CLOUD COMPUTING 2012-2013(SEM 1)

Weather Data

PROJECT REPORT – GROUP 4

Name	Student Number
Pushparaj Motamari	77160
Anh Phuong Tran	77182
Dipesh Dugar Mitthalal	77184

Map Reduce:

Mapper:

Given input is split using comma and City name appended by Date will form the Key Value will be a string representation of Temperature, Humidity and rainfall separated by semi colon.

Combiner:

Combiner is used in this Mapreduce program to reduce the number of key value pair hitting the Reducer. In reducer, Mathematical operation is applied to find Maximum temperature, Minimum temperature, average temperature, Maximum humidity, Minimum humidity, total humidity, Total rainfall and a counter. Counter is passed along with other values to keep track of number of records processed for a particular day.

Combiner is incorporated to enhance the performance of the whole map reduce program as it reduces the computation in reducer. Also there is reduction in output to disk and reading from disk in terms of pairs emitted from Mapper.

Reducer:

In Reducer, the operations performed in combiner is are repated.In addition to that, the output from reducer will be inserted into DynamoDB. We assume that table is already created and present in Dynamo DB. The validation for presence of table is not included in reducer. The table structure with the combination of city name and date as key and Maximum Temperature, Minimum Temperature, Average Temperature, Maximum Humidity, Minimum Humidity, Total Humidity, Rainfall and counter is preferred. Counter is used to check if there is already processed data present for same Date and city for how many hours.

So if a log file comes with 10 hours of data for a city and date and another file with remaining 14 hours of data, the proposed system will be able to handle it.

<u>Algorithm:</u>

```
Step1: Check for the count of records before insertion.
Step2: If it is 24 then insert into dynamo DB
Step 2.1: Else check for the presence of record with same key in Dynamo DB
          And retrieve it and process it along with the present data and update it.
Step 3: If step2.1 fails insert the available processed data with appropriate counter
```

DynamoDB's Write Throughput is set to 24. Other attempts using lower Write Throughput (5, 10) didn't succeed (for the moderate sample file (65k)). This Throughput depends much on the number of Reducers that Amazon EMR provides. (The higher the number, the more write attempts to the DB are made in the same span of time). It appears that Amazon EMR provide up to 20 reducers (base on log-file). Any Throughput number larger than 20 seems to work.

Load Balancer:

Load balancing is achieved by following the steps below.

1. **Created an AMI(**Amazon Machine Image) ami-5201853b from base image of id ami-c49217ad on availability zone:: North Virginia.

Apache installed on the the instance with AMI ami-5201853b uses mpm_prefork_module module to manage the servers. Version of apache installed on AMI ami-61a02008 is 2.4+.

Following changes were set in the apache httpd.conf file at path, /opt/bitnami/apache2/conf/bitnami/httpd.conf

```
  <IfModule mpm_prefork_module>
  #start with 2 servers
 StartServers   2
  # keep 2 servers ready to handle requests
 MinSpareServers 2
  # do not keep more than 3 servers idle
 MaxSpareServers 3
<IfVersion >= 2.3>
                 #Maximum number of connections that will be processed simultaneously.
                         MaxRequestWorkers      5
                 #Limit on the number of connections that an individual child server will handle during
its life.
                         MaxConnectionsPerChild  20
                 </IfVersion>
                 <IfVersion < 2.3 >
                         MaxClients           20
                         MaxRequestsPerChild     5000
                 </IfVersion>
         </IfModule>
```

1.1) Created an AMI(Amazon Machine Image) with the id ami-61a02008 from the base image of id ami-c49217 in the availability zones :: North Virginia. This AMI contains the files to serve the request and has the default apache configuration. The configuration is

```
<IfModule mpm_prefork_module>
  StartServers    5
  MinSpareServers 5
  MaxSpareServers 10
  <IfVersion >= 2.3>
    MaxRequestWorkers      20
    #MaxConnectionsPerChild  5000
    MaxConnectionsPerChild  1000
  </IfVersion>
  <IfVersion < 2.3 >
    MaxClients           20
    MaxRequestsPerChild     5000
  </IfVersion>
</IfModule>
```

2. **Create an auto scaling configuration** with Amazon AutoScaling Command Line Interface tool using the command

```
as-create-launch-config  apcheDefaultConfigBitnami --image-id ami-61a02008 --instance-type t1.micro
```

3. Create a load balancer using the UI in the Amazon EC2 console page.

1. Created the loadbalancer Cloudldblncr1 on the avaialability zone us-east-1c(North Virgnia) with the following Health Check configuration.

Ping Target: HTTP:80/index3.html
Timeout: 2 seconds
Interval: 6 seconds
Unhealthy Threshold: 2
Healthy Threshold: 2

The descriptions are ::

Ping Target:
Specifies the instance being checked. The protocol is either TCP, HTTP, HTTPS, or SSL. The range of valid ports is one (1) through 65535.

Timeout:
Specifies the amount of time, in seconds, during which no response means a failed health probe.

Interval:
Specifies the approximate interval, in seconds, between health checks of an individual instance.

UnHealthy Threshold:
Specifies the number of consecutive health probe failures required before moving the instance to the Unhealthy state.

Healthy Threshold :
Specifies the number of consecutive health probe successes required before moving the instance to the Healthy state.

4. Create an auto scaling group with Amazon AutoScaling Command Line Interface tool using the command, whose minimum size is 1 and maximum size is 10.

This auto-scaling group is configured with loadbalancer created in the previous step.(Cloudldblncr1).

```
as-create-auto-scaling-group cloudAutogrp1 --availability-zones us-east-1d,us-east-1c --launch-configuration cloudlnch2  --max-size 15 --min-size 4 --load-balancers  Cloudldblncr1
```

5. Create policies

Policy1: Increase the instances in auto scaling group by 1.

```
as-put-scaling-policy cloudplcy3 --type ChangeInCapacity --auto-scaling-group cloudAutogrp1   --adjustment 1 --cooldown 20
```

Policy2: Reduce the size of the instance count in the auto scaling group to 1.

```
as-put-scaling-policy cloudplcy4 --type ExactCapacity --auto-scaling-group cloudAutogrp1   --"adjustment=-1" --cooldown 20
```

6. Execute the policies

Execute the polcies created on the auto scaling created in th previous steps using the below commands.

1. as-execute-policy cloudplcy3 --auto-scaling-group cloudAutogrp1

2. as-execute-policy cloudplcy4 --auto-scaling-group cloudAutogrp1

7. Create alarms:

Alarms have three states: ALARM, OK, and INSUFFICIENT DATA. The state of your alarm changes according to a threshold you specify. First, define the criterion

for entering the ALARM state. Later, you can specify an action to be taken when your alarm enters any of the three states.

Create following alarms

ThreshHold: It is the condition, when true , the alarm will enter the ALARM state.

1. Alarm Name: BitnameAlarm1

Configuration of Metric.
UnHealthyHostCount >= 1 for 1 minutes
MetricName: UnHealthyHostCount
Dimensions: LoadBalancerName = Cloudldblncr1
Statistic: Average
Period: 1 Minute

Actions executed when state is ALARM
Actions: in ALARM state - Use policy "cloudplcy3 (Add 1 instance)" for group "cloudAutogrp1".

2. Alarm Name: BitNamiAlarm2

Configuration of Metric.
Threshold: HealthyHostCount >= 9 for 1 minutes
MetricName: HealthyHostCount
Dimensions: LoadBalancerName = Cloudldblncr1
Statistic: Average
Period: 1 Minute

Actions executed when state is ALARM
Actions: in ALARM state - Use policy "cloudplcy4 (Exactly 1 instance)" for group "cloudAutogrp1"

3. Alarm Name: MaxUnHealthyHostCount1Alaram

Namespace: AWS/ELB

MetricName: UnHealthyHostCount >= 10

Dimensions: LoadBalancerName = Cloudldblncr1

Statistic: Average

Period: 1 Minute

Actions: in ALARAM state: Use policy "cloudplcy3 (remove 1 instance)" for group "cloudAutogrp1".

Initial Setup:

1. Add 5 EC-2 instances of AMI id ami-5201853b to the load balancer.
2. Add 4 instances of AMI id ami-61a02008 to the load balancer. Which are also present in auto-scaling-group.
3. Start the JMeter using the WebSiteTestPlan.jmx. This will pump requests to the load balancer. The number of requests , number of users and time between requests can be configured. Time to wait for a response can also be configured, which is useful in testing the load-balancing setup

Working Explanation:

1. When the test setup starts there are total 9 instances of which 5 instances from ami-5201853 and 4 instances from ami-61a02008. The load balancer distributes the requests equally among the 9instances.In this process some instances among the 5 instances with slow apache configuration will become unhealthy triggering the alarm to add one more instance to the auto-scaling-group, which will be added to the load-balancer.

   T1 = Amount of request processing time available with the present instance pool of load balancer.
   T2 = Time for processing the alarm. + Time for a instance to get running.

   In order to serve all the requests. The following condition must hold.

   T1 > T2.

   The 5 instances of AMI ami-5201853 can handle requests =
   5* MaxSpareServers * MaxRequestWorkers = 5*3*5 =75.

   The four instances of AMI ami-61a02008 can handle rquest =
   4* MaxSpareServers* MaxRequestWorkers = 4*10*20 = 800.

   By the time all the 5 instance with ami-5201853 are full of capacity . The no_of requests available in the group are almost equal to
   800 – 4*(15) = 800 -60 = 740. Assuimg round robin distribution of requests by the load-balancer.

   The time to get all the 740 requests to get filled = 740*wait_time(specified in the request).

   If wait_time specified in the request is 10 seconds, then total Amount of request processing time available with the present instance pool of load balancer is 740*10 = 7400 seconds.

   T2 is 10 minutes i.e 600 seconds in the worst case as per Amazon. This implies
   7400 > 10*60 =600. That is T1 > T2. This proves that the load balancing set up explained above serves all the requests.

   Since the 5 instances with AMI-Id ami-5201853 are not attached to the load balancer, they will join the load balancer instance pool once they clear all pending requests and become healthy. They again help in load balancing.