20 percent of group when 80 <= CPUUtilization < +infinity
  - Add instances in increments of at least 1 instance(s)
  - 60 seconds to warm up after each step
- Decrease Group Size
  - alarm threshold: CPUUtilization < 20 for 2 consecutive periods of 60 seconds
  - Remove 10 percent of group when 20 >= CPUUtilization > 10
  - Remove 20 percent of group when 10 >= CPUUtilization > -infinity
  - Remove instances in increments of at least 1 instance(s)

1.1.4 ***Metrics Storage System.***

Local storage in the instance of each WebServer.

Relevant Classes:

- TempStorage: Runnable class that stores metric data locally. It will access the ConcurrentLinkedDeque "requestsInfo" periodically and update the local storage with the new processed info.
- WebServer: Has a SingleThreadScheduledExecutor that fires the TempStorage, that saves the context of each thread and writing it to a stats.backup file.

1.2 Instrumentation

For our initial testings we have adapted the ICount sample to use our thread-safe data structures. We instrument all the classes in the provided solver package, but plan to analyse if a better targeted instrumentation will be better.

1.3 Metrics Analysis

We have been analysing the metrics previsions, their performance and comparing it with an un-instrumented baseline cpu-time. We provide supplementary files with some graphs of our measures. Due to the circumstances of our group we didn't have time to generate results and analysis to the level we wished for this checkpoint. But it is something we are working actively.

2 FUTURE WORK

We wish to make a more in-depth analysis of the metrics. We plan to have a new machine image running which will access the data from CloudWatch and control the load balancing and auto-scaling. That machine could use HTTPRedirect to distribute the load. We are still discussing about the level of redundancy needed for the load balancer and auto-scaler instance. A DynamoDB database will be used in addition to the local storage. The local storage could be used as a cache to the DynamoDB.

A DIAGRAMS

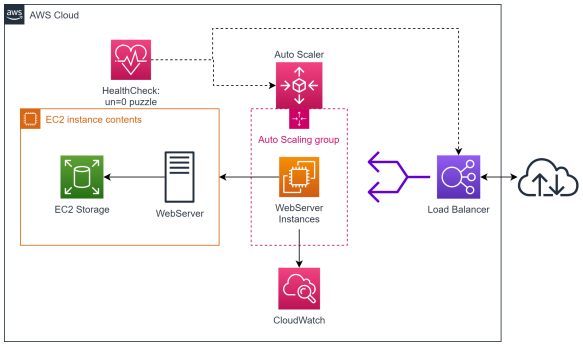

Figure 1: System

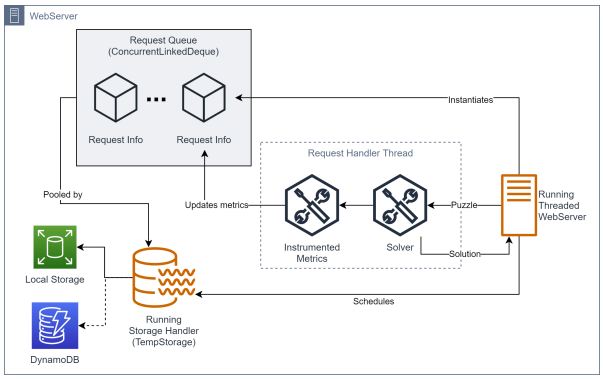

Figure 2: WebServer Instance