Understanding Impact of Hadoop Schedulers on Fog Nodes

Prashanth Kumar Purshotam
pp7080@rit.edu
Rochester Institute of Technology
Rochester, New York

Abstract

With an increase in edge devices in today's world. It is highly important to understand the various designs and architectures of cloud systems and how certain factors affect their performance. In this project, there is a focus on fog computing with respect to Hadoop and how the scheduler plays a major role in the performance of the system. The goal of the project is to utilize existing tools in Hadoop to understand how the scheduler impacts the systems performance without the need to run fog nodes to achieve the same.

Keywords Cloud Computing, Fog Computing, Hadoop, Schedulers, JVM

1 Introduction

Fog nodes are described as intermediate computing nodes that act as a bridge between edge devices and IOT devices and the cloud. These devices are not as large as the cloud and their main functionality is to deal with data that is constantly streaming and time sensitive. Cisco coined the term and they provide the following functions in a white paper they presented[3]:

- Analyze time-sensitive data which is generated at the network edge, instead of sending a huge amount of data generated by edge devices to the cloud for processing (closer to data source).
- Based on the algorithms utilized act on incoming data within very small timeframes (miliseconds).
- Selectively send data to the cloud for long term storage, which enables analysis of data and create models based on the data.

These devices generally have lesser capacity in terms of memory, processing power and hence will have certain constraints on the type of operations they handle[4].

This project focuses on how certain configurations in Hadoop can impact the processing power of Fog Nodes (given their constraints). This project has chosen Hadoop as it is still the preferred file system when comes to storing data. Considering fog nodes must handle constant incoming data from edge devices and considering the cloud they upload the data to predominately utilize Hadoop (in the interest of maintaining a homogenous environment), it focuses on the impact Hadoop scheduler configurations have on fog nodes.

2 Hadoop schedulers

Hadoop is essentially a multitasking system that handles different types of databases for clients that submit multiple job requests (to process data). Every job request by a client must be scheduled before it is executed. As resources can only be accessed finitely and the amount of resources each job utilizes varies. Hence, the order in which they are scheduled, the priority each job is given or the amount of resources it receives affects the overall performance of the system.

Considering the extra constraint placed on fog nodes[4] in terms of memory and compute power, the scheduler configurations play an important role in the overall performance of a fog node.

There are three types of Hadoop schedulers:

- FIFO scheduler
- Fair scheduler
- Capacity scheduler

2.1 FIFO Scheduler

This is the default scheduler used by Hadoop and it uses the principle of a queue where the first job is given resources to execute and then the next scheduled job and so on. It does not take in consideration factors like job size or the priority of the job. Hence a high priority job could be waiting in line to be executed when a lower priority job takes up the resources.
Considering this is the default scheduler in Hadoop, there is no need for a change in the configurations in the file yarn-site.xml file found in the /etc/Hadoop folder.

```
pp7080@resourcemanager:/usr/local/hadoop/etc/hadoop$ cat yarn-site.xml.save
<configuration>

  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>resourcemanager</value>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>127.0.0.1:8088</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

Figure 1. Here is a snap shot of the FIFO Scheduler file

2.2 Fair Scheduler

Fair scheduler allots resources to users in a manner which benefits all apps, they on average get the same share of resources over a period of time. Fair scheduler can handle scheduling jobs based on memory precedence and CPU precedence (it uses Dominant Resource Fairness to achieve the same).
Fair scheduler allows a single job to utilize the entire resource however if there are multiple jobs entering the cluster the access to memory and CPU is gradually decreased for each and every job. However, the amount of resource they receive is equal. This ensures that small jobs finish quickly and longer duration jobs don't starve for resources and execute on what is available to them. This is unlike the FIFO scheduler where only one job can execute at a time.

```
pp7088@resourcemanager:/usr/local/hadoop/etc/hadoop$ cat yarn-site.xml
<configuration>
  <property>
  <name>yarn.resourcemanager.scheduler.class</name>
  <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>resourcemanager</value>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>127.0.0.1:8088</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
pp7088@resourcemanager:/usr/local/hadoop/etc/hadoop$
```

Figure 2. Here is a snap shot of the Fair Scheduler file

2.3 Capacity Scheduler

The main aspect of the capacity scheduler is to utilize the cluster as a multi-tenant system. The origin of this cluster is mainly linked to cloud systems funded by multiple organizations who wish to share the resource they paid to maintain. Thus, this can allot capacities (a portion of the cluster) based on the agreed percentage of resources each contributor has access to. Though the scheduler places a limit on the capacity each user has access, once the resources are free or when they are not being used a user can access the entire cluster or the portion that is being unused.

This is achieved by creating multiple queues, each queue has access to certain percentage of the capacity of the resources, a job submitted by the user (say having a restriction of 10 percent of the capacity) will be placed on Queue A which is only for jobs that need 10 percent of the capacity.

3 YARN Scheduler Load Simulator

Apart from choosing a scheduler one has to tweak several other parameters before testing them on jobs to be executed. The YARN scheduler load simulator[2] allows users to use real job traces and simulate real work loads without the execution on real clusters. This allows the project to create different types of work loads (that are closer to the ones

```
pp7088@resourcemanager:/usr/local/hadoop/etc/hadoop$ cat yarn-site.xml
<configuration>

<property>
    <name>yarn.resourcemanager.scheduler.class</name>
    <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</value>
  </property>


  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>resourcemanager</value>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>127.0.0.1:8088</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

Figure 3. Here is a snap shot of the Capacity Scheduler file

experienced by Fog nodes) and simulate the work load to test the configuration created.

This simulator produces real time metrics while executing the jobs mentioned on the job trace. Some of the metrics are:

- Resource usages for whole cluster.
- Resource usage of each queue in the configuration.
- Execution trace of each job in simulated time

In our execution, we change the sls-runner.xml file (which is the file used by the YARN Schduler Load Simulator[2]) to simulate a fog node. We do this by selecting very little memory and a small thread pool setting. The number of cores selected are 2. This is done to mimic a fog device, the configuration is shown in figure 4.

```
<!-- SLSRunner configuration -->
<property>
  <name>yarn.sls.runner.pool.size</name>
  <value>10</value>
</property>

<!-- Nodes configuration -->
<property>
  <name>yarn.sls.nm.memory.mb</name>
  <value>1024</value>
</property>
<property>
  <name>yarn.sls.nm.vcores</name>
  <value>2</value>
</property>
```

Figure 4. Here is a snap shot of the Capacity Scheduler file

4 Experiment

The idea of the simulation is to take a real Hadoop job log and use it to run the YARN simulator[2]. However, there are several steps that are required before a Hadoop job log can be transformed into a file that is suitable for the simulator. We first begin with a suitable file that contains Hadoop job logs. This was found by requesting researchers to share the files they used to perform their research. The search led us to LogPai who run a github profile called LogHub[6].

The logs present on this github profile have been used in the research that resulted in the paper published by Jieming Zhu et. al. called Tools and Benchmarks for Automated Log Parsing[5]. The authors shared the logs which was used to build a Rumen file[1]. Rumen is a data extraction and analysis tool built to convert job logs to Rumen files.
The rumen file is further transformed into an SLS file (which is a format required by the YARN simulator[2]) by using a file provided by Hadoop called rumen2sls.sh. This file produces the sls file that is needed for our experiment. Once this file is obtained, we use it to run the YARN simulator along with the scheduler configurations described above.
Though one can produce results just by using Rumen files, the only drawback with this approach is the inability to define the node structure of the configuration. By using the sls file format, Hadoop gives us the ability to integrate a node file that defines the configuration on which we wish to run a simulation.

This produces an output in the form of a JSON file that is studied to understand the impact of the scheduler on the simulation. The JSON file produced from the simulation takes provides various data points that vary with respect to time. Time is recorded as a unix time stamp and the corresponding data points are recorded as key, value pairs along with data. The key data points that are of interest to us is noted and weighed against other data points which will give us an indication about the performance of the configuration.
The data points are run through a python file that takes in a json input and creates a dataframes. It utilizes the dataframes to plot a graph where one can visually compare the way two data points change with respect to time.

5 JVM Memory vs Performance

Some of the data points that are generated from the YARN load simulator which are of particular interest to this project are the ones related to JVM memory. In order to evaluate the performance of a scheduler one has to understand the efficiency with which jobs are executed when they are assigned to a cluster by the same. One data point that is a good indicator of efficient execution is the JVM memory of a node when execution occurs.
According to the research produced in papers studying JVM effects on Data Analytics tools[7], it is shown that applications performance suffer when the JVM memory used is higher than normal. Thus, by studying the variance of data points related to JVM memory (like free JVM memory, total JVM memory etc.), one can infer if the application performance is generally poor or execellent for a given configuration and Scheduler setup. The rest of the project looks at plotting various Scheduler parameters against JVM memory parameters in order to assess if a Scheduler is good enough for a particular configuration.

6 Results

From the data points produced in the json file (created by running the YARN load scheduler) we study the following key points:

- jvm.free.memory - Amount of free memory in the JVM.
- jvm.max.memory - Max available JVM memory at that instant.
- jvm.total.memory - Total memory availabe.
- running.applications - Number of applications running.
- scheduler.handle-NODEADDED.timecost - total cost of adding a node
- scheduler.handle-NODEUPDATE.timecost - total cost of updating a node.

Once these data points are obtained and passed through the python file they will give us an understanding of how the application performed (by studying the JVM performance). Here is a sample image that shows scheduler.handle-NODEUPDATE.timecost (the value related to updating a node) and the free JVM memory. This has been generated after passing the json file from the YARN simulator to the python file.

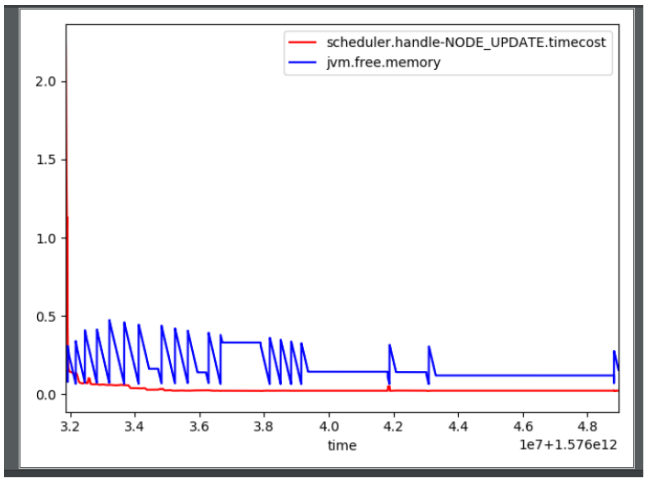

Figure 5. Comparison of Scheduler node update to JVM

This setup was a Capacity Scheduler in a simulated fog node setup and this shows how the change in Scheduler activity does not drastically affect the JVM memory left in the node. This tells us that Capacity Scheduler might be a good fit for such a configuration. This conclusion does not come from the one graph alone as an application developer must verify several such graphs before coming to conclusion. The other positive aspect about this methodology is that it prevents the user from having to run instances in order to recreate the work environment. This simulator produces near accurate results which are good from a starting perspective.

7 Limitations

One of the biggest drawabacks with this approach is that it cannot accurately tell a developer that a particular scheduler is the perfect for a configuration. However, it gives an accurate estimate of how the scheduler would affect performace for a given configuration without having to run the clusters.

8 Conclusion

This method is only meant to give the developer a starting point from which they can work their way towards choosing a perfect scheduler for their fog node configuration. It is not meant to replace other common methods that might be used to choose the best Scheduler.

References

[1] APACHE. [n.d.]. *Rumen: Data extraction and analysis tool built for Apache Hadoop.* https://hadoop.apache.org/docs/current/hadoop-rumen/Rumen.html

[2] Apache. [n.d.]. YARN Load Simulator. ([n. d.]). https://hadoop.apache.org/docs/current/hadoop-sls/SchedulerLoadSimulator.html

[3] CISCO. [n.d.]. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. *White Paper* ([n. d.]). https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf.

[4] Jordi García-Almiñana Admela Jukan Guang-Jie Ren Jiafeng Zhu Josep Farré Eva Marín Tordera, Xavi Masip-Bruin. [n.d.]. What is a Fog Node? A Tutorial on Current Concepts A Tutorial on Current Concepts towards a Common Definition. *IBM Research* ([n. d.]). https://arxiv.org/ftp/arxiv/papers/1611/1611.09193.pdf

[5] Jinyang Liu Pinjia He-Qi Xie Zibin Zheng-Michael R. Lyu. Jieming Zhu, Shilin He. [n.d.]. Tools and Benchmarks for Automated Log Parsing. ([n. d.]). https://arxiv.org/pdf/1811.03509.pdf

[6] LOGPAI. [n.d.]. *Freely accessible logs from production data for research purposes.* https://github.com/logpai/loghub

[7] Shivnath Babu Mayuresh Kunjir, Yuzhang Han. [n.d.]. Where does Memory Go?: Study of Memory Management in JVM-based Data Analytics. ([n. d.]). https://pdfs.semanticscholar.org/8590/b5d66e0dc429578cf6ac64b8abda6a125701.pdf