Comparing Multi-Server Configuration for Big Data Processing using Hadoop, SWIM and Gnuplot

Praditya Raudi Avinanto 13514087
Informatics / Computer Science
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Jl. Ganesha 10 Bandung 40132, Indonesia
13514087@std.stei.itb.ac.id

***Abstract*—Nowadays most of enterprises are using big data to run their business. And for business sake they just want to use necessary specification of multi-server to handle this bunch data. But it's really difficult to find a method to know the most suitable specification for them. They need to compare any options of specification for them, which is will very costly. For this problem SWIM (Statistical Workload Injector for MapReduce) benchmark is the proposed solution. SWIM can do rigorous performance measurement of MapReduce systems. SWIM contains suites of workloads of thousands of jobs, with complex data, arrival, and computation patterns. So SWIM can change the given workload to be most suitable form for your server to handle and it depends on how much memory and sum of nodes do your server had. This workload called as synthetic representative workload. Furthermore we can run it on Hadoop so you can see the statistical result of the performance of your configuration by runtime execution of a task with that big data. Then we can visualize the data using Gnuplot to see the difference. For this experiment we used Hadoop because this software is the origin of MapReduce and integrated to SWIM. Based on experiment you can compare and decide which one of the configuration is good enough to support your enterprise or business.**

Keywords—big data; MapReduce; SWIM; hadoop; performance, gnuplot

I. INTRODUCTION

In this era it's very difficult to find any enterprise that just uses small amount of data for their business. Most of them are using big data now in order to keep them stay in competitive environment. Of course this is become a problem because the bigger the data, the better performance computing we needed. And for business sake we don't want to waste our money for unnecessary high-end specification of multi-server configuration to handle this data. They just want the specification or configuration of distributed multi-server that are good enough to handle this bunch of data. And of course this will give you another problem, even maybe this become a bigger problem than before, if we see it on economy sector.

Today there is some method to know about the comparison of two multi-server configurations accurately. The first one is you need to try it on a similar configuration with the proposed one with the same amount of machine, and you do it for both configurations. Of course this method will burn your wallet. The other options is to hire a distributed-system analyst to calculate and estimate the comparison between two configurations that you proposed based on written specification of the machine and his experiences. This method will safe you more money but will be far less accurate than previous method. The third method is to use a cloud-virtual machine to virtualize your configuration and you can see the result afterwards. This method is as accurate as the first method but at some rate this method will be not effective. First thing is that not all cloud-virtual machines are free to use. We admit it that there are indeed free to use cloud-virtual machine. But this free services must be had some constraint to some extent of usage. For example maybe you're not permitted to use more than four PCs at one moment. But so far the third solution is the best solution because it used least cost and give most accurate result.

But even we have the better solution to this problem, we proposed SWIM (Statistical Workload Injector for MapReduce). SWIM can do rigorous performance measurement of MapReduce systems. SWIM can change the given workload to be most suitable form for your server to handle and it depends on how much memory and sum of nodes do your server had. This workload called as synthetic representative workload. Furthermore we can run it on Hadoop so you can see the statistical result of the performance of your configuration by runtime execution of a task with that big data. Then we can visualize the data using Gnuplot to see the difference. Although maybe this method is less accurate than the first method but this method's accuracy is not a joke. SWIM is really trusted approach because it proved by how frequently many developers around the world used it to compare the performance of their own experiment using distributed-system. This method beats other methods because it gives enough tolerable accuracy, free and has no constraint to use it.

In the following sections, we present literature study based on this topic, proposed method to solve this topic, experiment result of the method, discussion and conclusion.

II. LITERATURE STUDY

A. Distributed System

A distributed system is a model of a system in which each component is independent but still working together to serve the client. Each components or we usually call it nodes are connected so they can communicate and coordinate their actions by passing messages. The components interact with

each other in order to achieve a common goal. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components.

Distributed system is also used to solve computational problems. They solve it by divide the problem into many tasks and then each computer do the task simultaneous. And for some kind of problem they do message passing to each nodes.

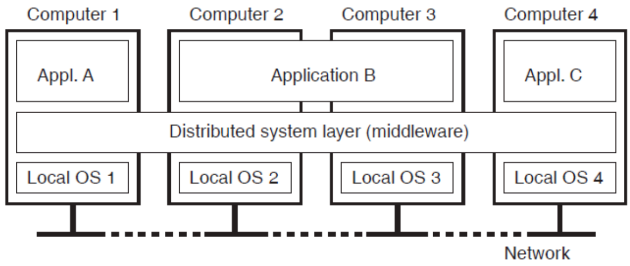

Fig 1. Architecture of distributed system

B. Hadoop

Apache Hadoop is an open-source software framework written in java for storing data and running applications on clusters of nodes that have the distribution of data. Hadoop has two core parts, they are storage part, known as Hadoop Distributed File System (HDFS), and processing part, known as MapReduce programming model.

The base Apache Hadoop framework is composed of the following modules:

- Hadoop Common – contains libraries and utilities needed by other Hadoop modules;
- Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
- Hadoop YARN – a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications and
- Hadoop MapReduce – an implementation of the MapReduce programming model for large scale data processing.

C. MapReduce

MapReduce is a processing technique and a framework for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map will convert a set of data into tuples (key/value). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. Map task is always performed first before Reduce task. Map reduce is frequently used for distributed computing because it support scalability on the data that being proceed.

Below is the explanation about how MapReduce works

- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
    - Map stage: The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop Distributed File System (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
    - Reduce stage: This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

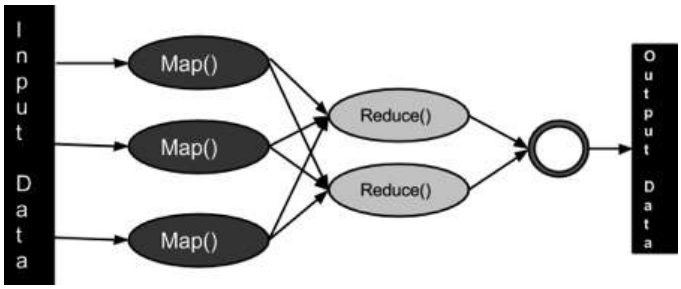

Fig 2. Visualization of mapreduce process

D. SWIM (Statistical Workload Injector for MapReduce)

SWIM includes

- Repository of real life MapReduce workloads from production systems.
- Workload synthesis tools to generate representative test workloads by sampling historical MapReduce cluster traces.
- Workload replay tools to execute the historical or test workloads with low performance overhead.

SWIM enables rigorous performance measurement of MapReduce systems. SWIM contains suites of workloads of

thousands of jobs, with complex data, arrival, and computation patterns. This represents an advance over previous MapReduce pseudo-benchmarks of limited diversity and scope. SWIM informs both highly targeted, workload specific optimizations, as well as designs that intend to bring general benefit.

SWIM can give you an accurate representative workload of big data so it will suit your configuration for further objectives like testing or comparing some distributed system. SWIM is currently integrated with Hadoop. The performance and evaluation science behind it is extensible to MapReduce systems in general.

E. Gnuplot

Gnuplot is a free to use command-line based program for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. This program is used to visualize mathematical functions and data. It can visualize the data in many forms including dot chart, line chart, bar chart, and etc. Furthermore this program can set the type of the output like png, jpg, postcard and manymore. Now Gnuplot has grown to support many non-interactive uses such as web scripting.

III. THE PROPOSED METHOD

This sections will explains about the proposed SWIM method, more precisely about how to compare two difference multi-server configurations that process big data using Hadoop and SWIM. The step is divided into four sub-sections. Those four sub-sections are preparing the nodes, preparing the workloads using SWIM, running the synthetic representative workloads using Hadoop, and visualizing the result using Gnuplot. In this experiment we will compare four nodes runtime execution and three nodes runtime execution using the same workloads on each topology. Before we start the explanation of each step we want to let you know what configuration we used for this experiment.

- Processor : 64-bit Intel Quad Core Xeon E5530
- Memory : 12 GB
- Operating System : Ubuntu 14.04 64-bit
- Connection : 1 Gbps
- Data : One hour Facebook workloads in 2010 (you can find it on the SWIM repository)
- Software for big data processing : Hadoop 2.7.1

In this section we assume the reader already know how to configure Hadoop in multi-node cluster, know how to run it, has all dependencies which are needed to run SWIM or Hadoop and have SWIM script from the Github or another sources. All scripts or programs that mentioned in the below are coming from SWIM Github repository. So the steps will be described in detail below

A. Preparing the Nodes

So first we need to configure our distributed system. The first one is consist of four nodes with one nodes as the master and the rest as the slaves. Connect it with LAN cable or WiFi as long as it has transfer rate 1 Gbps. If your network has a transfer rate more than 1 Gbps, its okay but the result will likely be different. But if you insist to gain 1 Gbps speed you can use traffic control or Unix tc command to slow down your transfer rate.

So then you can see the topology like this.

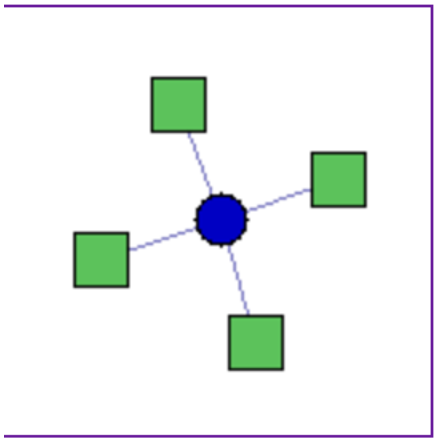

Fig 3. Scheme of four nodes configuration

After that, with the same manner, you configure again a distributed system that consists of three nodes. The topology will look like this

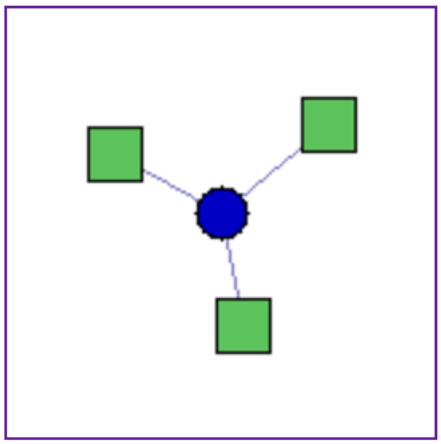

Fig 4. Scheme of three nodes configuration

B. Preparing SWIM Workload

So in this sub-section we will explain you the step of how to create representative workload using SWIM, but not too technically. If you want to read the full instruction of these steps you can visit the SWIM repository. For a brief explanation, SWIM will create a representative workload based on historical logs. In this case we use historical logs of Facebook's one hours workloads in 2010. For example if the real configuration needed to run that Facebook workload was 10 nodes of high-end server, then SWIM will create the new representative workload so that the burden felt by each of 10 nodes will be the same as the burden felt by each of nodes of our configuration. In general this sub-section consist of three steps.

- Parse historical Hadoop logs

  First we create a .tsv file from folder of historical workloads logs using parse-hadoop-jobhistory.pl . Here is the command

```
perl parse-hadoop-jobhistory.pl [job history
dir] > outputFile.tsv
```

- Synethsizes workloads by sampling historical Hadoop trace

  Then we create synthesizes representative workloads of short duration using potentially months of trace data parsed by Step 1 previously. The key synthesis technique is continuous time window sampling in multiple dimensions. Here is the command

```
perl WorkloadSynthesis.pl
       --inPath=FacebookTrace.tsv
       --outPrefix=FB-
       2010_samples_24_times_1hr_
       --repeats=2
       --samples=24
       --length=3600
       --traceStart=FacebookTraceStart
       --traceEnd=FacebookTraceEnd
```

  The final output of this step is FB2010_samples_24_times_1hr.tsv as the representative synthetic workloads.

- Generate scripts to execute the synthetic workload

  In this step we will convert the output of the previous step into scripts that call stub Hadoop jobs to reproduce activity in the workload.

  Here are the commands

```
javac GenerateReplayScript.java

java GenerateReplayScript
        [path to synthetic workload file]
        [number of machines in the original
production cluster]
        [number of machines in the cluster
where the workload will be run]
        [size of each input partition in
bytes]
        [number of input partitions]
        [output directory for the scripts]
        [HDFS directory for the input data]
        [prefix to workload output in HDFS]
        [amount of data per reduce task in
byptes]
        [workload stdout stderr output dir]
        [hadoop command]
        [path to WorkGen.jar]
        [path to workGenKeyValue_conf.xsl]
```

Commands above will generate a script test folder. Inside that folder there are 50 jobs scripts that are ready to be run.

C. Running the Hadoop

Running Hadoop consist of three steps:

- Creating the map and reduce scripts in java and change it to jar depends on then user's need

  On this step we will use HDFSWrite.java for writing the input data set. WorkGen.java as the map and reduce script for read/shuffle/write data with prescribed data ratios. Both scripts already given on the SWIM repository

- Move the input file from your file system to hadoop file system (HDFS) using HDFSWrite.jar

  Here is the command

```
bin/hadoop jar HDFSWrite.jar
org.apache.hadoop.examples.HDFSWrite -conf
conf/randomwriter_conf.xsl workGenInput
```

  Command above will write the input to HDFS folder named workGenInput

- Run the job and see the result or logs

  Here is the command

```
cd ${SCRIPT TEST DIR}
./run-jobs-all.sh
```

  run-jobs-all.sh is a bash script that calls all of the run-job-i.sh with 0 <= i <= 49. The result or logs of this script or jobs will be shown inside workGenLogs folder inside the script test directory.

D. Visualize the Data Using Gnuplot

You can see at figure 5, the runtime execution is shown at the last line of the log files. And this log files are corresponding to synthetic scripts above. So the total of the logs must be the same (50 log files).

```
                Data-local map tasks=1
                Rack-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=11603
                Total time spent by all reduces in occupied slots (ms)=3531
                Total time spent by all map tasks (ms)=11603
                Total time spent by all reduce tasks (ms)=3531
                Total vcore-seconds taken by all map tasks=11603
                Total vcore-seconds taken by all reduce tasks=3531
                Total megabyte-seconds taken by all map tasks=11881472
                Total megabyte-seconds taken by all reduce tasks=3615744
        Map-Reduce Framework
                Map input records=671089
                Map output records=139
                Map output bytes=15012
                Map output materialized bytes=15302
                Input split bytes=216
                Combine input records=0
                Combine output records=0
                Reduce input groups=139
                Reduce shuffle bytes=15302
                Reduce input records=139
                Reduce output records=33
                Spilled Records=278
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=405
                CPU time spent (ms)=9200
                Physical memory (bytes) snapshot=722149376
                Virtual memory (bytes) snapshot=2548035584
                Total committed heap usage (bytes)=588251136
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        org.apache.hadoop.examples.WorkGen$Counters
                MAP_BYTES_WRITTEN=13900
                MAP_RECORDS_WRITTEN=139
                RED_BYTES_WRITTEN=3300
                RED_RECORDS_WRITTEN=33
        File Input Format Counters
                Bytes Read=78593574
        File Output Format Counters
                Bytes Written=3944
        org.apache.hadoop.mapreduce.v2.app.job.impl.TaskImpl$TaskVerdict
                ORIGINAL_WIN=2
        org.apache.hadoop.mapreduce.v2.app.job.impl.TaskImpl$Topo
                Fn_nn=2
Job ended: Sat Apr 15 01:24:37 MDT 2017
The job took 24 seconds.
```

Fig 5. Snapshot of Hadoop's job log file

Then to visualize the total runtime execution by each configuration we need to extract just the last line of each log files to get the time elapsed by the job. To extract this log files, there are two options, the first option is manually type it by our hand, and the second one is to create a program to automatically extract all of the log files.

After we got the time elapsed data for each logs the file must be look like Table 1.

No Job	Experiment 1	Experiment 2
0	30	40
1	20	20
2	31	33
3	22	10
4	24	42

Table 1. Example of extracted runtime execution

Then we just need to run the Gnuplot to visualize the data. Here is example one of the Gnuplot command to visualize the data :

```
set title 'Time Execution Comparasion'
set xlabel 'JobID'
set ylabel 'Time'
set output "SWIM-Comparasion.png"
plot 'Result.txt' using 1:2 with lines, 'Result.txt' using 1:3 with lines
```

Then the result will be shown in SWIM-Comparasion.png

IV. Experiment Result

We have performed two experiments with the same workload using difference configuration of multi-server system. The first experiment we used four nodes configuration as you can see on figure 3, one node as the master and three nodes as the slaves. On the second experiment we used three nodes configuration as you can see on figure 4, one node as the master and two nodes as the slaves. Here is the result of the experiment

Job ID	Time Executed by Four Nodes System	Time Executed by Three Nodes System
0	20	25
1	21	24
2	20	25
3	23	25
4	20	25
5	33	33
6	20	25
7	19	24
8	37	26
9	19	24
10	20	24
11	33	45
12	20	24
13	19	24
14	19	23
15	19	24
16	19	25
17	24	28
18	32	25
19	22	43
20	41	68
21	63	76
22	60	79
23	63	88
24	59	60
25	60	53
26	19	24
27	19	23

28	20	24
29	19	23
30	20	26
31	41	47
32	35	44
33	34	43
34	41	66
35	20	29
36	21	48
37	62	52
38	37	60
39	84	110
40	109	128
41	130	202
42	20	50
43	20	88
44	19	21
45	20	30
46	22	33
47	20	33
48	23	41
49	20	26

Table 2. Extraction of runtime execution of four nodes and three nodes configuration

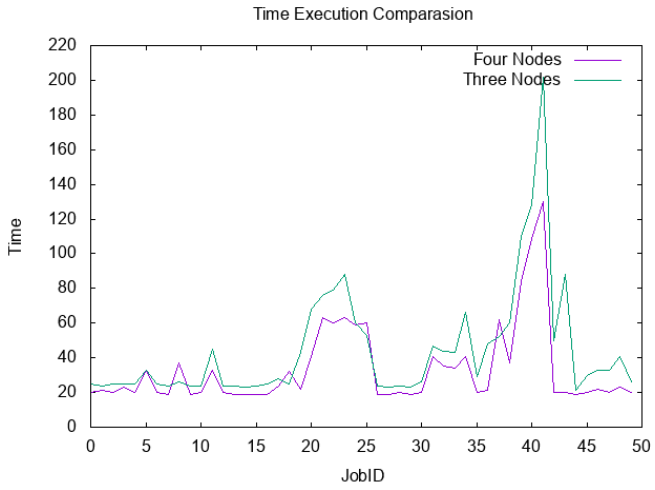

Fig 6. Line chart comparison runtime of four nodes and three nodes configuration

As you can see from the result above, four nodes configuration is faster than three nodes configuration. And obviously we all know about that, but what we want to know here is how is the comparison between both systems. As we can see above at the beginning of the job the difference is not too significant, but in the middle until the end of the job the more significant difference appeared.

From the observation above we can conclude the cause of that phenomenon. In the first part of job the difference is not too big because the slaves or resources to process the data still available, but in the middle while the four nodes configuration still have one more resource to use, they can run faster. This is happened because the four nodes got less waiting time for the execution of the workloads than the three nodes. The biggest difference is shown at the jobID number 41, because in this part the biggest byte of data is required. So the waiting time for resources is longer needed.

V. Discussion

We have performed two experiments and the results have been shown at the above section. We find out that two systems have a lot of difference and its vary depends on the parts of the job. Because more data means that we need more resources or nodes to handle it. So as long as the require data is not that big then both systems are not really have any difference in performance.

For further studies, other researcher also use the specification of the machine as variable to their experiment which is we don't use it in this experiment because of limited resources. Also some of them already found out some bugs on their experiment and tried to solve that problem using their own version of Hadoop or SWIM.

VI. Conclusion

In this paper we have presented a proposed method to choose the most suitable multi-server configuration for any enterprise that has big data in order to developing their business. This method is not perfect but still is the best option among the others. Because this method is free but give a proper accuracy. The result of experiment are :

1. Four nodes server is faster than three nodes server
2. The bigger the data, the bigger the difference.
3. The choice depends on the data that the enterprise using. If the data is not too big then it's better to choose the three nodes one because it use less cost but still can compromise with the data.

This experiment is just one of bunch variety of experiments. The author know that this experiment is not too relevant because if we use difference specification of server it will give us the more related to real world cases, because the choice of most cases in real world is influenced by the difference of server specification, not just the amount of the machine. The author just tried his best to do the experiment using limited resources (machines) that he could access.

This experiment is still can be improved by using another approach for example we tried using parallel system which are has a different topology from distributed system.

ACKNOWLEDGMENT

The author would like to thank God for His blessing so the author can finish this paper. The author would also like to thank Dr.Ir. Rinaldi Munir, MT., Dr. Dessi Puji Lestari, and Dr. Eng. Ayu Purwarianti, ST.,MT. as author's teacher on Socio-Informatics and Professionalism subject. The author also would to thank to Prof. Haryadi Gunawi and Riza Oktavian from University of Chicago for giving the author chance to study about distributed system under their upbringing. For the last, the author would like to thank everyone who supported us for making this paper.

REFERENCES

[1] Andrews, Gregory R, *Foundations of Multithreaded, Parallel, and Distributed Programming*, 2000.

[2] T. Williams and C. Kelley, *gnuplot 5.0 An Interactive Plotting Program,* 2004.

[3] C. Yanpei, A. Ganapathi, R. Griffith, and R. Katz, *The Case for Evaluating MapReduce Performance Using Workload Suites,* 2011

[4] Hadoop MapReduce. Retrieved May, 3 2017, from https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm

[5] Tom White, *Hadoop-The Definitive Guide, 4th Edition,* 2015

[6] Statistical Workload Injector for MapReduce (SWIM). Retrieved May, 3 2017, from https://github.com/SWIMProjectUCB/SWIM/wiki

STATEMENT

I hereby state that the paper I am writing is my own, not an adaptation, or a translation of someone else's paper, and not plagiarism.

Bandung, 5 May 2017

Praditya Raudi A / 13514087