|  |  |  |
| --- | --- | --- |
| **Module Code**  **COMP5850M** | School of Computing  University of Leeds  **Coursework 1 - Report** | University of Leeds logo |

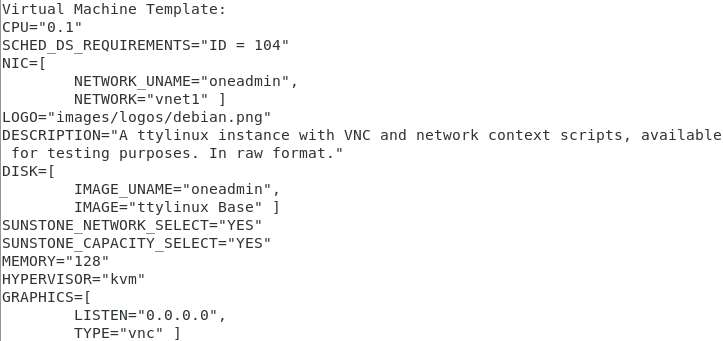
Full Name: Username:

Coursework Title: VIM Deadline Date: 09/03/2020

**Part 1: Java OpenNebula Cloud API (OCA) (10 marks)**

*Provide an explanation of the implementation of this task. The inclusion of the entire code is not required but you may include snippets if you wish.*

*VM template (1 mark)*



*Information OpenNebula provides about the VM (1 mark)*



*Measure the time it takes to instantiate/delete the VM. To get these measurements you are expected to run the experiments n times (e.g. n = 5). A statistical analysis (average, standard deviation) is expected. (2 marks)*

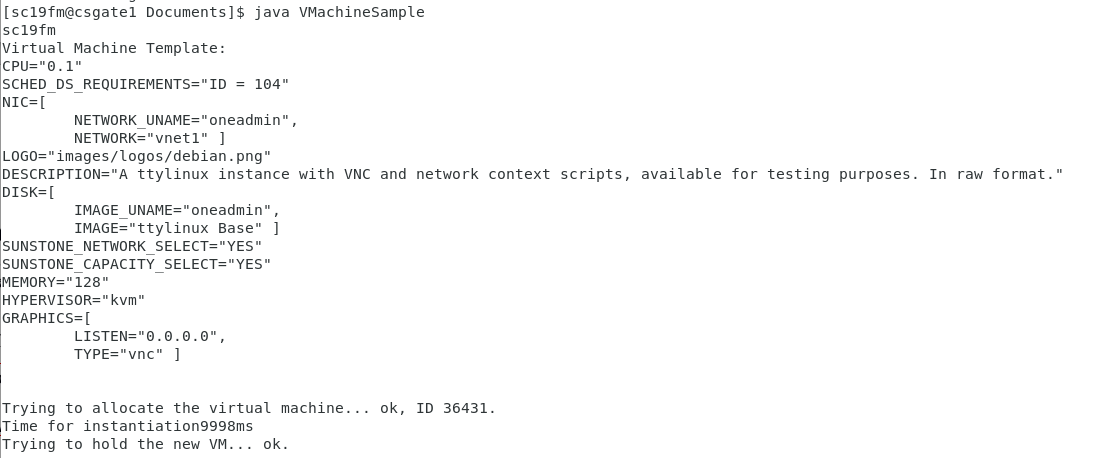
|  |  |  |
| --- | --- | --- |
| **Run No.** | **VM instantiation time** | **VM deletion time** |
| **1** | 8934 ms | 157 ms |
| **2** | 8486 ms | 128 ms |
| **3** | 9998 ms | 126 ms |
| **4** | 11203 ms | 135 ms |
| **5** | 8111 ms | 167 ms |
| **Average** | 9346.4 ms | 142.6 ms |
| **Standard Deviation** | 1123.22 | 16.43 |

*Explain how you have obtained these measurements (2 marks)*

Use currentTimeMillis() method to get the current system time (startTime) before instantiation a VM. Then I write a loop, keep checking the vm’s status. Until the vm is up running, break the loop. Then use the same method to get the time (endTime) after instantiation. And the instantiation time is the endTime minus the startTime.

Then use the same way to get the VM deletion time. Set startTimeDelete before delete. Set endTimeDelete after finalizeVM() method and make sure there is nothing wrong. Finally use these two numbers to calculate the deletion time.

*Evidence of successful run, e.g. screenshot (4 marks)*



****

**Part 2: VM Migration (15 marks)**

*Provide an explanation of the implementation of this task. The inclusion of the entire code is not required but you may include snippets if you wish.*

*Requirements (2 marks)*

Migrate existing VM to a better host. Considering the host’s load balance, CPU usage, memory usage and disk usage. Ideally, the host with lower load balance, CPU usage, etc should be better.

*Solution Design (2 marks)*

Check all of the hosts’ information in the host pool. Assign each factor with a weight. As far as I concerned, the number of VM shouldn’t be the main factor. Because some of the VMs require more CPU and memory, and some of them are not. The more important factor should be the CPU, memory and disk usage. So I assign 0.5 to the number of VM, assign 1 to other 3 factors.

As for these usage, I use the maximum usage divided by the current usage, then multiply 100 to represent these usage.

Finally, I add up the four parameters with each weights for each host. Then the host with lowest number is exactly the host I want the VM to migrate.

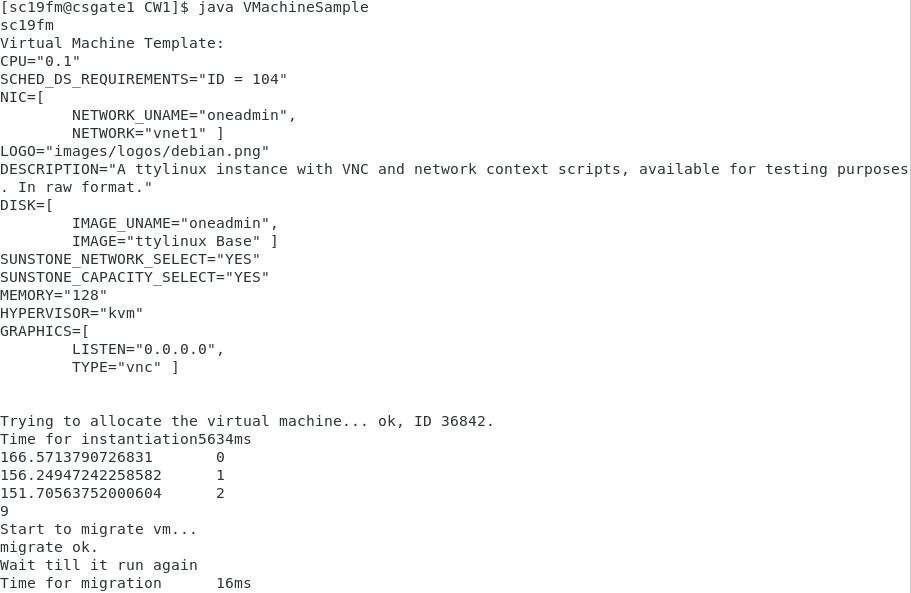
*Implementation (2 marks)*

Get the host id, CPU usage, memory usage and disk usage from the host pool. Then put each host with it’s properties into an array. Sum up these number with each weight. Then compare with each other, find the lowest and it’s host id. Live migrate the vm to the target host. Wait until the vm is up running again. Finally, check the target host information, make sure the vm is already in the target host.

*Measure the time it takes to migrate the VM. (2 marks)*

|  |  |
| --- | --- |
| **Run No.** | **VM migration time** |
| **1** | 15 ms |
| **2** | 24 ms |
| **3** | 21 ms |
| **4** | 19 ms |
| **5** | 16 ms |
| **Average** | 19 ms |
| **Standard Deviation** | 3.29 |

*Evidence of successful run, e.g. screenshot (3 marks)*

**

**

*Discussion of the results (4 marks)*

The host with more number of VM doesn’t mean it’s CPU and memory are occupied more. To consider if a host is suitable for migration, we should focus on it’s CPU, memory and disk usage. More basically, we should see the host’s status, whether it’s on or off, and it’s allocated CPU, memory is available or not. And if the hosts are in the same cluster, which means within the same LAN, it should take shorter time to migrate. I use live migrate the VM to a new host, which means transfer running vm between hosts without disconnecting the client or application. And this way only take a small time to migrate a VM.

*170.9930616450781*

*169.18557389709287*

*190.60436548827036*

*186.8285870777604*

*186.8285870777604*

*186.8285870777604*

*186.8285870777604*

*186.8285870777604*

*186.8285870777604*

*170.9930616450781 0*

*169.18557389709287 1*

*I'm gonna migrate VM to 7*

**Part 3: Resource Scaling and Performance/Energy Consumption Trade-Off**

**(10 - 25 marks, depending on application and challenge)**

*Details of the application considered (stress, MPI, Hadoop, other) (1-3 marks)*

Use hadoop data parallel processing job, run MapReduce application on up to 4 Vms and 4 hosts.

*Design of the experiments (1-4 marks)*

I design 6 experiment:

1. 1 MapReduce application run on 1 VM and 1 physical host
2. 1 MapReduce application run on 2 VMs and 1 physical host
3. 1 MapReduce application run on 2 VMs and 2 physical hosts
4. 1 MapReduce application run on 4 VMs and 1 physical hosts
5. 1 MapReduce application run on 4 VMs and 2 physical hosts (2 VMs for each host)
6. 1 MapReduce application run on 4 VMs and 4 physical hosts

Then use Zabbix-based monitoring infrastructure to keep an eye on the power consumption, CPU usage and runtime for each experiment.

*Implementation of the experiments (1-4 marks)*

Expand the .txt file size for the application to 167MB. (I did this by copy and paste the content of the book several times) In this way can make the application run for longer time, and it’s better for the obversation.

Follow the instruction, run the MapReduce application on a single node. Use “time” command record the runtime for a single node as well as the application start time and end time. Then go to Zabbix website, find the CPU usage and power consumption of the host within the period of time.

Then do the same thing for multi VMs and hosts. Create another VM and follow the instruction make it as the slave node. For different experiments, live migrate the VM to the same/different hosts with the master node. Then use the same method to record the CPU, power and runtime.

*Discussion of results (3-10 marks))*

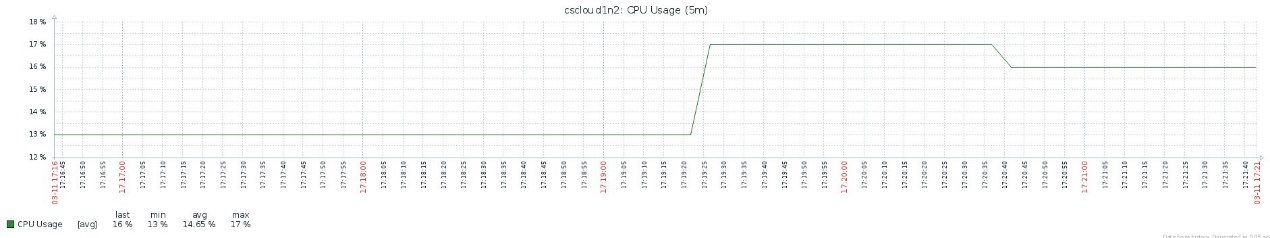
1. 1 MapReduce application run on 1 VM and 1 physical host



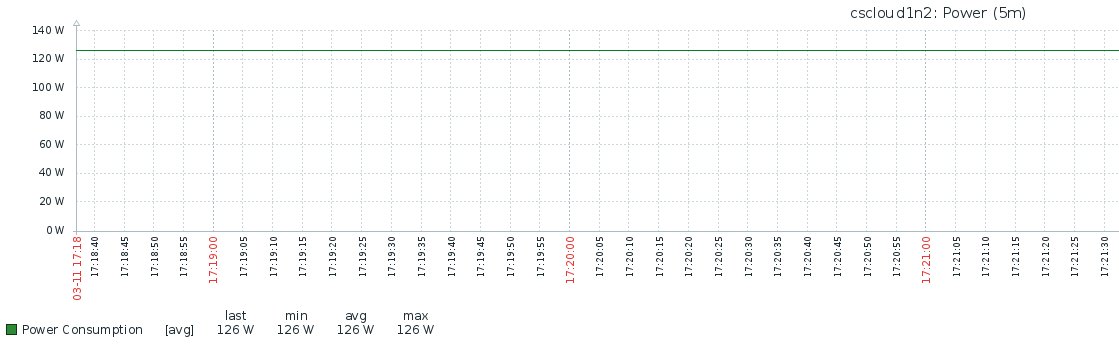
Runtime as below: 1:51.92



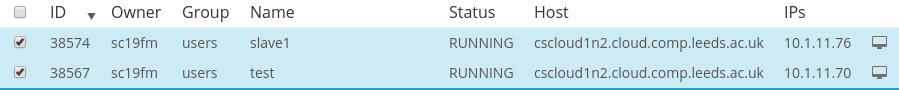
CPU usage for host cscloud1n2.cloud.comp.leeds.ac.uk



Power for host cscloud1n2.cloud.comp.leeds.ac.uk



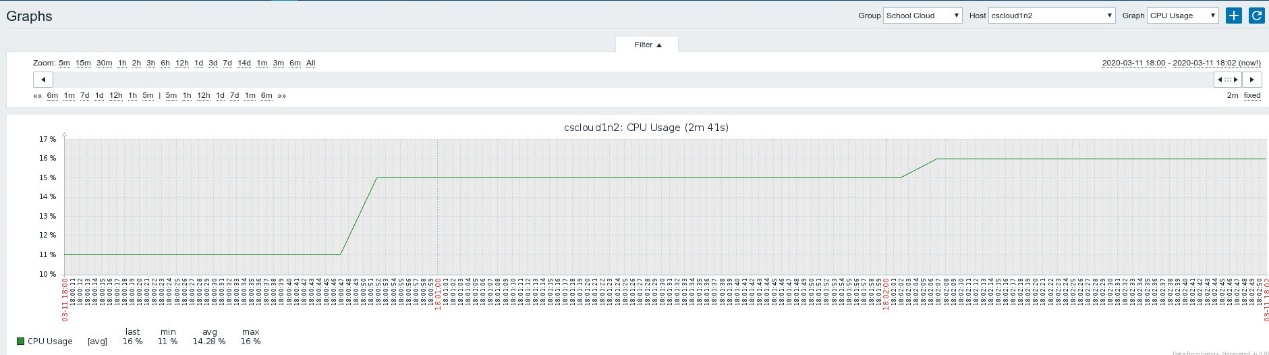
1. 1 MapReduce application run on 2 VMs and 1 physical host



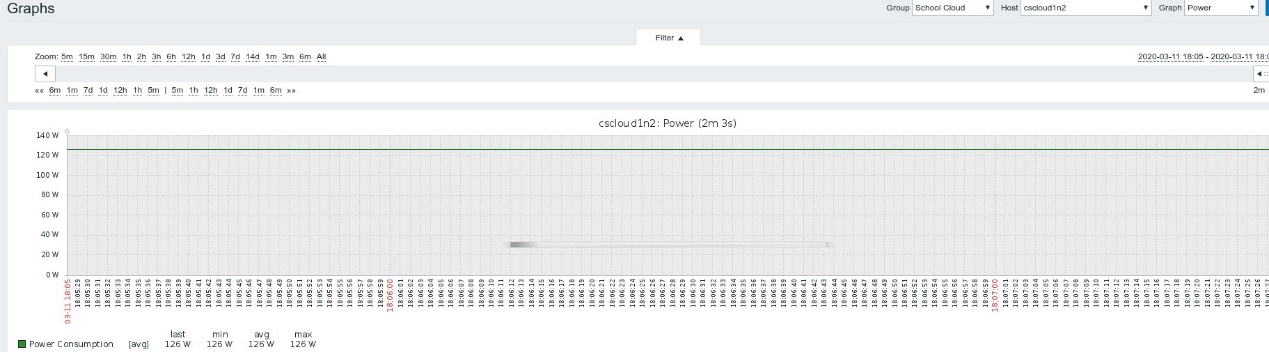
Runtime as below: 1:52.36



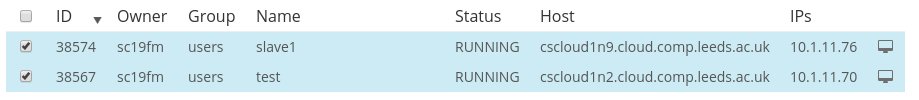
CPU usage for host cscloud1n2.cloud.comp.leeds.ac.uk



Power for host cscloud1n2.cloud.comp.leeds.ac.uk



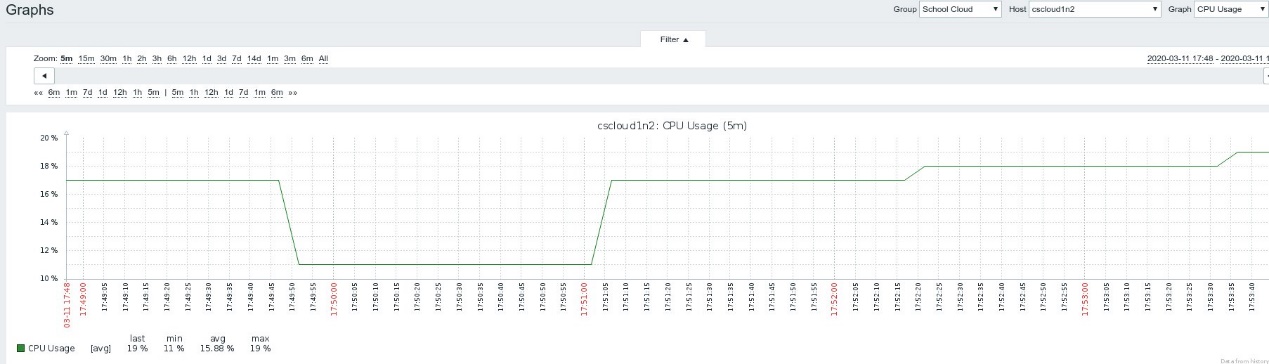
1. 1 MapReduce application run on 2 VMs and 2 physical hosts



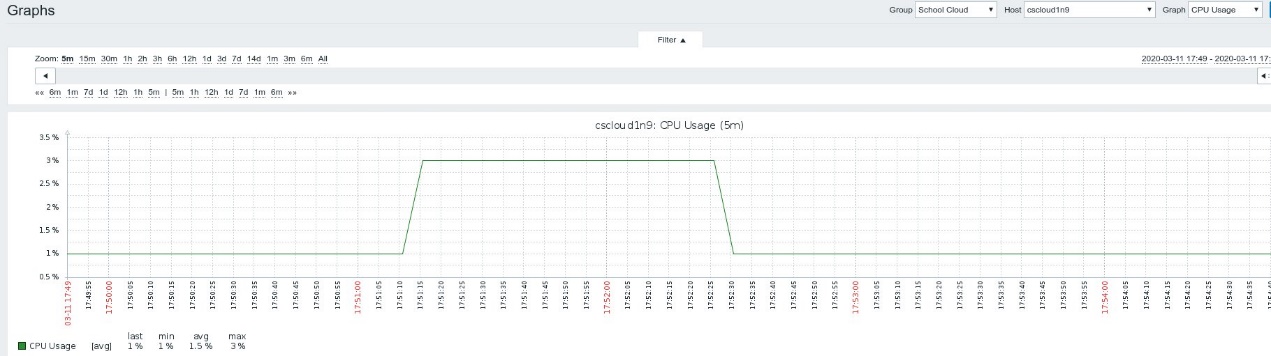
Runtime as below: 1:57.31



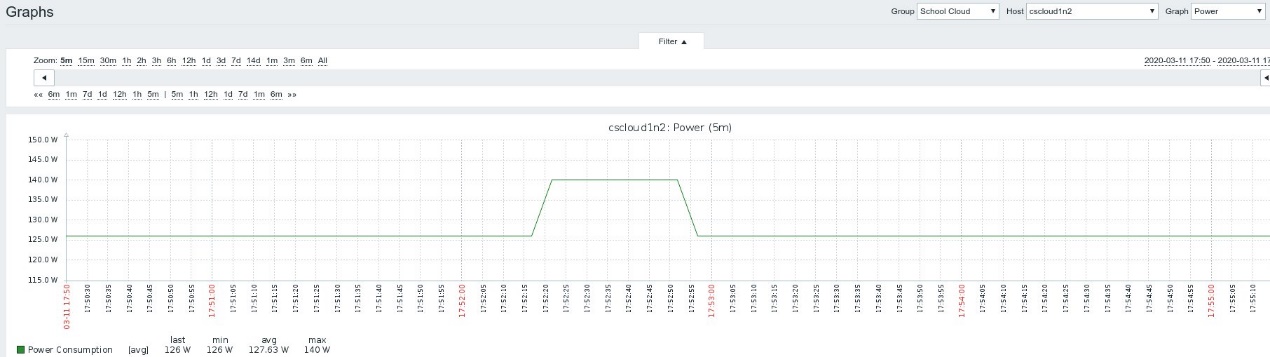
CPU usage for host cscloud1n2.cloud.comp.leeds.ac.uk



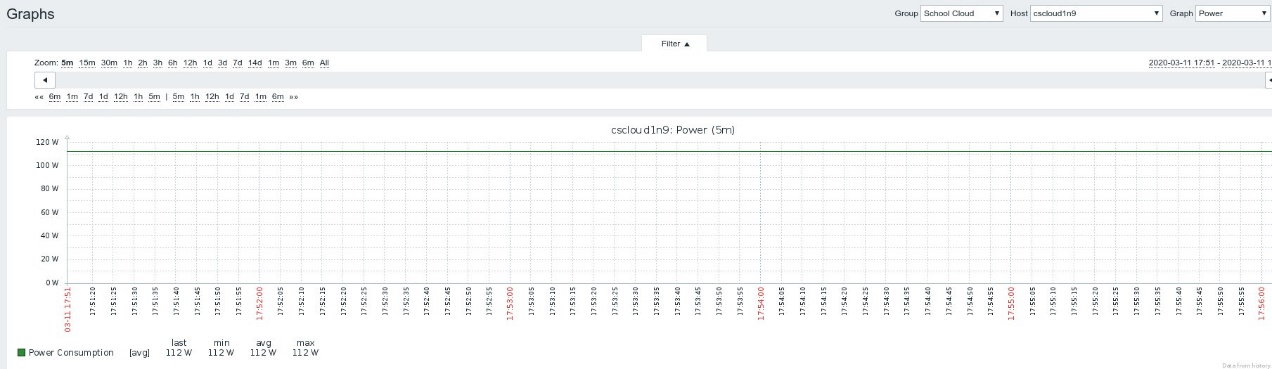
CPU usage for host cscloud1n9.cloud.comp.leeds.ac.uk



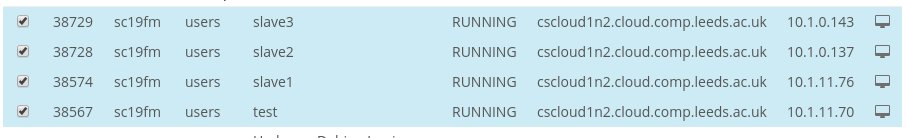
Power for host cscloud1n2.cloud.comp.leeds.ac.uk



Power for host cscloud1n9.cloud.comp.leeds.ac.uk



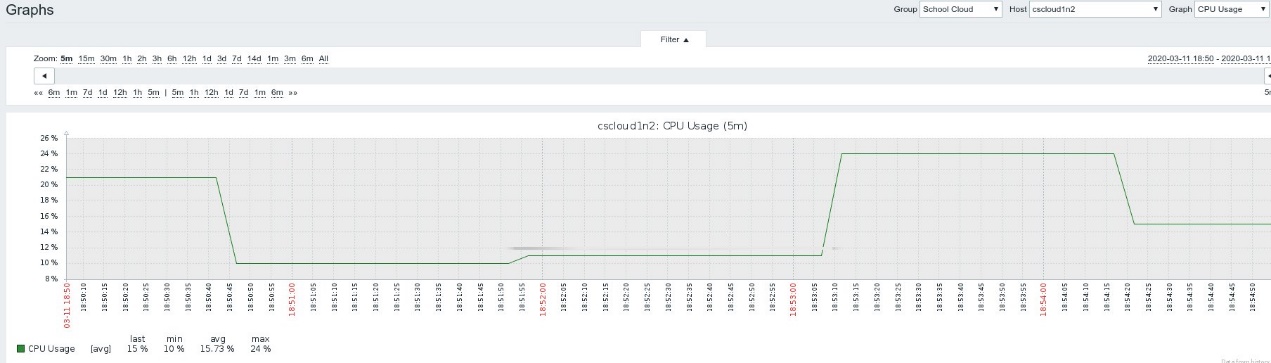
1. 1 MapReduce application run on 4 VMs and 1 physical hosts



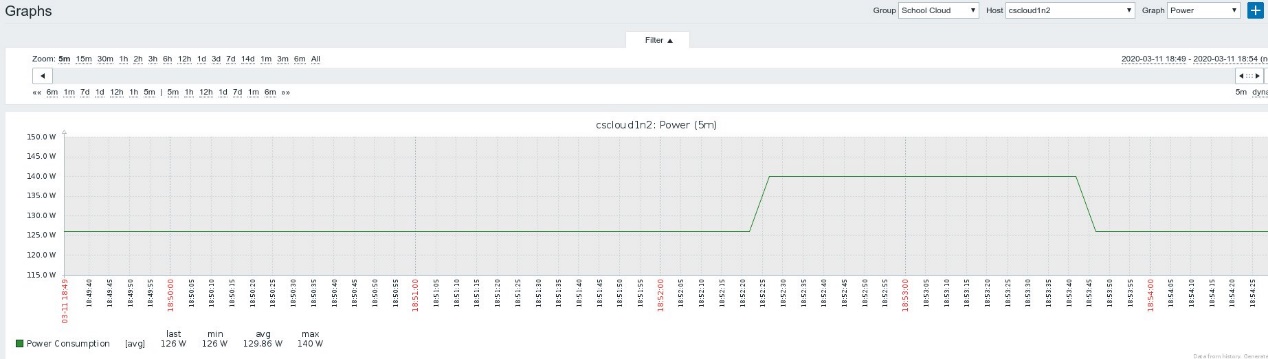
Runtime as below: 1:58.50



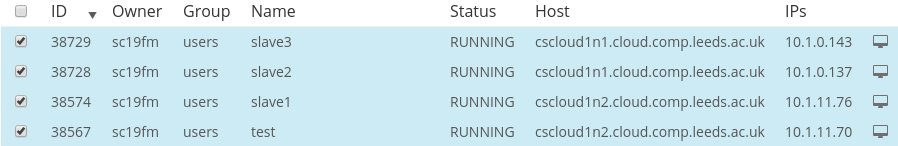
CPU usage for host cscloud1n2.cloud.comp.leeds.ac.uk



Power for host cscloud1n2.cloud.comp.leeds.ac.uk



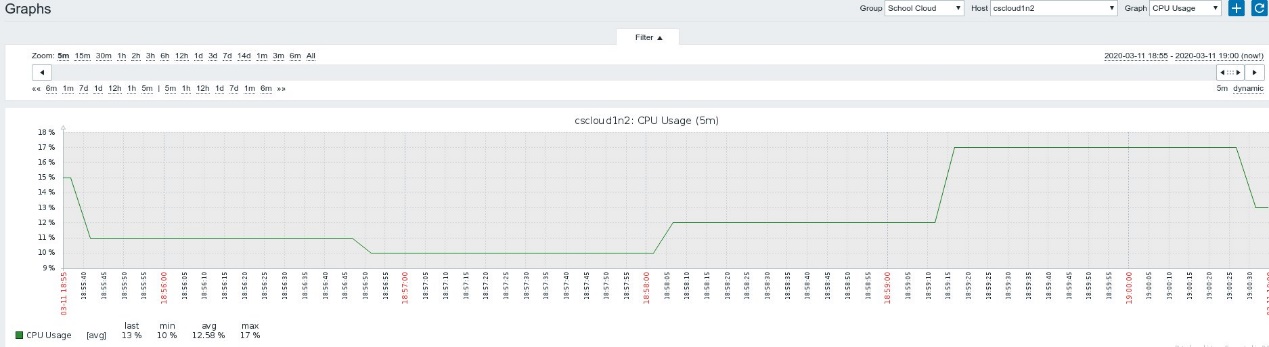
1. 1 MapReduce application run on 4 VMs and 2 physical hosts (2 VMs for each host)



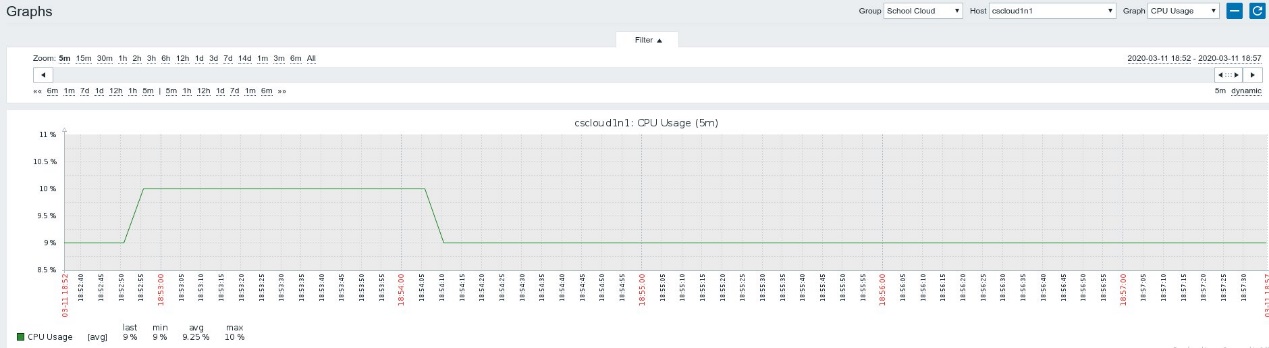
Runtime as below: 1:56.07



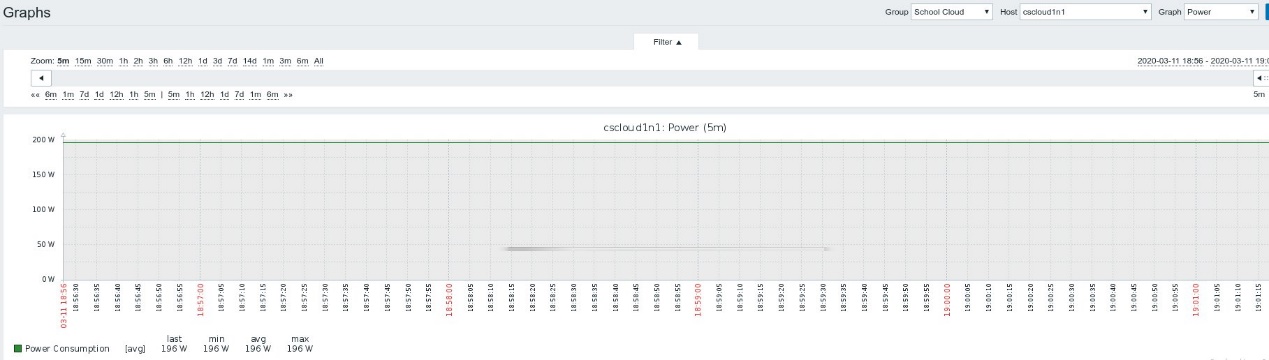
CPU usage for host cscloud1n2.cloud.comp.leeds.ac.uk



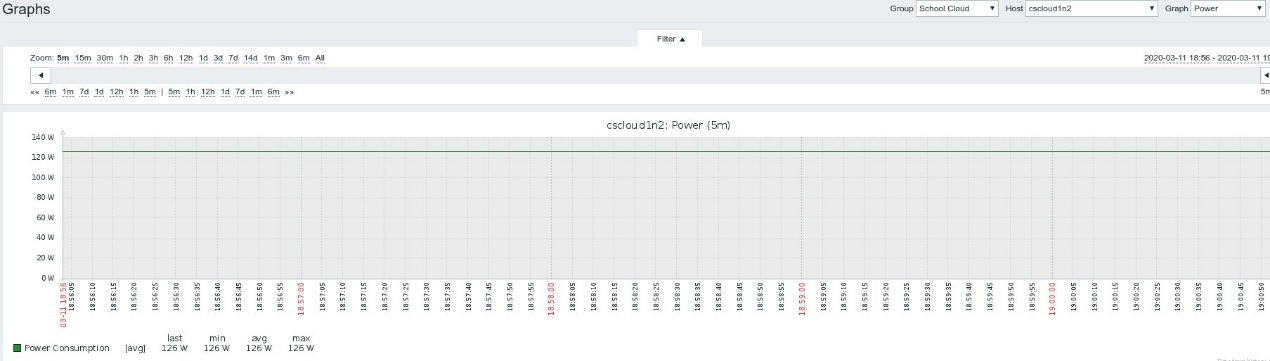
CPU usage for host cscloud1n1.cloud.comp.leeds.ac.uk



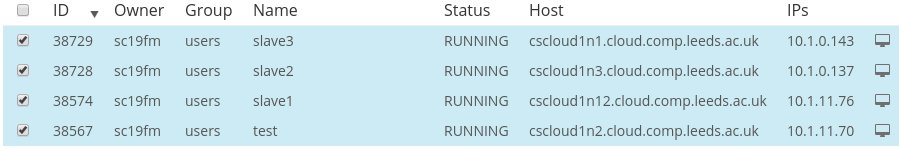
Power for host cscloud1n2.cloud.comp.leeds.ac.uk



Power for host cscloud1n1.cloud.comp.leeds.ac.uk



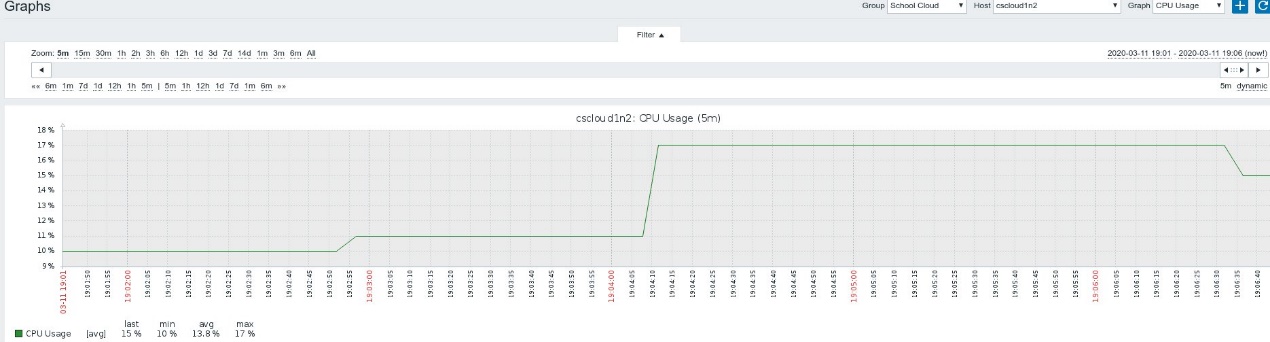
1. 1 MapReduce application run on 4 VMs and 4 physical hosts



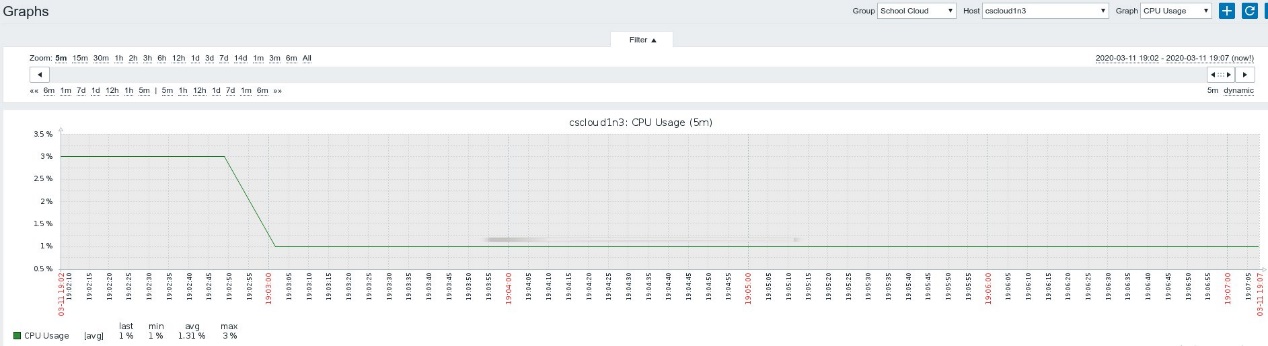
Runtime as below: 1:53.46



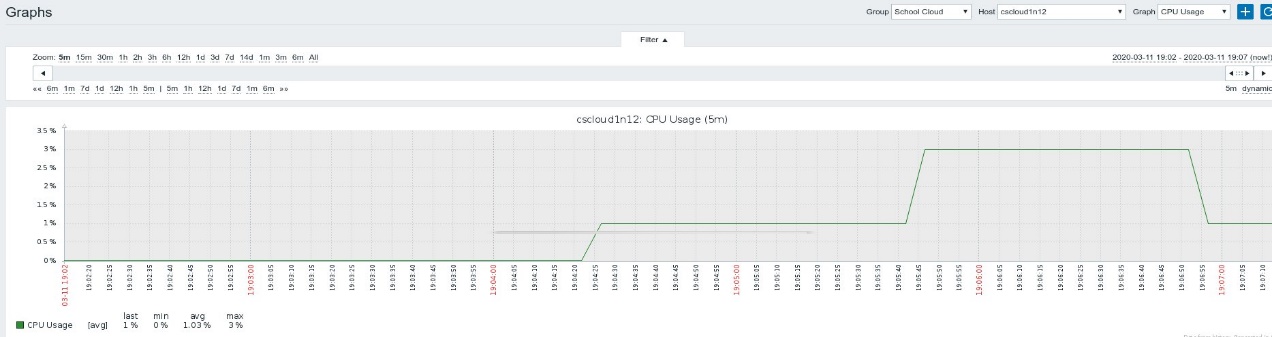
CPU usage for host cscloud1n2.cloud.comp.leeds.ac.uk



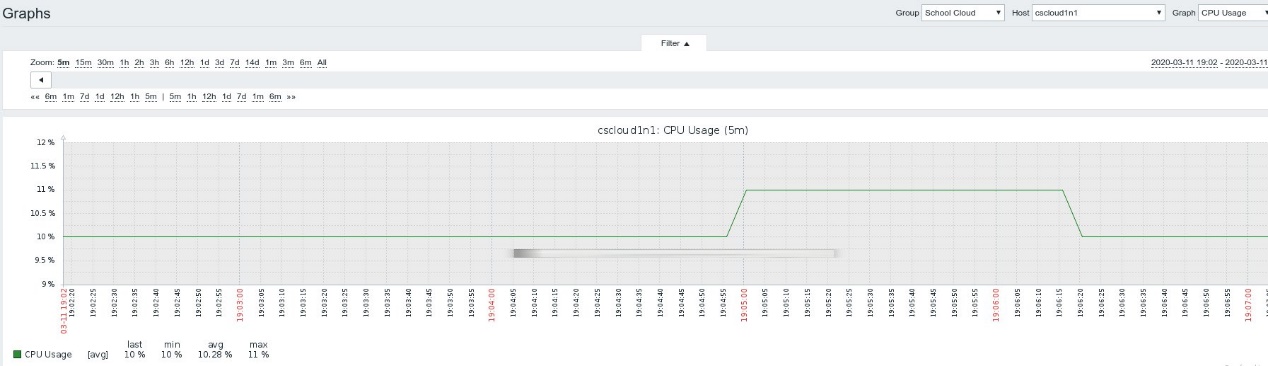
CPU usage for host cscloud1n3.cloud.comp.leeds.ac.uk



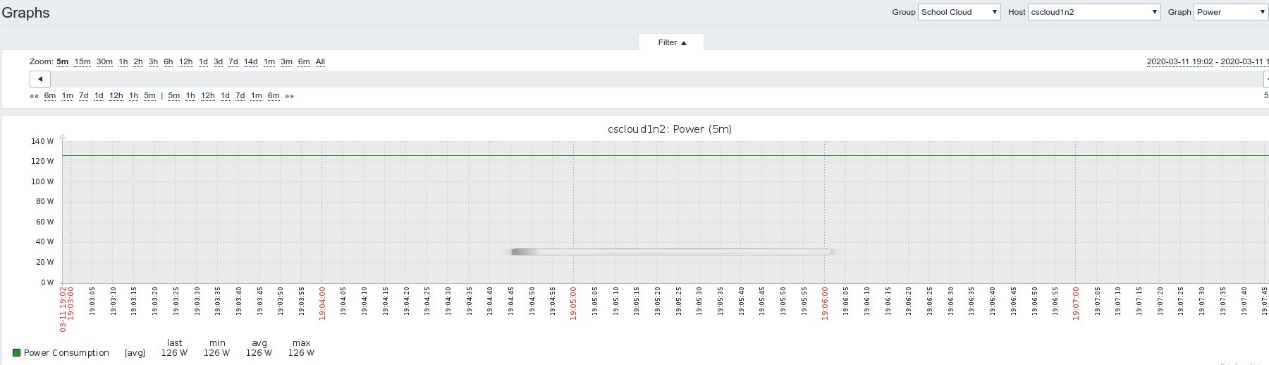
CPU usage for host cscloud1n12.cloud.comp.leeds.ac.uk



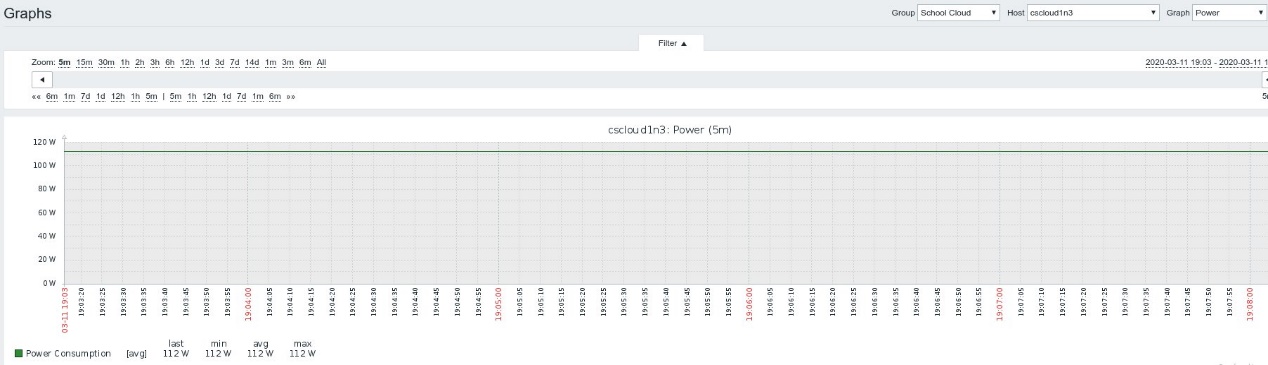
CPU usage for host cscloud1n1.cloud.comp.leeds.ac.uk



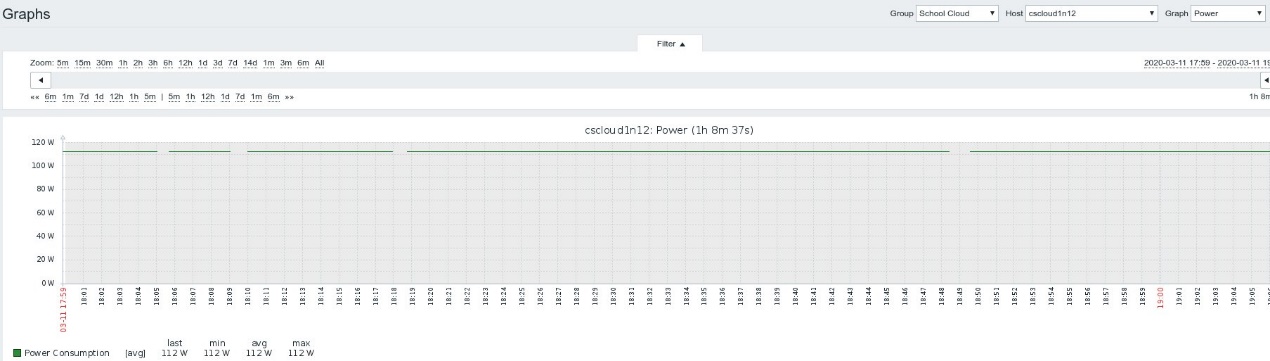
Power for host cscloud1n2.cloud.comp.leeds.ac.uk



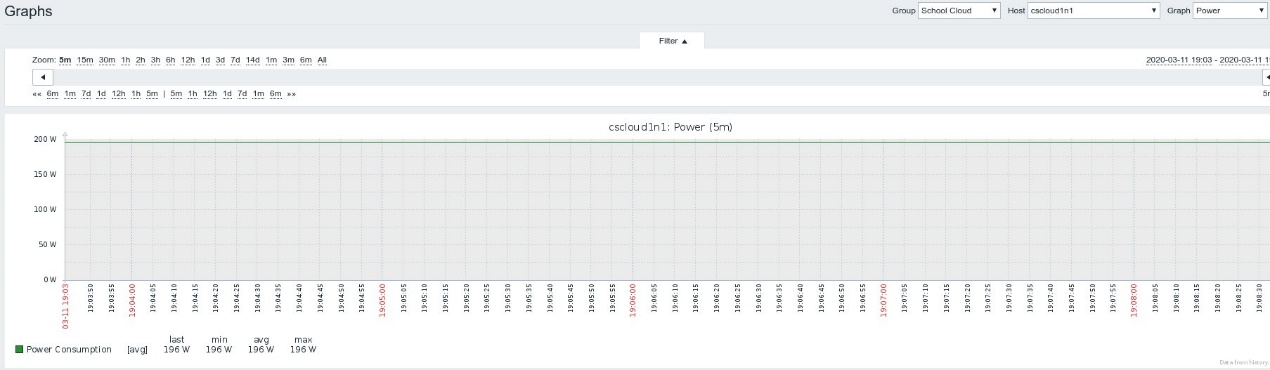
Power for host cscloud1n3.cloud.comp.leeds.ac.uk



Power for host cscloud1n12.cloud.comp.leeds.ac.uk



Power for host cscloud1n1.cloud.comp.leeds.ac.uk



As the screen shots show above.

For the runtime, different experiments only show small difference. And this is because VM are independent, they not be effected by other VM even if they are running on the same physcial host. But these VM have master and slave relationship, they can communicate with each other. And this communication can happen in LAN or WAN, it depends on if these VMs are run on the same host. Obviously, communication in LAN can be faster than WAN. And this could cause the different runtime.

For CPU usage, the usage increased 2 or 3 second after I input the execution command, this is probably because the network delay. With the same number of VM that run on different number of hosts, the CPU usage can be different. Clearly, if the VMs run on separate physcial machine, the CPU for each host is lower. And this is because hosts share the load balance. For example one VM for each host, rather than 4 VMs in the same host.

For power consumption, this is quite strange cause the power barely changes when I run the application. And this probably because there are about 80 VMs running on each host. And each time when running an application on one of the VM, this can be a farely small changes for the host. So the power is not likely to be affect by one small application. Another thinking is because the host’s CPU is not able for turbo boost, which means it cannot changes it’s frequency and always provide the it’s best performance.

*Evidence of successful run, e.g. screenshot (4 marks)*

**Any other comments:**

**记得把代码密码删了**

**记得测试**

**记得把代码分开，part1和2**

**还有截图output的**

2020-03-11 19:48:16,062 INFO mapred.LocalJobRunner: reduce > reduce

2020-03-11 19:48:16,063 INFO mapred.Task: Task 'attempt\_local1142230462\_0001\_r\_000000\_0' done.

2020-03-11 19:48:16,074 INFO mapred.Task: Final Counters for attempt\_local1142230462\_0001\_r\_000000\_0: Counters: 29

File System Counters

FILE: Number of bytes read=1041239444

FILE: Number of bytes written=1041732362

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=174700491

HDFS: Number of bytes written=546391

HDFS: Number of read operations=12

HDFS: Number of large read operations=0

HDFS: Number of write operations=3

Map-Reduce Framework

Combine input records=0

Combine output records=0

Reduce input groups=41992

Reduce shuffle bytes=347078276

Reduce input records=29448120

Reduce output records=41992

Spilled Records=29448120

Shuffled Maps =2

Failed Shuffles=0

Merged Map outputs=2

GC time elapsed (ms)=7

Total committed heap usage (bytes)=170020864

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

File Output Format Counters

Bytes Written=546391

2020-03-11 19:48:16,079 INFO mapred.LocalJobRunner: Finishing task: attempt\_local1142230462\_0001\_r\_000000\_0

2020-03-11 19:48:16,079 INFO mapred.LocalJobRunner: reduce task executor complete.

2020-03-11 19:48:16,384 INFO mapreduce.Job: map 100% reduce 100%

2020-03-11 19:48:16,385 INFO mapreduce.Job: Job job\_local1142230462\_0001 completed successfully

2020-03-11 19:48:17,710 INFO mapreduce.Job: Counters: 35

File System Counters

FILE: Number of bytes read=1654985926

FILE: Number of bytes written=2270202652

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=483622806

HDFS: Number of bytes written=546391

HDFS: Number of read operations=24

HDFS: Number of large read operations=0

HDFS: Number of write operations=5

Map-Reduce Framework

Map input records=3434860

Map output records=29448120

Map output bytes=288182024

Map output materialized bytes=347078276

Input split bytes=224

Combine input records=0

Combine output records=0

Reduce input groups=41992

Reduce shuffle bytes=347078276

Reduce input records=29448120

Reduce output records=41992

Spilled Records=88344360

Shuffled Maps =2

Failed Shuffles=0

Merged Map outputs=2

GC time elapsed (ms)=1014

Total committed heap usage (bytes)=462876672

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

File Input Format Counters

Bytes Read=174700491

File Output Format Counters

Bytes Written=546391

yarn jar OurMapReduceJob.jar org.myorg.WordCount hdfsbook.txt outputtt 104.18s user 6.97s system 87% cpu 2:06.57 total