PERFORMANCE EVALUATION OF A BIG DATA APPLICATION ON APACHE SPARK

by

Jeanne Alcantara

Bachelor of Engineering in Computer Engineering, Department of Electrical and Computer Engineering.
Ryerson University, Toronto, Canada, 2017

A project
presented to Ryerson University

in partial fulfillment of the requirements for the degree of
Master of Engineering
in the program of Electrical and Computer Engineering.

Toronto, Ontario, Canada, 2019

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF AN MRP

I hereby declare that I am the sole author of this MRP. This is a true copy of the MRP, including any required final revisions.

I authorize Ryerson University to lend this MRP to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this MRP by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my MRP may be made electronically available to the public.

ABSTRACT

Jeanne Alcantara

Master of Engineering

Electrical and Computer Engineering

Ryerson University, Toronto, Canada, 2019

Apache Spark enables a big data application—one that takes massive data as input and may produce massive data along its execution—to run in parallel on multiple nodes. Hence, for a big data application, performance is a vital issue. This project analyzes a WordCount application using Apache Spark, where the impact on the execution time and average utilization is assessed. To facilitate this assessment, the number of executor cores and the size of executor memory are varied across different sizes of data that the application has to process, and the different number of nodes in the cluster that the application runs on. It is concluded that different pairs (data size, number of nodes in the cluster) require different number of executor cores and different size of executor memory to obtain optimum results for execution time and average node utilization.

.

Acknowledgements

The author would like to greatly thank her supervisor, mentor and professor, Dr. Olivia Das, for her great enthusiasm, insight, guidance, motivation and most importantly, her inspiring the author to pursue a career related to the theme of the project all of which were key factors in bringing this project to the accomplishment stage. The overall experience has been a great one and the author has gained a lot of unforgettable knowledge. The author will remain grateful and will remember this experience going forward.

The author would also like to thank her fellow collaborator on the project, Hina Tariq, for whose help, guidance and great insight was contributive to bringing this project to fruition.

Finally, the author expresses deep thanks and gratitude to her family and friends for all their unwavering support and unshakeable faith in her throughout the course of the degree. It will never be forgotten.

Table of Contents

List of Tables

List of Figures

List of Appendices

List of Acronyms

1 AWS – Amazon Web Service

2 DAG – Directed Acyclic Graph

3 UI – User Interface

4 EMR – Elastic MapReduce

5 NF – Number of nodes/File size

6 EC – Executor cores

7 AU – Average utilization

8 ET – Execution Time

9 EM – Executor Memory

Chapter 1

Introduction

Apache Spark is an open-source distributed general-purpose cluster-computing framework. Henceforth we refer this framework interchangeably as either Apache Spark or Spark in this report. Spark is general-purpose in the sense that it allows running of applications from wide variety of domains such as machine learning, graph processing, data streaming; It is distributed in the sense that a Spark based application runs on multiple nodes; It is a cluster-computing framework in the sense that a cluster of nodes (that are distributed) are dedicated to run the same Spark application. Spark is increasingly being used to run Big Data applications. Several such applications exist today in the aforementioned domains [1]. This is because Spark enables a big data application—one that takes massive data as input and may produce massive data along its execution—to run in parallel on multiple nodes. Thus, for a big data application, performance is a critical issue.

Nguyen et al. concluded that performance of a big data application that runs on Spark is significantly impacted by two factors, one, the maximum amount of heap memory an executor process is allowed to consume while running on a node; second, the maximum number of cores of the node an executor process is allowed to utilize while running on a node [1]. This work complements the work of Nguyen et al. by considering two additional factors, one, the amount of data the application has to process, and the other, the number of nodes the application runs on. The choice of these two factors is intuitive since prior performance evaluation of traditional distributed software architectures conclude that while increase in the amount of workload leads

to increase in the application's execution time, increase in the number of nodes leads to the decrease in its execution time [2, 3, 4].

1.1 Motivation

The motivation behind this project was to analyze how the performance of a system—in terms of the execution time and average utilization, a metric that pertains to the average CPU consumption of all the nodes in the cluster—is affected by varying certain Spark configuration parameters such as the executor memory and the number of executor cores for a pair of number of nodes and input file size as noted in **Table 1**.

Table 1 *Number of nodes/File-size pairs*

Pair	**Number of nodes/File-size**
1	3/512MB
2	3/1GB
3	3/2GB
4	7/512MB
5	7/1GB
6	7/2GB

Spark is used to facilitate parallelization and processing of large amounts of data at high speed through Amazon Web Service (AWS). To this end, a WordCount application is executed, an application that counts the number of words in a document. To have an illustrative view of the system effects such as CPU utilization, memory usage, network load and, load at a node, Ganglia—a web-based performance monitoring system—is used. The change in application performance results through the alteration of two Spark configuration parameters.

1.2 Problem statement

One of the main problems in processing the big data is the toll on performance of the application that operates on the data. Evaluating which performance metrics take a greater effect on a system is hard to determine due to the multitude of configuration parameters involved in case of Spark[1]. Through this project, an analysis into the performance of an application is investigated.

1.3 Objective

By expounding on the problem statement and the aforementioned motivation, the objective of this project was to assess the magnitude of the toll on the performance of the application in terms of the execution time and average node utilization based on varying workloads. This was done while altering the number of executor cores and size of executor memory such that it yields lowest execution time for a certain workload (described as a pair as listed in **Table 1**).

1.4 Technologies used

Over the course of this project, the following technologies are used:

- Spark, an open-source platform used mainly for large dataset processing.
- Spark Web UI, an interface that provides directed acyclic graphs (DAGs) of the jobs performed, event timelines and the duration of each stage among many other metrics.
- Amazon Web Service (AWS) Elastic MapReduce (EMR) product, a tool used for big data processing and through which a Spark cluster was created and used in the implementation process.
- AWS S3 for storage.
- Ganglia, a performance monitoring application that provided the required graphs and performance metrics.
- WordCount, an application that counts the number of words in a document. and It is analyzed and evaluated in terms of performance changes.

1.5 Project organization

The organization of this project is as follows:

- Chapter 1 details a brief introduction to the subject matter, performance evaluation of big data applications using platforms such as Hadoop and Spark, followed by the problem definition, the motivation behind the project, the objective as a result of the motivation and a list of the resources used in the implementation process of this project.

- Chapter 2 details a brief literature review and a background into the implementation technologies of the project.
- Chapter 3 details the setup of the project, such as the set up of the environment and the default configurations and the incorporation and interpretation of the results obtained.
- Chapter 4 presents the discussion drawn as a result of the findings.
- Chapter 5 presents the conclusions and the future directions of the project.

Chapter 2

Related works

This section details a brief overview of existing approaches to performance evaluation of applications that use big data technologies such as Apache Hadoop and Apache Spark.

The work in [5] evaluates the performance of few log file analysis applications that were run on Hadoop as well as Spark. The log files analyzed were all cloud-based. The implementation included the usage of Spark, where the execution times, mean CPU usage, mean memory usage, mean network usage and mean disk usage for Spark and Hadoop were obtained by changing the number of nodes in a cluster and the memory of the cluster. The evaluation was also performed in different modes such as YARN cluster mode, YARN client mode and standalone mode. The results indicated that the that the increase of slave nodes implies a significant decrease in the execution time whereas increasing the input file led to an increase in the execution time, along with unexpected results such as higher network usage and disk usage, whereas Hadoop has higher processing times and mean utilization values. As a whole, it was noted that Spark had the best performance between the two frameworks.

In [6], big data workloads were analyzed using Hadoop, Spark and Flink to observe which framework outperforms the others in terms of performance and scalability. The big data workloads that were used in the proposal were WordCount, TeraSort, PageRank, Grep, Connected Components and K-Means. According to the results of the WordCount application, Spark was observed to have the best performance compared to Hadoop and Flink when the

number of nodes in a cluster was varied in terms of execution time and it was also observed that entities such as the network and input file size have the least impact on Spark, resulting in its superiority over Hadoop and Flink on the subject of general performance metrics.

In [7], big data workloads were used as inputs for application performance evaluation using Spark. An example of big data workload used was one that related to fraud detection. The impact of read/write latency on the performance of the application was analyzed. It was observed that the latency was impacted by the input file size and that the performance of the application can be negatively affected depending on the configuration of the system.

In [8], the performances of Hadoop and Spark for applications such as PageRank, WordCount, Sort, Naive Bayes, K-Means and TeraSort were compared. The results stated that, for the WordCount application, Spark had a speedup of 2.76 times but the performance dropped when the file grew larger, a phenomenon that was also demonstrated in the throughput. It was also noted that the CPU usage of Spark was better than that of Hadoop's. It was concluded that Spark is the optimal choice for applications that are iteration intensive and for machine learning applications and web searches. Another conclusion included the observation that a system must have adequate memory to run Spark, especially when the input file size is large.

In [9], the performance of a machine learning algorithm, Alternating Least Squares based Matrix Factorization, was assessed on a Spark cluster. Different configuration parameters—such as the number of spark executor cores as well as partition—were varied to analyze the impact on the performance of the algorithm.

The studies conducted above highlighted the speed, performance and efficiency of Spark in Big Data applications such as WordCount, TeraSort, Sort and PageRank and in several machine learning applications as well, including Naive Bayes and K-Means. The aforementioned studies

indicated that Spark has the best execution time and CPU usage statistics over all the other platforms.

This project deviates from the above works in a manner that relates to the investigation of the effect on the performance of a WordCount application under the changes of the executor core in Spark and the changes of the executor memory for different combinations of the number of nodes in a cluster and the file size as listed in **Table 1**, where the size of the text file to be used for the WordCount application is varied from 512MB, to 1GB and to 2GB for 3 nodes versus 7 nodes.

2.1 Background

This section entails a brief background on the technologies used in the project.

2.1.1 Hadoop

Hadoop is a medium consisting of different frameworks that supplies the ability to store immense amounts of data, remarkable processing power and the realization of parallelization [10]—an advantageous and highly sought after feature for data analytics, for big data applications in particular.

Hadoop consists of some modules that are categorized into the following [11]:

- Hadoop Distributed file System (HDFS), fault resilient distributed file system that makes servers scalable in terms of storage.

- YARN, a tool that allows both the resource management functionality and job scheduling.
- MapReduce, a tool provided by Hadoop that facilitates parallelization through the separation of a large set of data into chunks that run independently in a cluster or a set of clusters.

Along with the above, the following are components that can be run with Hadoop [11]:

- Spark, a big data processing tool.
- Hive, a data warehouse platform that allows the management, as well as the reading and writing, of large data volumes in distributed storage through the use of SQL [12].
- Oozie, a workflow scheduler tool that facilitates the management of Hadoop tasks [13].

2.1.1.1 Advantages of Hadoop

The usage of Hadoop incorporates the following benefits:

- Ability to process large amounts of data [14, pp. 437].
- The usage of Hadoop incorporates the added feature of parallelization and autonomy [15, pp. 2135].
- Failure resiliency, an advantage that is carried out from Hadoop's feature of duplicating the data sent to a specific node to all the other nodes in a cluster, effectively preventing overall failure [15, pp. 2135].

As a result of the advantages that Hadoop bring, Hadoop is used in IT industries, health care industries, telecommunications, higher education and computer software industries, to name a few.

2.1.1.2 Disadvantages of Hadoop

In spite of the advantages that Hadoop brings in—such as the ability to process large amounts of data at high speed and its resistance to failure due to data replication—Hadoop has the following disadvantages:

- The strong dependence of both the efficiency and scalability of a cluster on one name node [16].
- Security issues that arise from the involvement of big data when used in cloud computing [17].
- Presence of difficulties in the management and maintenance of relevant and crucial information due to the localization of this responsibility in one server in HDFS [16].

2.1.1.3 Architecture of Hadoop

A sample architecture of Hadoop is as shown below:

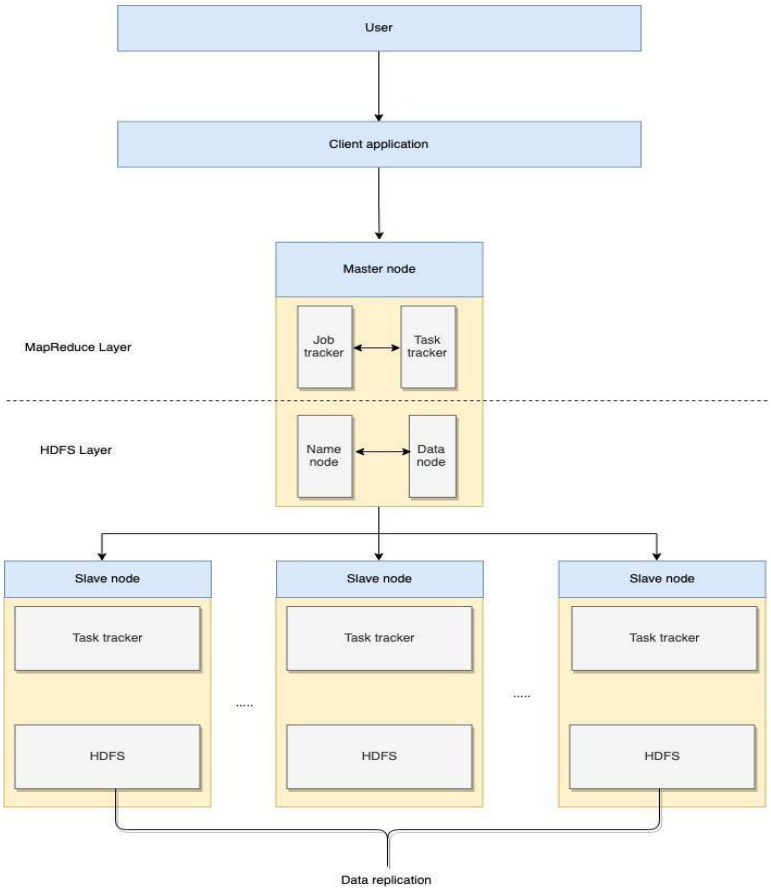


Figure 1 *Hadoop architecture*

The above figure, **Figure 1**, illustrates a sample architecture of Hadoop. As observed in the illustration, the Hadoop architecture comprises of a master node and any number of slave nodes. Additionally, two of the main components constitute the architecture of Hadoop, HDFS and MapReduce. A job tracker and task tracker module exist in the MapReduce layer of the master node, whereas a name node and data node lie in the HDFS layer of the architecture. Lying in the slave nodes in turn are components such as a task tracker and a data node. The task tracker in the architecture interacts with the job tracker and vice versa and is responsible for sending periodic progress reports to the job tracker module as well as doing tasks. The job tracker acts as a medium that allocates tasks to another active task tracker in the event that a task tracker module

fails. The name node in turn is tasked with managing HDFS files and to perform tasks related to MapReduce, such as splitting the data into chunks and storing the information, whereas the data node is the backbone of the data replication functionality, existing in all the nodes in the architecture, constituting the resiliency failure feature of the platform and additionally, the realization of parallelization due to the task tracker module that is present in all the nodes [18, pp. 50].

2.1.2 Spark

Like Hadoop, Spark is an open source platform that facilitates the processing and analytics of large amounts of data [19], where applications such as TeraSort, PageRank and WordCount can be used to expedite and optimize data processing. Expounding on this, Spark was designed for highly iterative operations on datasets and real-time analytics and is thus not tailored for applications that are incompliant with this design due to the possibility of resource wastage, for example, applications that require updates on a one by one basis such as web service storage are rendered unsuitable for Spark [20, pp. 8]. The design of Spark also allows batch processing, machine learning and the ability to run in Hadoop clusters, thus being able to process the data present in components such as HDFS, HBase, Hive and Cassandra to name several and to run on Hadoop. Other features of Spark include [19]:

- Spark's inherent 80 high-level operators facilitate the ability to build parallel applications and to use them through other platforms such as Java, Scala, SQL and R, rendering Spark easy to use.

- Apart from being able to run on Hadoop, Spark can run on Apache Mesos and Kubernetes, as well as access and process data in many data sources.

2.1.2.1 Advantages of Spark

The advantages of Spark include the following [21, pp. 172]:

- The data is stored in the memory, allowing for better and immediate access.
- The advantages of Hadoop, such as the fault tolerance feature and ability to process high data volumes due to the platform's build up from Hadoop.
- Higher efficiency as a result of the in-memory processing feature.

Due to Spark's two most notable features—the ability to process high amounts of data at a speed that is one hundred times faster than Hadoop, and to access data better resulting from Spark's in-memory processing feature—Spark is used in notable organizations and industries such as Amazon, eBay and Nokia for Big Data processing [22].

2.1.2.2 Disadvantages of Spark

The drawbacks of Spark are as listed below:

- Low-quality security in contrast to Hadoop [23, pp. 3].
- The requirement of a high amount of memory to support Spark's in-memory processing feature [24, pp. 10].
- Spark's usage complexity compared to Hadoop's MapReduce imposes a learning curve [23, pp. 3].

2.1.2.3 Architecture of Spark

A sample architecture of Spark is as shown below:

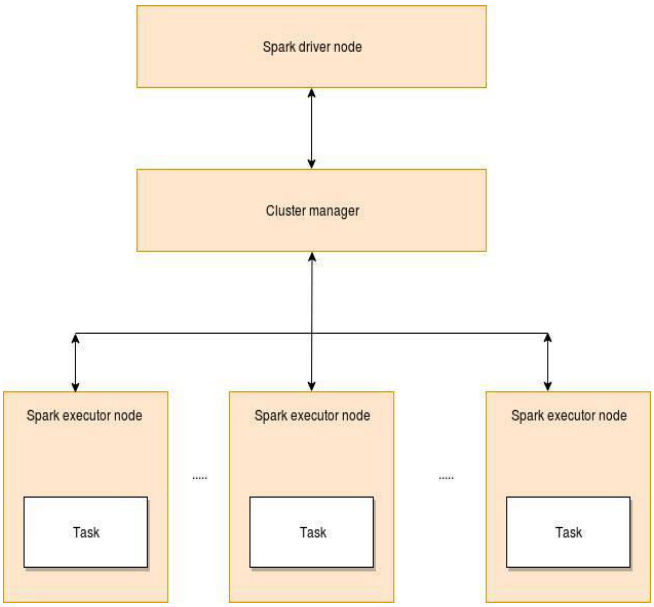


Figure 2 *Spark architecture.*

With respect to the architecture provided above, the architecture consists of components such as the driver node, or master node, a cluster manager module and any number of spark executor nodes, where data computations and storage occur [25]. The spark driver node consists of a component called SparkContext, which directly interacts with the cluster manager module and vice versa and is assigned the capability to establish a connection with other clusters, such as YARN, Mesos or a spark standalone cluster, through which tasks can be sent to the executor nodes [26]. The executor nodes, or worker nodes, also retain the ability to store the data in the cache for better, faster and immediate access, one of the most prominent features of Spark and a

contributing factor towards the platform's well-known processing speed that is superior to that of Hadoop's.

2.1.3 Ganglia

To have a graphical and numerical overview of the impacts on the performance of an application (for example, the impact of change in the number of executor cores), a software could be helpful. The software that was elected to fulfill such requirement in this project is Ganglia, a performance monitoring system targeted towards computing systems that operate at high performance in the form of clusters and grids [27]. Through ganglia, insights into the performance of node cluster when subjected to certain conditions (such as increased number of nodes within a cluster or an increased amount of data inputted into the system) can be visualized [28]. Additionally, statistics and performance metrics such as the number of CPUs, the average current load, the average utilization in the last hour and the number of hosts that are up and down, can be visualized as well. The above metrics can also be configured to be viewed for a specific node in a cluster within a preferred time period set by a user, thus rendering ganglia as a powerful and useful tool in analyzing the performance of a system.

2.1.3.1 Advantages of Ganglia

The advantages of ganglia are listed below [29]:

- The ability to customize the view, graphs and metrics to obtain certain desired metrics.

- The facilitation of studying the impact on a system's performance when subjected to increased load, leading to system improvement measures.
- The ability to view how busy a system is as a result of increased load or increased memory usage.

2.1.3.2 Challenges of Ganglia

Despite notable and helpful benefits that Ganglia brings in, there exist some limitations relating to ganglia. Some of these limitations are as listed below:

- The complexity of the network layouts that are able to adversely affect the distribution of information to all the workstations involved within a system (that complies to Ganglia's multicasting, one-to-many users, protocol), thus hindering the performance of ganglia as a whole.
- The heightened need for more memory as the complexity of a network increases to alleviate the above challenge.

2.1.3.3 Architecture of Ganglia

The architecture of Ganglia is as shown below:

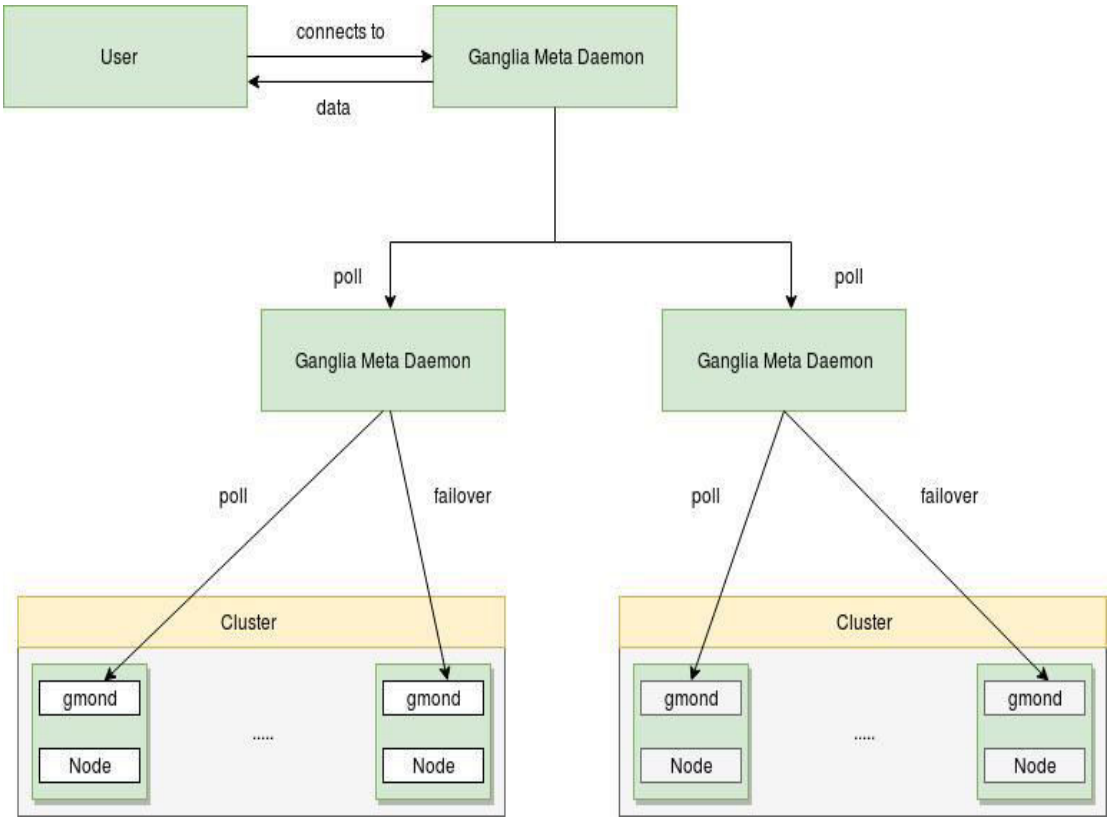


Figure 3 *Ganglia architecture.*

As shown above, a Ganglia architecture comprises of a Ganglia Meta Daemon component that the user directly interacts with, and vice versa, and an additional n number of Ganglia Meta Daemons below the aforementioned component for an n number of clusters. Each cluster comprises of a node and another Ganglia daemon, Ganglia Monitoring Daemon. The function of the Ganglia Meta Daemon, or gmetad, accumulates the data across all clusters involved and runs on the main server [29, pp. 14]. The gmetad component polls, an action that involves a component checking for information from the component it interacts with, the gmetad components that interact with an n number of clusters, which in turn also poll the nodes in the cluster. The other daemon, Ganglia Monitoring Daemon, gmond, is responsible for keeping

surveillance on a cluster and exists in all the nodes in all the clusters involved [30, pp. 822]. The architecture also accounts for a scenario where a node could potentially experience failure, thus prompting the gmetad component to designate a failover [29 pp. 74], or replacement, node in preparation for this scenario, a feature that resembles Hadoop's fault resiliency feature.

2.1.4 Amazon Web Service

The Amazon Web Service (AWS) is a service offered by Amazon to provide users the ability to perform secure cloud computing through features such as the variety of clusters, like Apache Spark for instance, the storage of databases and methods of delivering content to businesses and a multitude of users, facilitating the ability to build applications that have a higher flexibility, scalability and reliability, all of which are made available by paying for these services after registering for an AWS account. In the case for current students, students are able to make use of Amazon educate, a service that allocates $75.00 as credit in exchange for the students to be able to use the various products provided by AWS, such as S3 and FSx for storage and EMR and Athena for analytics purposes [31]. In addition, AWS offers users developer tools such as AWS CodeStar that assists in the creation and launching of AWS apps and machine learning tools such as Amazon Rekognition that analyzes video and images, among many other types of products offered by Amazon, to meet the many needs of users and businesses.

2.1.5 Spark Web UI

Spark additionally offers users the opportunity of viewing metrics that would be insightful regarding the evaluation of performance through its web UI component, an interface that provides DAGs of the task and additional details of a task such as the duration of a job, the active, completed and failed jobs, the number of stages, an event timeline of the task, the computing time of the executor and scheduler delay, along with additional details of the executors. By using the Spark Web UI, a more detailed and elaborate conclusion can be derived with respect to the performance of the application when subjected to certain conditions.

2.2 Chapter Summary

In summary, this chapter provided an insight into the previous works done that were related to the theme of the project–performance evaluation of applications processing big data—detailing the findings as well as the methods carried out in order to realize the findings for the purpose of giving a more detailed antithesis between the project and the works done previously. This chapter additionally provided a background into the technologies that were used in the implementation of the project and the architecture of these technologies for a graphical concept overview of how a specific implementation technology works.

Chapter 3

Experimental setup and Results

This section briefly details the experimental setup of the project.

3.1 Default configurations

Several default configurations were prepared in order to achieve certain metrics when some parameters were changed while one was kept constant. Table 2 detailing the default configurations is as shown below:

Table 2 *Default configuration*

Parameter	Default Value
Number of nodes	3
Executor memory	1GB
Input file size	512MB
Number of executor cores	1

3.2 Configuration variation

Using the table listed as **Table 1** as a reference, different configurations were set in order to better observe the effects each performance parameter has on the overall performance over a wider scale.

Table 3 *Configurations*

Parameter	Configurations
Number of nodes	7
Executor memory	2GB, 4GB, 8GB
Input file size	1GB, 2GB
Number of executor cores	2, 4, 8

With respect to the pairs listed in **Table 1**, a parameter from **Table 3** was varied to a certain value whereas the other parameters assumed a default value. For example, for the 3/512MB pair, the executor memory was kept at 1GB whereas the number of executors was changed from 1 to 2 or any of the other values listed in **Table 3**.

3.3 Cluster setup

The cluster that was used in the project was set up using AWS. The configurations of the cluster are as shown below:

Launch mode: Cluster

Software release: emr-5.23.0

Software applications configuration: Spark: Spark 2.4.0 on Hadoop 2.8.5 YARN with Ganglia 3.7.2 and Zeppelin 0.8.1

Hardware configuration instance type: m3.xlarge

Number of instances: 3

After the successful configuration of the cluster and once it starts running, a user can remotely access the cluster using an SSH client and a key pair for security and authentication purposes, at which point a myriad of commands related to submitting a spark job and uploading the output or data to the bucket in a user's AWS account are possible. In this fashion, the execution times were recorded for further observation and analysis. The other metric— the average utilization per configuration—on the other hand, was viewed on Ganglia, whose web UI was accessed through SSH.

3.4 WordCount algorithm in python

This project analyzed the changes in the performance of a WordCount application (running on Spark) when the Spark configuration parameters related to performance (number of executor cores and executor memory) were altered over various combinations of the number of nodes in a cluster, and the input file size. The algorithm for the WordCount application is shown in **Appendix A**, which was obtained from Github and was developed by the user, Aliga8or, using the Python programming language. The WordCount algorithm functioned by importing the add operator to facilitate summing and a library that allowed the configuration of several Spark parameters such as the executor memory for example. Following the necessary imports, the path of the text file in the cloud was specified denoted by the variable inputFile, in the case of this project, the S3 storage product provided by AWS, and was read. The lines in the text file were then read and split at the whitespace between each word, therefore commencing the counting process and the provision of the total number of words in the file. For instance, for the sentence "The quick brown fox jumped over the lazy dog", the sentence would be segmented at the whitespace, leading to "the", "quick", "brown" and so on until the end of the document or sentence is reached, leading to the generation of the total number of words, 8. Through this algorithm, the number of words in documents of different file sizes was generated.

3.5 Results

The results obtained from the different configurations are listed in this section.

3.5.1 Tables and sample graphs of the execution times and average utilization for different configurations

This section entails the results obtained from the variation of the above parameters listed in the default configurations table in **Tables 1 and 2**.

Table 4 shows the average utilization of the nodes (AU) and execution times of the application (ET) obtained for the pair NF=(# *of nodes* / *file size*) versus the number of executor cores (EC)[32], when the executor memory was kept at the default value of 1GB.

Table 4 *Executor core* (EC) *configuration results when the executor memory is kept at the default value of 1GB.* NF *implies the pair* (*# of nodes / file size*)*,* AU *implies average utilization,* ET *implies execution time.*

NF	3/512MB		3/1GB		3/2GB		7/512MB		7/1GB		7/2GB	
EC	AU (%)	ET (s)	AU (%)	ET (s)	AU (%)	ET (s)	AU (%)	ET (s)	AU (%)	ET (s)	AU (%)	ET (s)
1	10	190.894153	6	382.404513	8	773.103187	7	195.030082	3	374.135082	3	781.723696
2	9	189.201580	6	373.956062	8	744.711293	5	199.555822	3	388.913508	3	746.666616
4	8	193.448359	7	377.666209	8	751.089509	4	203.148644	3	384.977494	4	766.701263
8	7	187.431934	7	382.858586	7	750.406515	4	190.286180	3	386.646908	4	764.302005

Listed below is a graph of the execution times respective to the number of executor cores for each 3 nodes configuration.

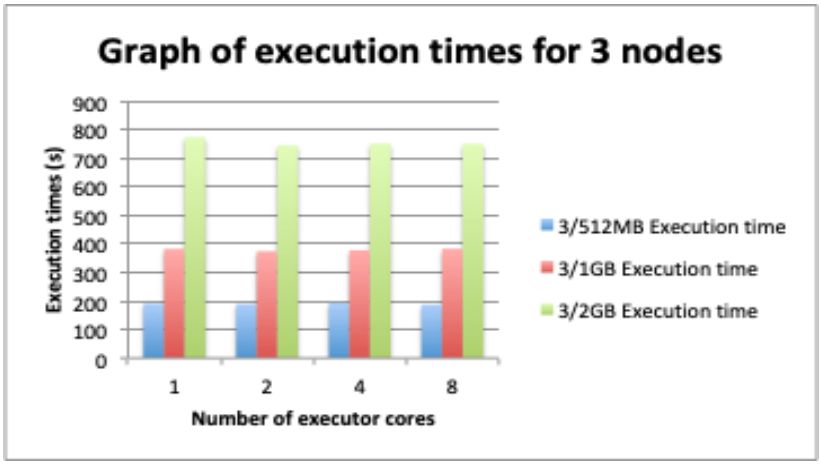


Figure 4 *Number of executor cores-Execution times graph for the 3 nodes configuration.*

Below is a graph of the utilization times corresponding to the number of executor cores for each 3 nodes configuration.

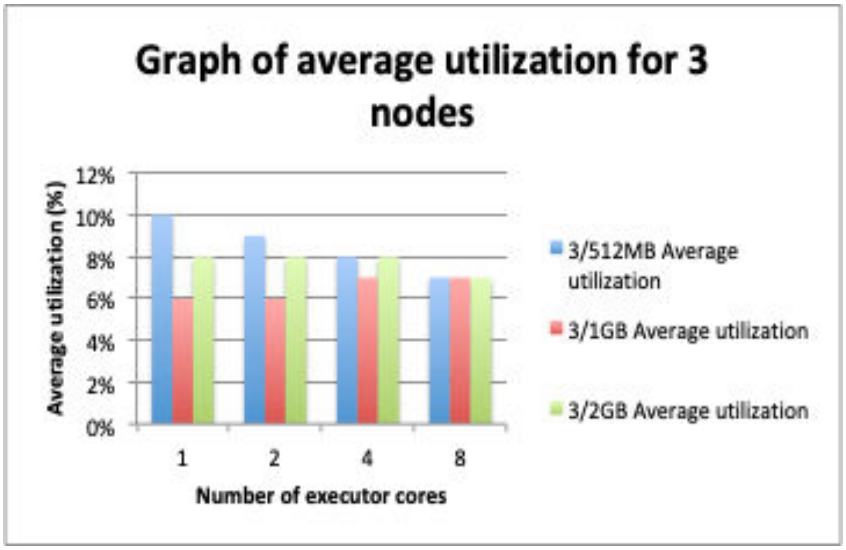


Figure 5 *Number of executor cores-Average utilization graph for the 3 nodes configuration*

Illustrated below is a graph of the execution times corresponding to the number of executor cores for each 7 nodes configuration.

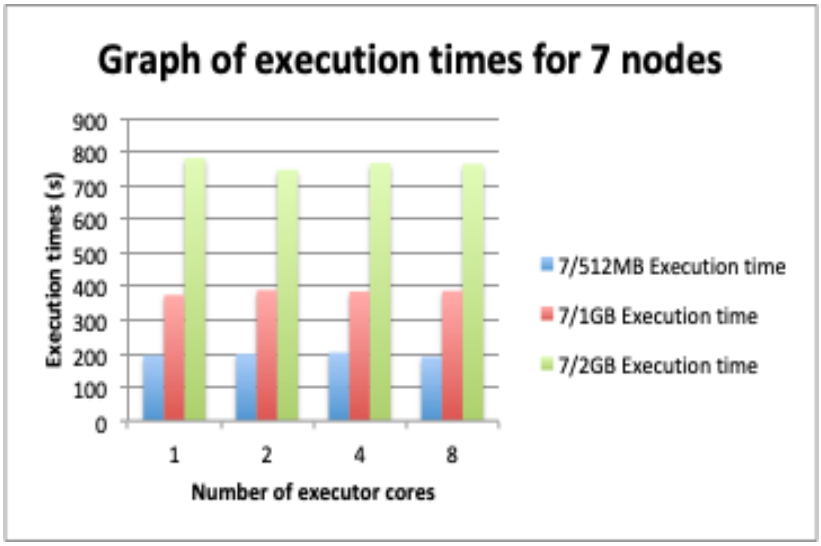


Figure 6 *Number of executor cores-Execution times graph for the 7 nodes configuration*

Below is a graph of the utilization times corresponding to the number of executor cores for each 7 nodes configuration.

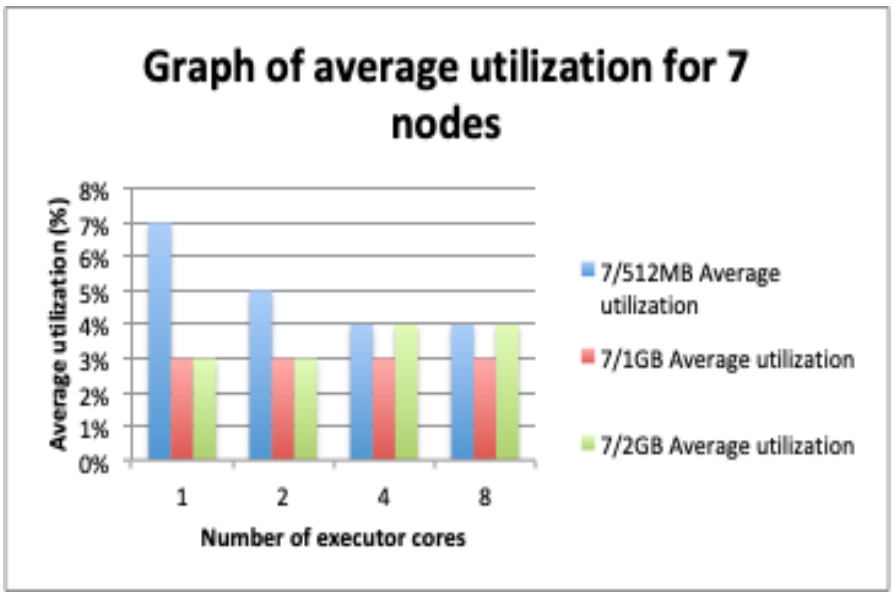


Figure 7 *Number of executor cores-Average utilization graph for the 7 nodes configuration*

With respect to the graphs obtained above, it was observed that the execution times of both the 3 and 7 nodes configurations are within the same range when the number of executor cores were changed incrementally (the executor memory being at the default configuration of 1GB). In spite of the slight increase in the execution time between 3/512MB and 7/512MB pairs when the number of executor cores is 1 (similarly for the rest of the pairs), there was a significant decrease in the average utilization as a result of the node increase, the lowest recorded average utilization being 4%. The result is depicted in **Table 4** as well as in **Figures 4 to 7.** Overall, while the execution time did not decrease, the average utilization had.

Table 5 shows the average utilization of the nodes (AU) and execution times of the application (ET) obtained for the pair NF=(# *of nodes* / *file size*) versus the executor memory (EM) [28], when the number of executor cores was kept at the default value of 1.

Table 5 *Executor memory* (EM) *configuration results when the number of executor cores is kept at the default value of* 1. NF *implies the pair* (*# of nodes / file size*), AU *implies average utilization,* ET *implies execution time.*

NF	3/512MB		3/1GB		3/2GB		7/512MB		7/1GB		7/2GB	
EM	AU (%)	ET (s)	AU (%)	ET (s)	AU (%)	ET (s)	AU (%)	ET (s)	AU (%)	ET (s)	AU (%)	ET (s)
1GB	12	197.759940	8	396.108658	7	756.927927	3	201.633905	3	387.120666	3	752.214988
2GB	8	193.504326	7	391.070779	8	752.478046	3	201.312989	3	391.149764	3	747.853411
4GB	7	192.727274	8	385.200333	8	763.991094	3	189.851224	3	383.684466	3	751.057607
8GB	7	189.624714	8	374.192056	8	752.089426	2	189.074740	3	382.410938	3	748.539087

Listed below is a graph of the execution times respective to the executor memory for each 3 nodes configuration.

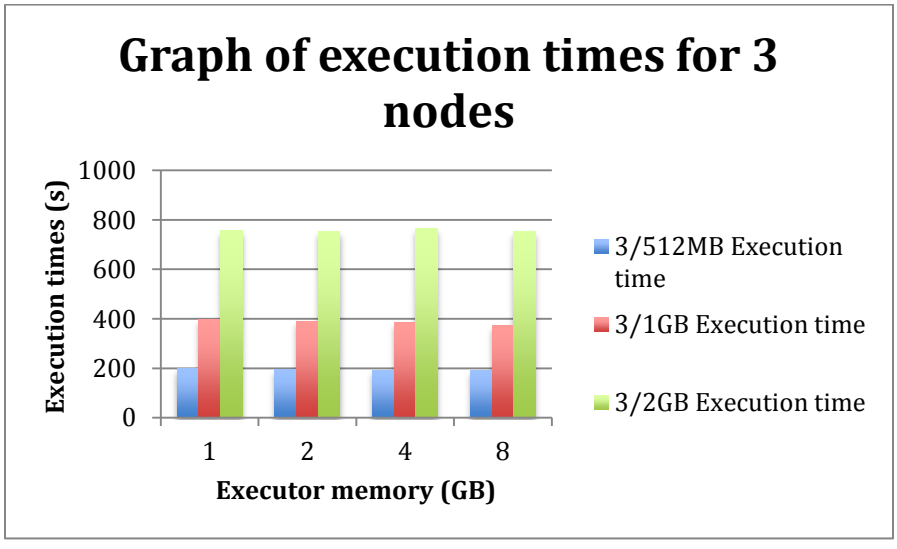


Figure 8 *Executor memory-Execution times graph for the 3 nodes configuration.*

Below is a graph of the average utilization respective to the executor memory for each 3 nodes configuration.

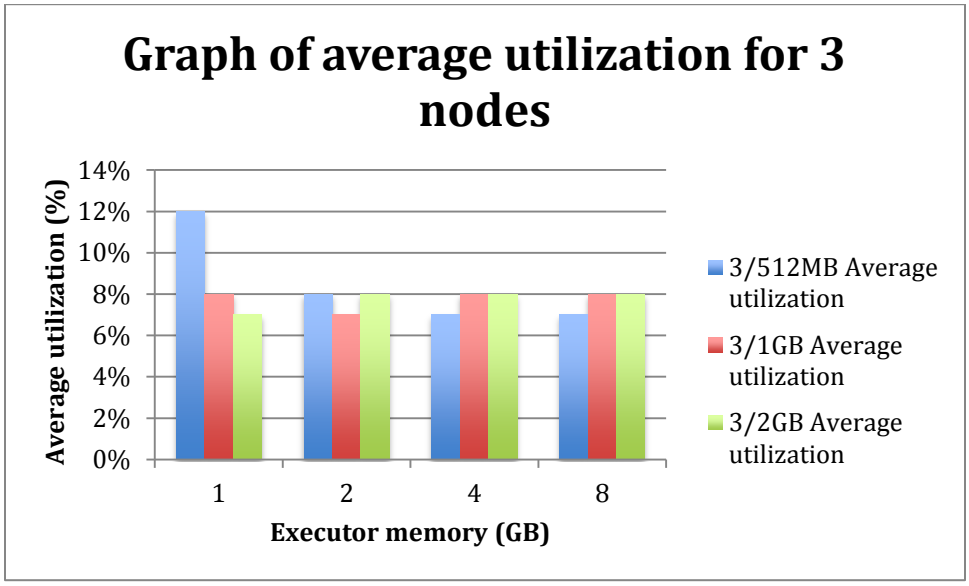


Figure 9 *Executor memory-Average utilization graph for the 3 nodes configuration*

Listed below is a graph of the execution times respective to the executor memory for each 7 nodes configuration.

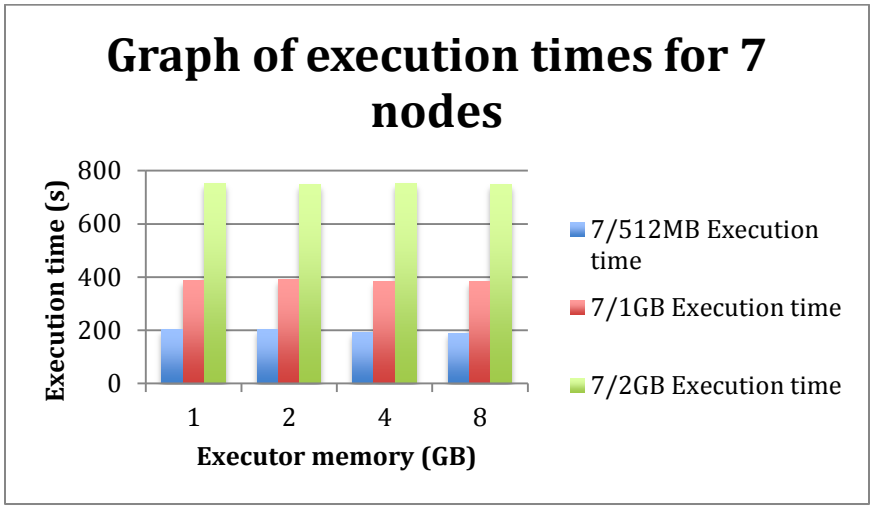


Figure 10 *Executor memory-Execution times graph for the 7 nodes configuration*

Below is a graph of the average utilization respective to the executor memory for each 7 nodes configuration.

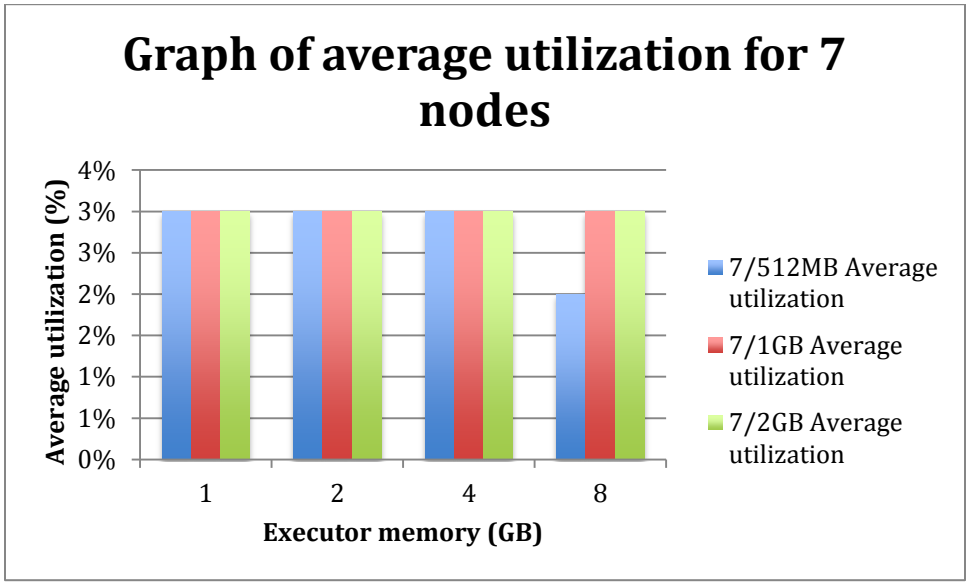


Figure 11 *Executor memory-Average utilization graph for the 7 nodes configuration*

As shown in **Figures 8 to 11**, it was observed that, in contrast with the results obtained from **Table 4**, the average utilization for the 7 nodes configuration dropped to 2%. Expounding on that phenomenon, the average utilization between **Table 4** and **Table 5** demonstrates a drop when the executor memory was varied (keeping the executor core at a value of 1). Additionally, the execution times for each of the 3 nodes pairs are within the same range as tabulated in **Table 5** (similarly for 7 nodes pairs), with fluctuations present. For example, for the 7/2GB configuration in **Table 5**, the execution time increased when the executor memory was 2GB compared to when the number of executor cores was 2 in **Table 4**, whereas for the rest, the execution time and average utilization decreased or remained constant respectively.

Overall, the above exhibited the observation that, while increasing the executor memory per configuration does certainly decrease the average utilization, the execution time can increase, further implying a tradeoff between the two measures of performance and further research into achieving goals of twofold— to decrease both the execution time and average utilization per configuration.

3.5.2 Summarization of the results for the 3 nodes configuration

Section 3.5.2.1 details the sample graphs obtained from Ganglia for the 3/512MB pair in **Table 4** and a summarization of the results obtained from **Table 4** for the rest of the configurations. Section 3.5.2.2 similarly deals with an encapsulation of the findings achieved in **Table 5**.

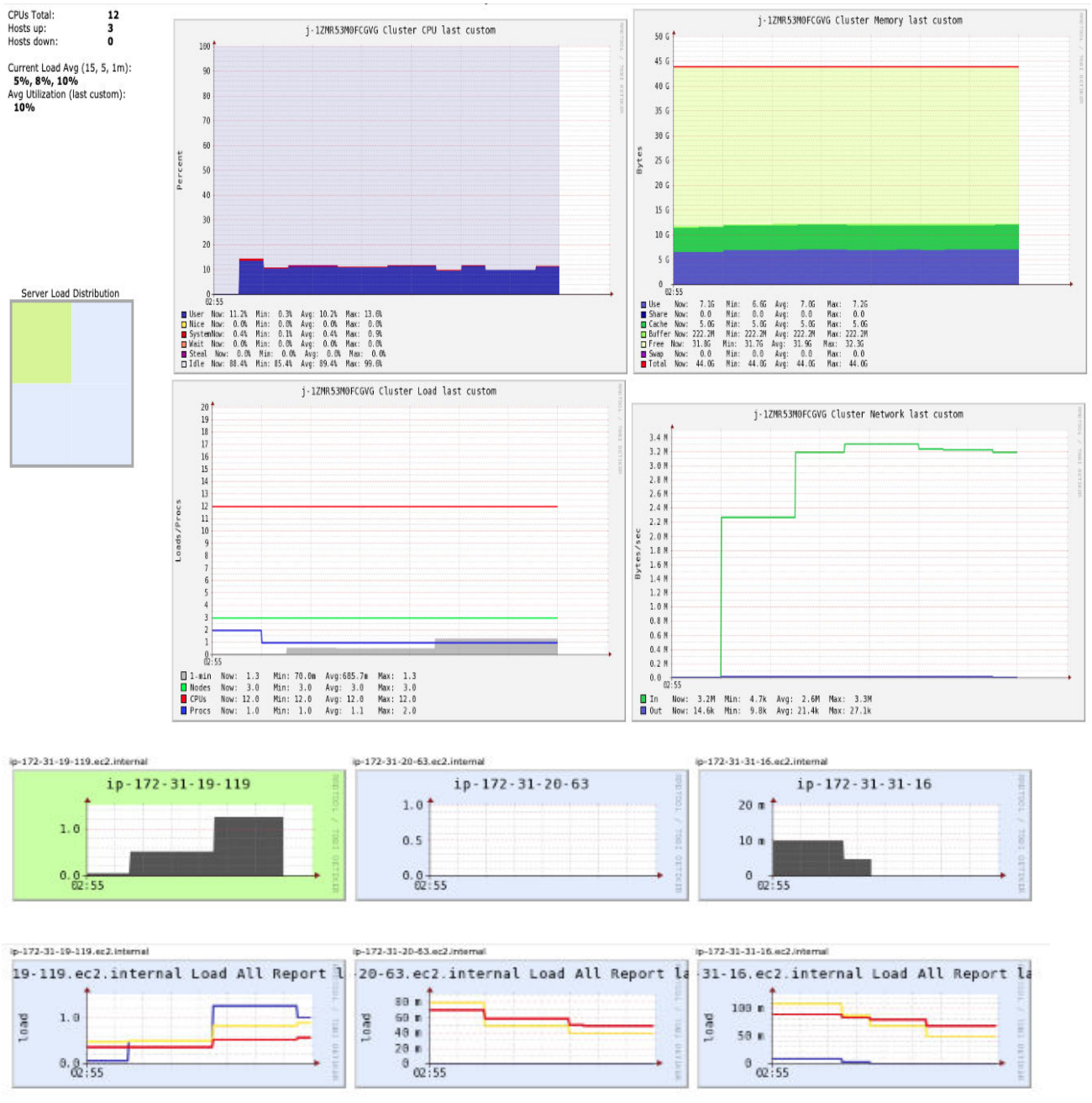
3.5.2.1 Executor cores

The sample graphs listed in this section correspond to the results of the pair, 3/512MB, in **Table 4**. The graphs in this section consist of graphs from two views - cluster view and master node view. For the cluster view, the average utilization along with the number of hosts that are up and down are listed as well as several graphs such as the load per node in the last hour, the one load statistic per node, the CPU usage in the last hour, the memory usage, the cluster load usage and the network load. On the other hand, the master node graphs list the CPU usage details and the load details.

3/512MB

For 1 executor core

Cluster view



Master node view

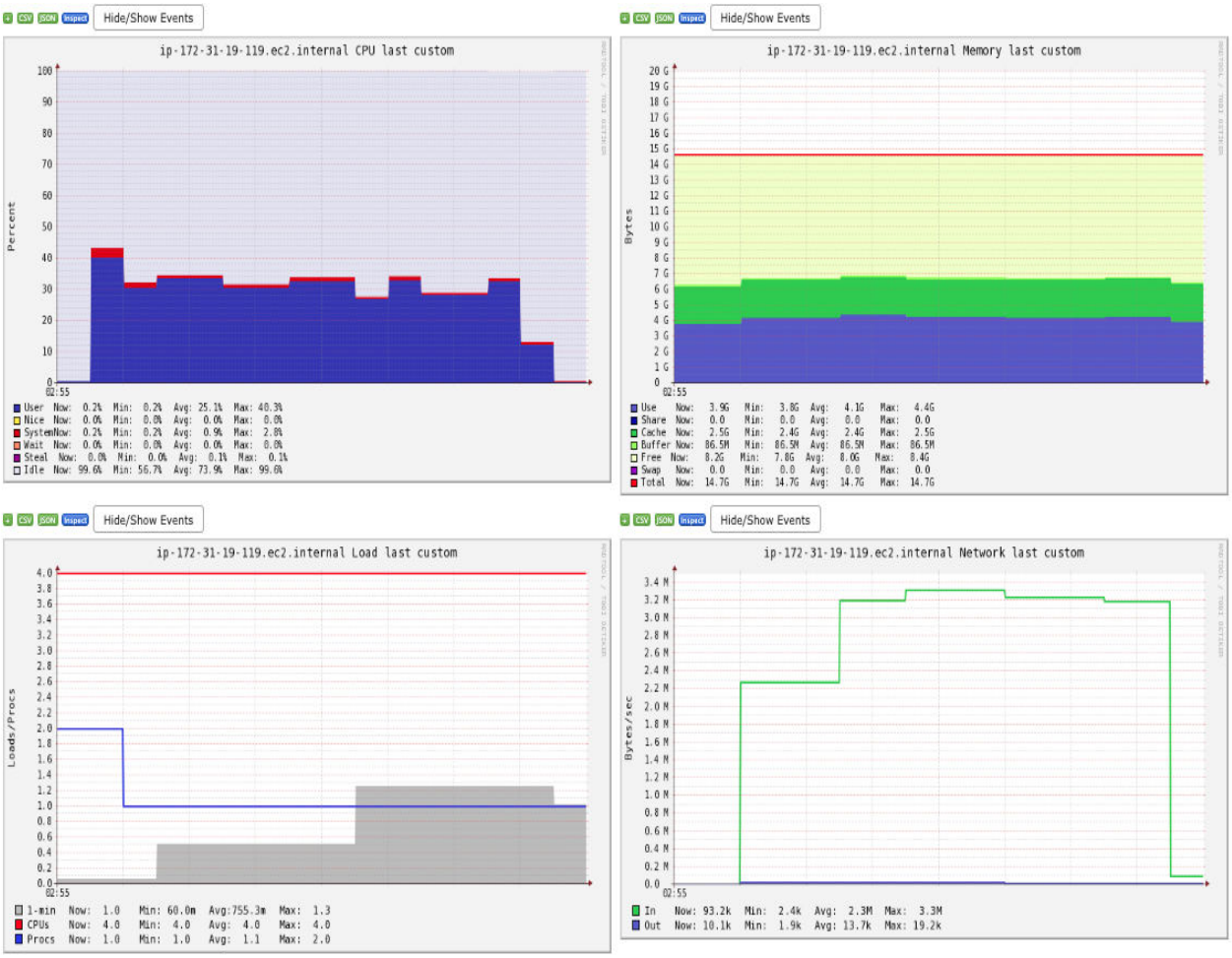
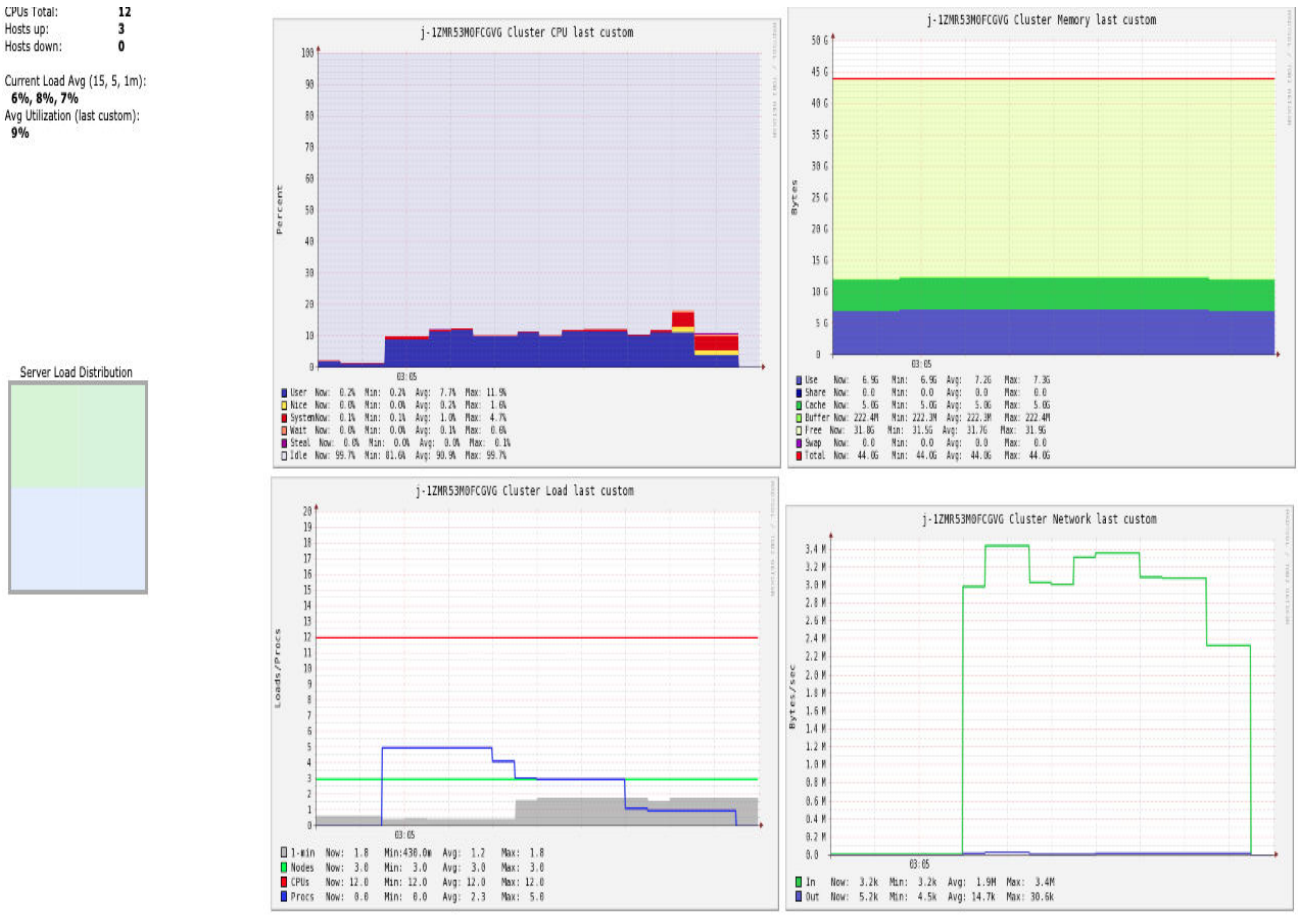
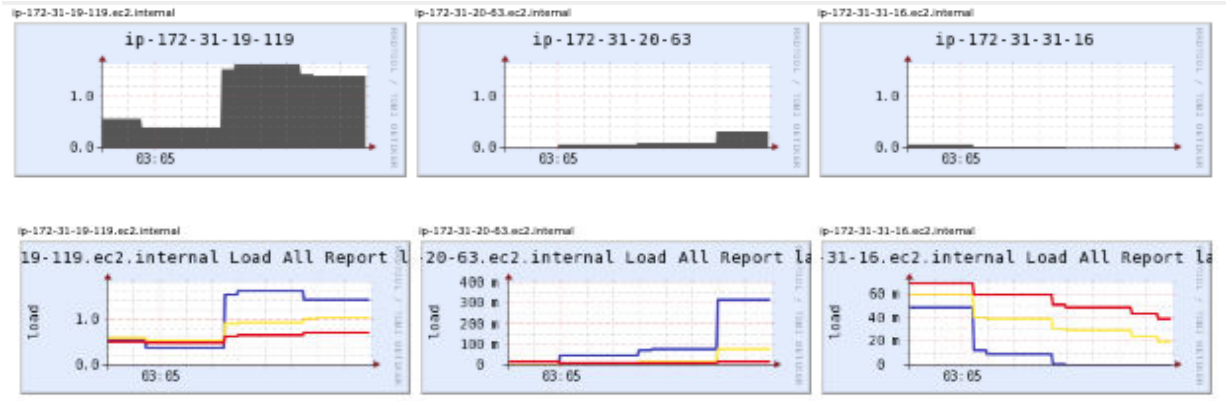


Figure 12 *Ganglia graphs for 1 executor core configuration for 3/512MB*

For 2 executor cores

Cluster view





Master node view

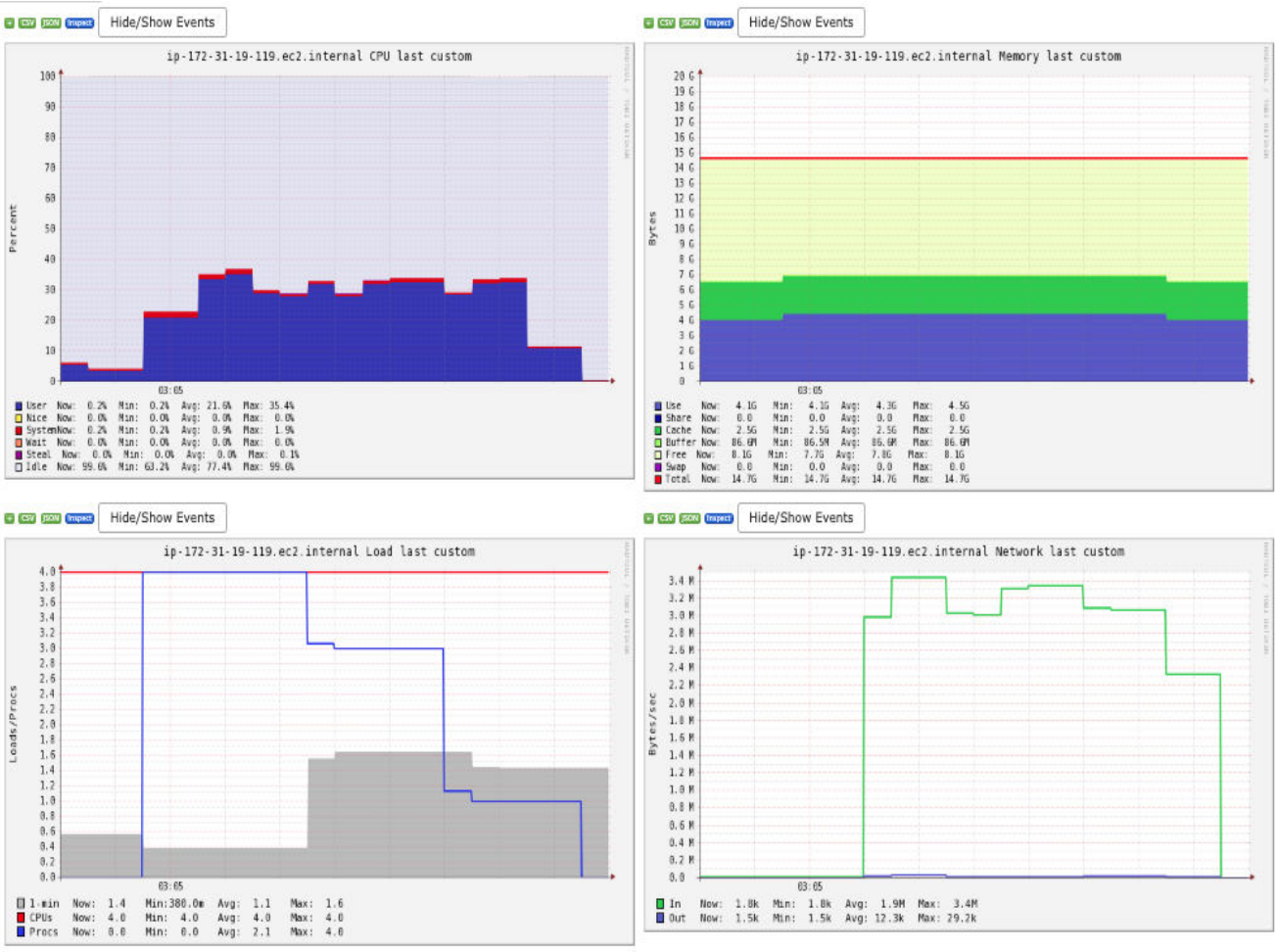
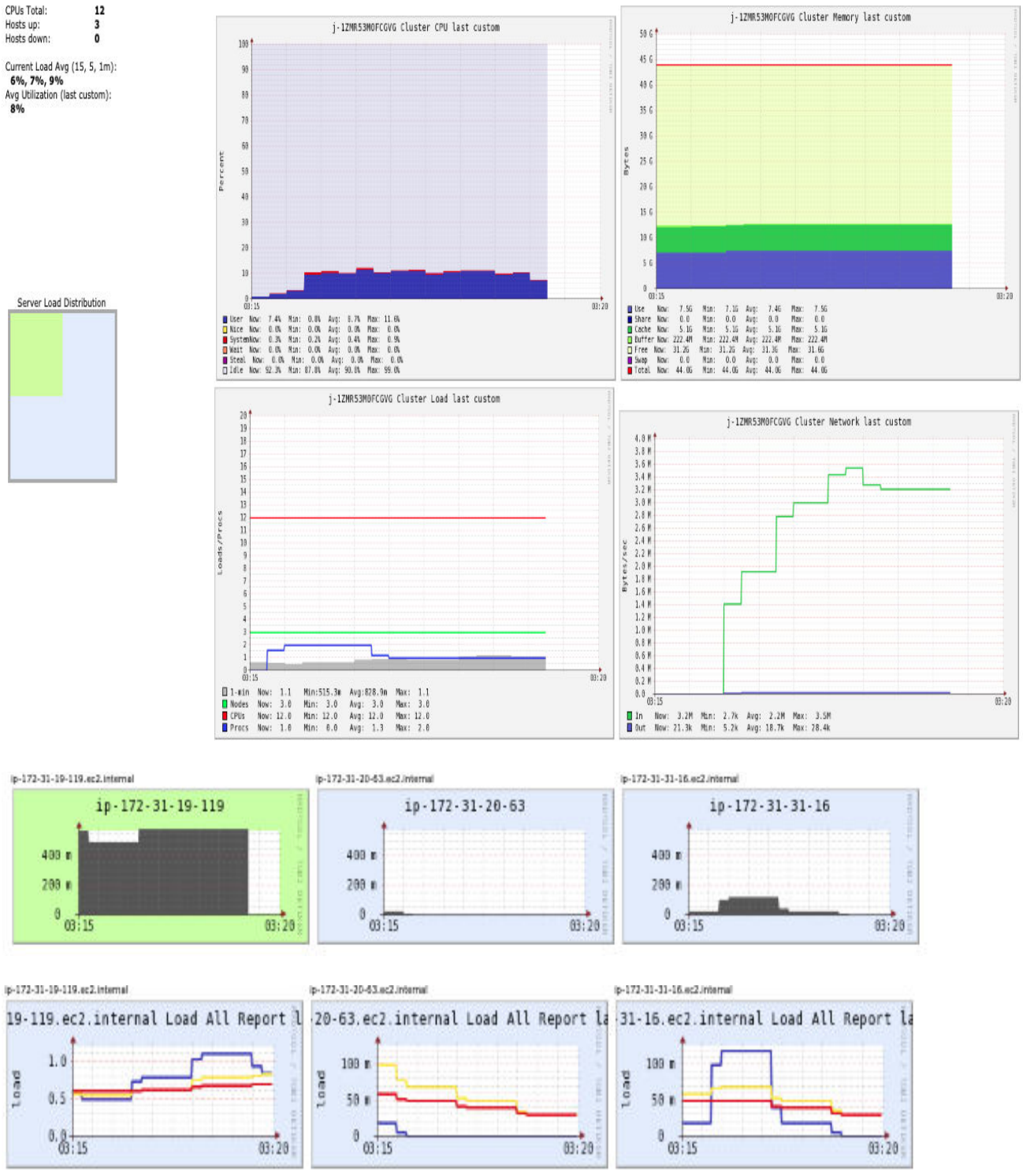


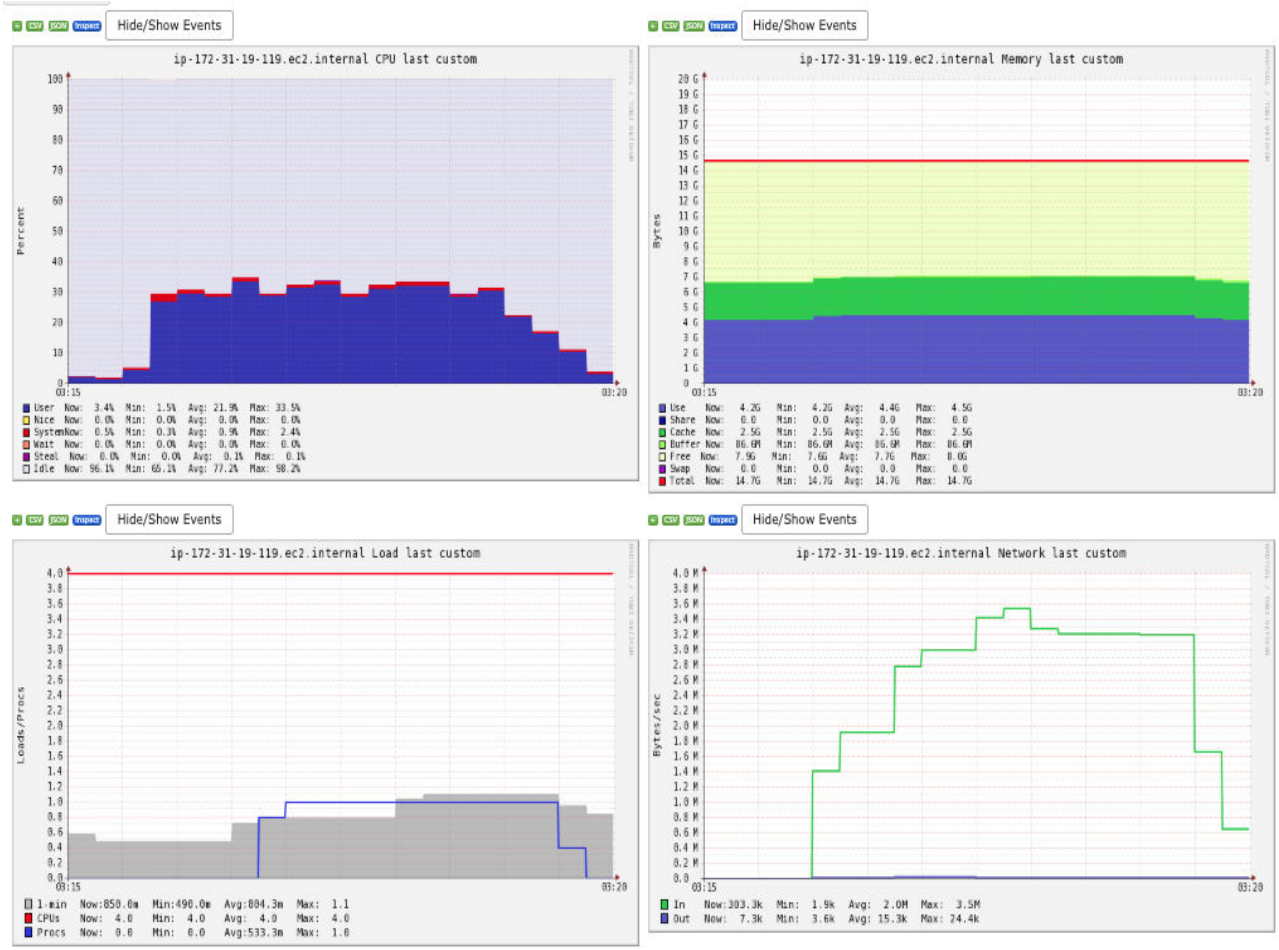
Figure 13 *Ganglia graphs for 2 executor cores configuration for 3/512MB*

For 4 executor cores

Cluster view



Master node view



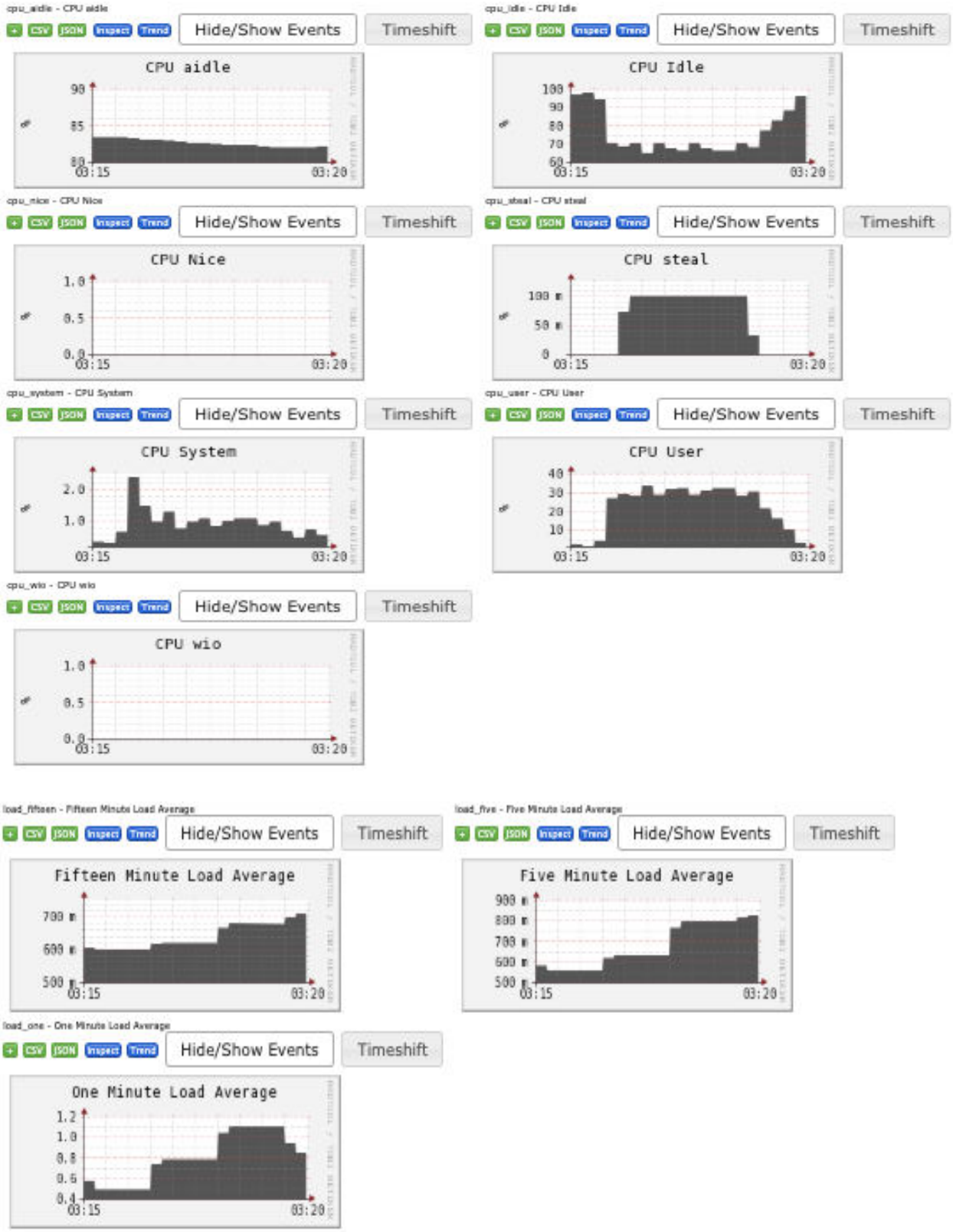
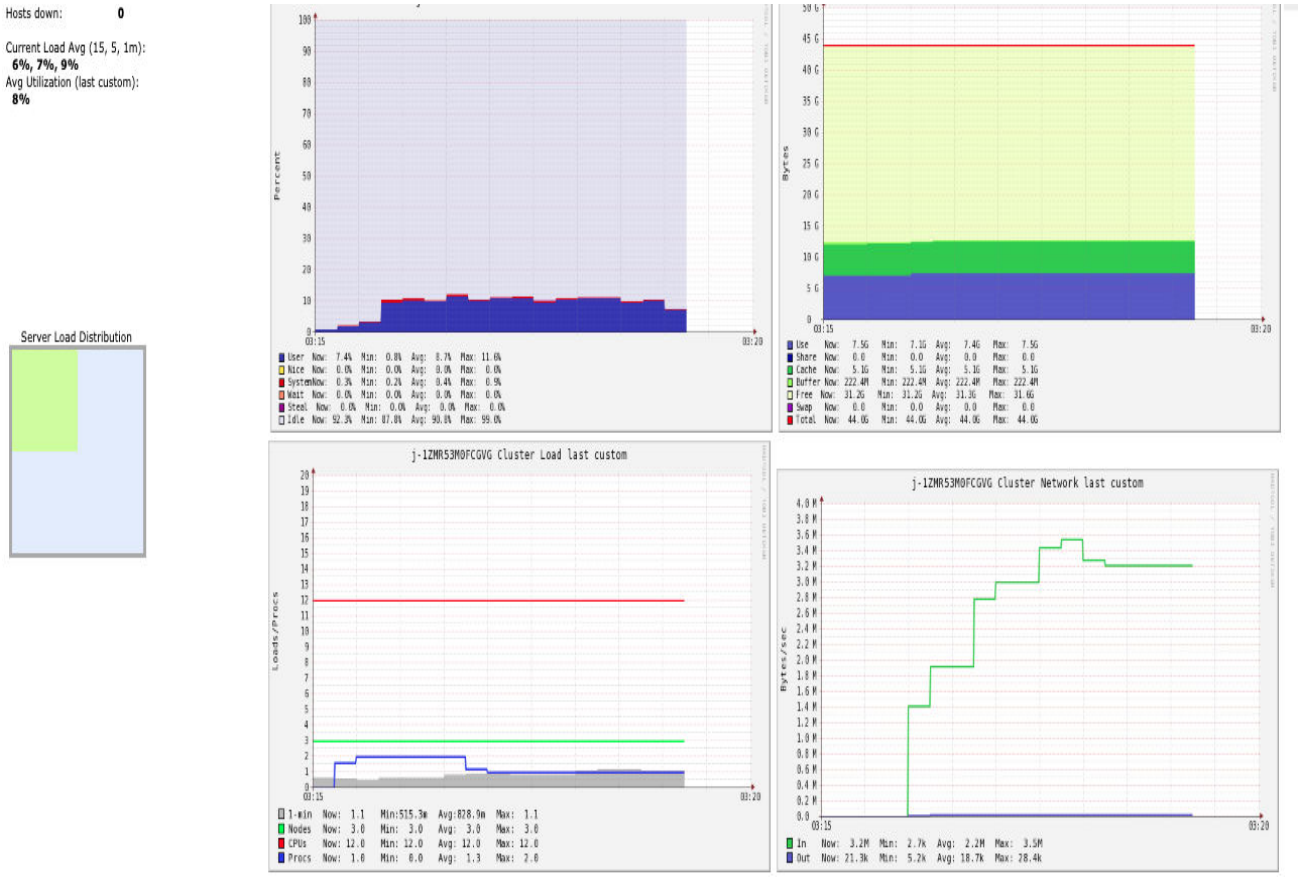
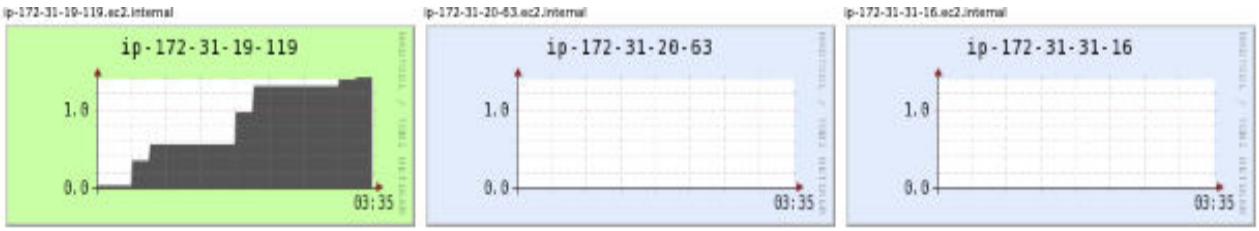


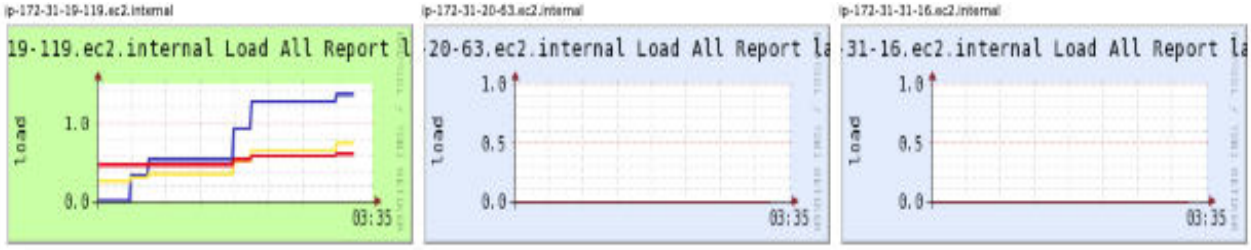
Figure 14 *Ganglia graphs for 4 executor cores configuration for 3/512MB*

For 8 executor cores

Cluster view







Master node view

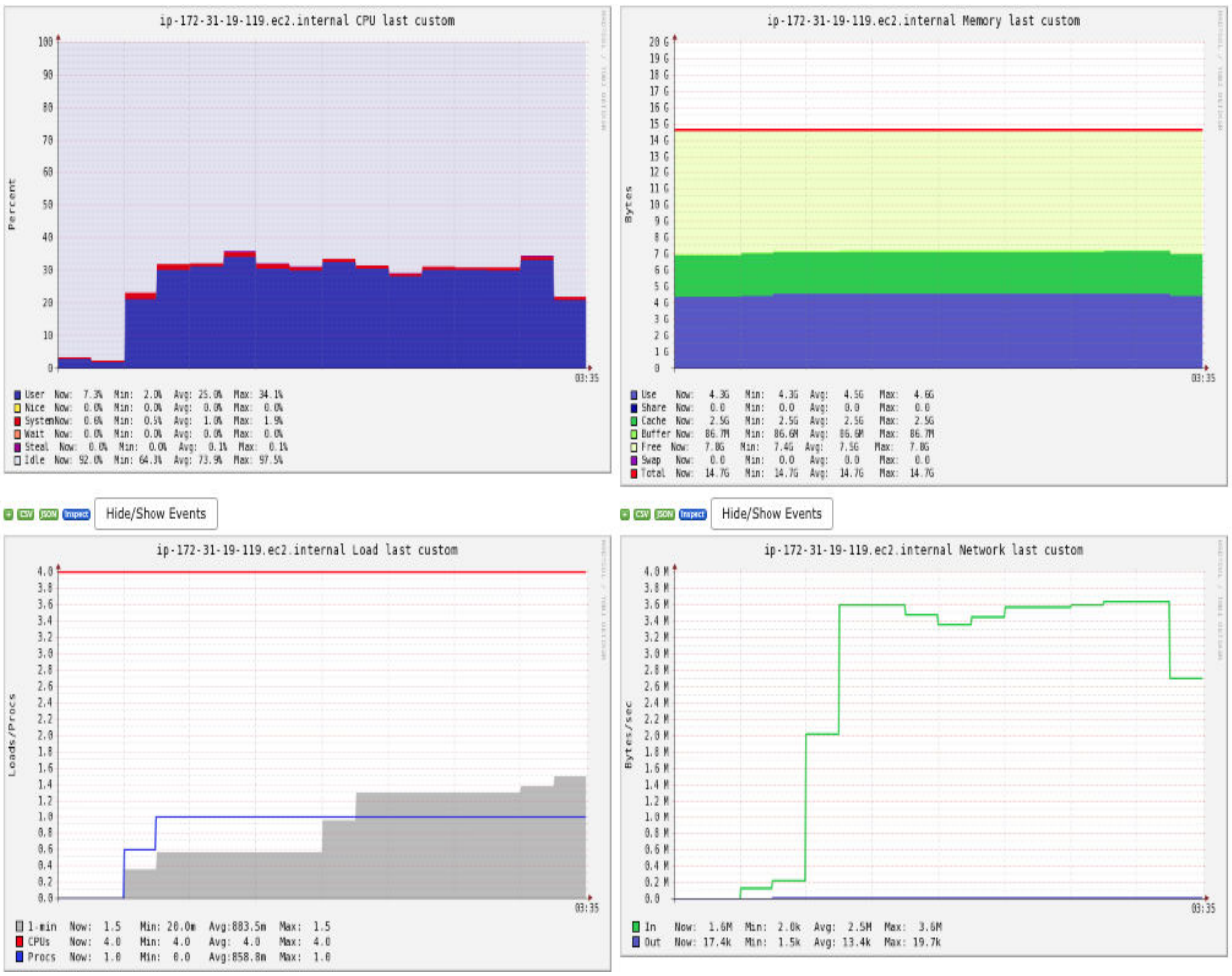


Figure 15 *Ganglia graphs for 8 executor cores configuration for 3/512MB*

Figures 12 to 15 are the corresponding Ganglia graphs for the 3/512MB pair for varying executor cores as listed in **Table 4**. The Ganglia graphs provided insightful performance metrics, namely the CPU utilization, the memory utilization, network usage and internal load for the cluster and all the individual nodes. **Figure 12** listed the aforementioned metrics for the cluster in addition to the number of hosts that were active and otherwise, along with the average utilization of the cluster recorded as 10%. It was also observed that the node under the name ip - 172 - 31 - 19 – 119, the master or driver node, was found to have performed a majority of the computations in terms of load compared to the other nodes. **Figures 13 to 15**, on the other hand, provided similar metrics for the driver and executor nodes in the cluster, where metrics such as CPU idle and CPU steal among others were taken into account to derive the CPU utilization, similarly with the memory utilization, network usage and internal, where other factors led to the final result.

Figures 12 to 15 depicted several differences. These differences include a higher average utilization percentage value in **Figure 12**, changes in the user utilization among all the configurations in the master node view, where **Figure 12** has the highest user utilization value, variations in the user memory usage of the master node from the master node graphs, consistencies in the memory usage of the cluster at approximately 5G and consistencies in the byte rate stated in the cluster network graphs. It was observed that, by varying the executor cores for the 3/512MB, the memory usage and the network usage will remain constant whereas the average utilization of the cluster decreases.

3/1GB

Table 4 showed that the average utilization increased from 6% to 7%, where the utilization degraded after varying the number of executor cores to 2, after which point the utilization increased to 7% and remained constant for two configurations. This result demonstrated that a certain number of executor cores provide better utilization statistics than other variations, a useful insight for future work with regards to determining the benchmark for how much the utilization can decrease or increase to. It was also observed that the execution time fluctuated across the number of executor cores varied.

3/2GB

Table 4 showed that the average utilization decreased from 8% to 7% when the file size was increased to 2GB and that the utilization increased in comparison with the results obtained for the 3/1GB pair. The decrease in the utilization as the number of executor cores increase was also noteworthy and is a phenomenon worth exploring in order to determine whether the number of executor cores and the utilization are related or otherwise with respect the 3/2GB pair and how much the utilization can decrease to while the number of executor cores increase.

3.5.2.2 Executor memory

This section details an analysis on the results listed in **Table 5** for 3 nodes.

3/512MB

The results indicated in **Table 5** showed that the average utilization decreased from 12% to 7% as the executor memory was increased and gradually increasing user utilization in the node views of Figures 24 to 26.

3/1GB

With respect to the results of the 3/1GB pair, the average utilization has shown to fluctuate between 7% and 8%, where a utilization of 7% was experienced when the executor memory was 2GB, demonstrating a limit the utilization can decrease to over an executor memory range of 1GB to 8GB. Once the executor memory was varied to 4GB and 8GB, the utilization became constant at 8%.

3/2GB

Table 5 demonstrated an increase in the average utilization as the executor memory was increased from 1GB to 8GB, where the utilization remained constant after 1 GB.

3.5.3 Summarization of the results and Spark Web UI results for the 3 nodes configuration

This section details samples of the DAGs (Directed Acyclic Graphs), which detail the flow of the tasks per stage as shown below, the event timeline per stage, task duration, read and write durations, stage details, tasks details and executor details for the 3/512MB pair listed in **Table 4**. Section 3.5.3.1 entails screenshots of the results from the Spark Web UI for the 3/512MB pair in

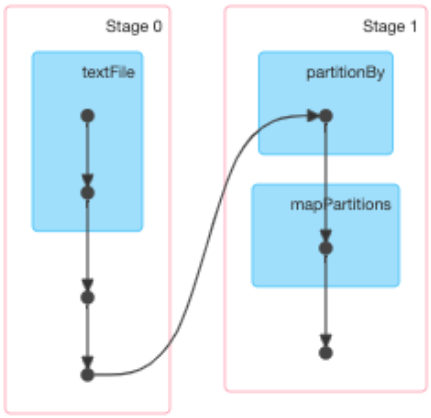
Table 4 along with drafted conclusions of the tabulated Spark Web UI results for the rest of the pairs in the same table, as is the case for section 3.5.3.2.

3.5.3.1 Executor cores

This section comprises of the results obtained for the configurations respective to **Table 4**.

3/512MB

For 1 executor core



Details for Stage 0 (Attempt 0)

Total Time Across All Tasks: 3.2 min
Locality Level Summary: Any: 8
Input Size / Records: 500.0 MB / 10242617
Shuffle Write: 201.6 KB / 436

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 8 Completed Tasks

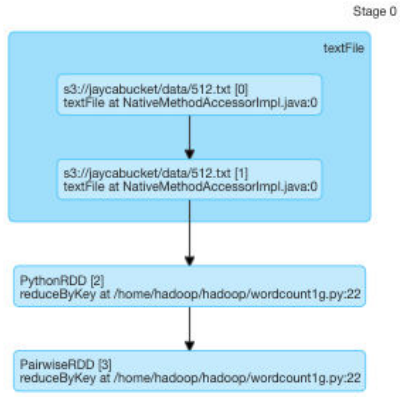
Metric	Min	25th percentile	Median	75th percentile	Max
Duration	19 s	24 s	24 s	24 s	26 s
GC Time	23 ms	31 ms	32 ms	42 ms	96 ms
Input Size / Records	52.0 MB / 1064793	64.0 MB / 1311176	64.0 MB / 1311193	64.0 MB / 1311201	64.0 MB / 1311205
Shuffle Write Size / Records	24.9 KB / 54	24.9 KB / 54	25.0 KB / 54	26.0 KB / 56	26.0 KB / 56

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Shuffle Write Size / Records	Blacklisted
driver	ip-172-31-19-119.ec2.internal:45169	3.2 min	8	0	0	8	500.0 MB / 10242617	201.6 KB / 436	false

Tasks (8)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:55:09	26 s	96 ms	64.0 MB / 1311193	9 ms	24.9 KB / 54	
1	1	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:55:35	24 s	42 ms	64.0 MB / 1311201	4 ms	24.9 KB / 54	
2	2	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:56:00	24 s	31 ms	64.0 MB / 1311176	3 ms	25.0 KB / 54	
3	3	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:56:24	24 s	37 ms	64.0 MB / 1310658	3 ms	26.0 KB / 56	
4	4	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:56:48	24 s	32 ms	64.0 MB / 1311199	3 ms	24.9 KB / 54	
5	5	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:57:12	25 s	31 ms	64.0 MB / 1311205	3 ms	24.9 KB / 54	
6	6	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:57:37	24 s	31 ms	64.0 MB / 1311192	3 ms	25.0 KB / 54	
7	7	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:58:01	19 s	23 ms	52.0 MB / 1064793	3 ms	26.0 KB / 56	



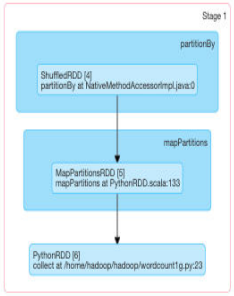
Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	19 s	24 s	24 s	24 s	26 s
Scheduler Delay	5 ms	7 ms	10 ms	14 ms	44 ms
Task Deserialization Time	3 ms	5 ms	7 ms	12 ms	28 ms
GC Time	23 ms	31 ms	32 ms	42 ms	96 ms
Result Serialization Time	0 ms	0 ms	1 ms	1 ms	2 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	52.0 MB / 1064793	64.0 MB / 1311176	64.0 MB / 1311193	64.0 MB / 1311201	64.0 MB / 1311205
Shuffle Write Size / Records	24.9 KB / 54	24.9 KB / 54	25.0 KB / 54	26.0 KB / 56	26.0 KB / 56

Details for Stage 1 (Attempt 0)

Total Time Across All Tasks: 0.4 s
Locality Level Summary: Any: 8
Shuffle Read: 201.6 KB / 436

▾ DAG Visualization



▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	50 ms	51 ms	52 ms	59 ms	63 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	24.1 KB / 50	25.2 KB / 56	25.4 KB / 56	26.1 KB / 56	26.3 KB / 56

▾ Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Read Size / Records	Blacklisted
driver	ip-172-31-19-119.ec2.internal:45169	0.5 s	8	0	0	8	201.6 KB / 436	false

▾ Tasks (8)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Errors
0	8	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:58:20	73 ms		24.1 KB / 56	
1	9	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:58:20	69 ms		26.3 KB / 50	
2	10	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:58:20	59 ms		25.4 KB / 56	
3	11	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:58:20	57 ms		25.2 KB / 56	
4	12	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:58:20	58 ms		26.1 KB / 56	
5	13	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:58:20	62 ms		25.2 KB / 56	
6	14	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:58:20	60 ms		25.4 KB / 56	
7	15	0	SUCCESS	ANY	driver	localhost	2019/04/12 02:58:20	58 ms		24.1 KB / 50	

Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	50 ms	51 ms	52 ms	59 ms	63 ms
Scheduler Delay	3 ms	4 ms	4 ms	5 ms	6 ms
Task Deserialization Time	3 ms	3 ms	4 ms	4 ms	5 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Result Serialization Time	0 ms	0 ms	0 ms	1 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	0 ms	0 ms	0 ms	0 ms	1 ms
Shuffle Read Size / Records	24.1 KB / 50	25.2 KB / 56	25.4 KB / 56	26.1 KB / 56	26.3 KB / 56
Shuffle Remote Reads	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(1)	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.2 min (0.3 s)	524.3 MB	206.5 KB	206.5 KB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.2 min (0.3 s)	524.3 MB	206.5 KB	206.5 KB	0

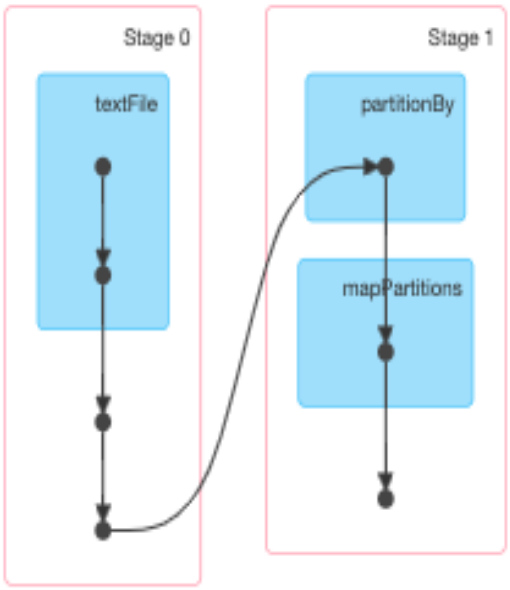
Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
driver	ip-172-31-19-119.ec2.internal:45169	Active	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.2 min (0.3 s)	524.3 MB	206.5 KB	206.5 KB

Figure 16 *DAGs, event timelines, aggregated metrics, details of the tasks per stage and executor details for the 3/512MB configuration at 1 executor core.*

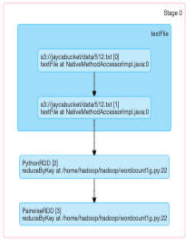
For 2 executor cores



Details for Stage 0 (Attempt 0)

Total Time Across All Tasks: 3.1 min
Locality Level Summary: Any: 8
Input Size / Records: 500.1 MB / 10242617
Shuffle Write: 201.6 KB / 436

▾ DAG Visualization



▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 8 Completed Tasks

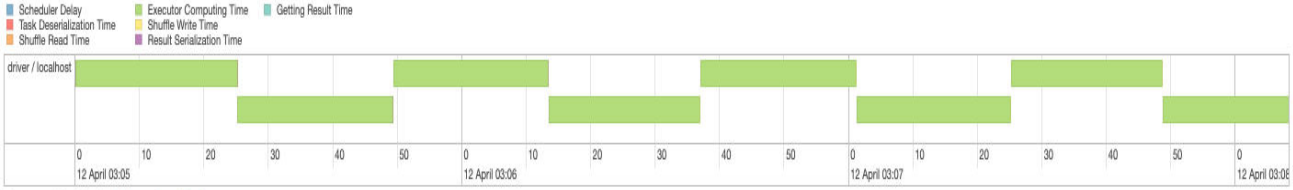
Metric	Min	25th percentile	Median	75th percentile	Max
Duration	20 s	23 s	24 s	24 s	25 s
GC Time	27 ms	33 ms	35 ms	41 ms	0.1 s
Input Size / Records	52.0 MB / 1064793	64.0 MB / 1311176	64.0 MB / 1311193	64.0 MB / 1311201	64.0 MB / 1311205
Shuffle Write Size / Records	24.9 KB / 54	24.9 KB / 54	25.0 KB / 54	26.0 KB / 56	26.0 KB / 56

▾ Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Shuffle Write Size / Records	Blacklisted
driver	ip-172-31-19-119.ec2.internal:35395	3.1 min	8	0	0	8	500.1 MB / 10242617	201.6 KB / 436	false

▾ Tasks (8)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:04:59	25 s	0.1 s	64.0 MB / 1311193	9 ms	24.9 KB / 54	
1	1	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:05:25	24 s	41 ms	64.0 MB / 1311201	4 ms	24.9 KB / 54	
2	2	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:05:49	24 s	31 ms	64.0 MB / 1311176	3 ms	25.0 KB / 54	
3	3	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:06:13	23 s	36 ms	64.0 MB / 1310658	3 ms	26.0 KB / 56	
4	4	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:06:36	24 s	33 ms	64.0 MB / 1311199	3 ms	24.9 KB / 54	
5	5	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:07:01	24 s	35 ms	64.0 MB / 1311205	3 ms	24.9 KB / 54	
6	6	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:07:25	23 s	34 ms	64.0 MB / 1311192	3 ms	25.0 KB / 54	
7	7	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:07:48	20 s	27 ms	52.0 MB / 1064793	2 ms	26.0 KB / 56	



Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	20 s	23 s	24 s	24 s	25 s
Scheduler Delay	5 ms	5 ms	6 ms	8 ms	47 ms
Task Deserialization Time	4 ms	6 ms	8 ms	9 ms	27 ms
GC Time	27 ms	33 ms	35 ms	41 ms	0.1 s
Result Serialization Time	0 ms	0 ms	0 ms	1 ms	2 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	52.0 MB / 1064793	64.0 MB / 1311176	64.0 MB / 1311193	64.0 MB / 1311201	64.0 MB / 1311205
Shuffle Write Size / Records	24.9 KB / 54	24.9 KB / 54	25.0 KB / 54	26.0 KB / 56	26.0 KB / 56

Details for Stage 1 (Attempt 0)

Total Time Across All Tasks: 0.4 s
Locality Level Summary: Any: 8
Shuffle Read: 201.6 KB / 436

- DAG Visualization

Stage 1
partitionBy
ShuffledRDD [4]
partitionBy at NativeMethodAccessorImpl.java:0
mapPartitions
MapPartitionsRDD [5]
mapPartitions at PythonRDD.scala:133
PythonRDD [6]
collect at /home/hadoop/hadoop/wordcount1g.py:23

▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	51 ms	52 ms	54 ms	60 ms	62 ms
GC Time	0 ms	0 ms	0 ms	0 ms	3 ms
Shuffle Read Size / Records	24.1 KB / 50	25.2 KB / 56	25.4 KB / 56	26.1 KB / 56	26.3 KB / 56

▾ Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Read Size / Records	Blacklisted
driver	ip-172-31-19-119.ec2.internal:35395	0.5 s	8	0	0	8	201.6 KB / 436	false

▾ Tasks (8)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Errors
0	8	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:08:08	64 ms		24.1 KB / 56	
1	9	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:08:08	71 ms		26.3 KB / 50	
2	10	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:08:08	60 ms		25.4 KB / 56	
3	11	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:08:08	60 ms		25.2 KB / 56	
4	12	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:08:08	57 ms		26.1 KB / 56	
5	13	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:08:08	58 ms		25.2 KB / 56	
6	14	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:08:08	60 ms		25.4 KB / 56	
7	15	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:08:08	68 ms	3 ms	24.1 KB / 50	

Scheduler Delay
Task Deserialization Time
Shuffle Read Time
Executor Computing Time
Shuffle Write Time
Result Serialization Time
Getting Result Time

driver / localhost

500 550 600 650 700 750 800 850 900 950
03:08:08

Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	51 ms	52 ms	54 ms	60 ms	62 ms
Scheduler Delay	4 ms	4 ms	5 ms	5 ms	5 ms
Task Deserialization Time	2 ms	2 ms	2 ms	3 ms	4 ms
GC Time	0 ms	0 ms	0 ms	0 ms	3 ms
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	24.1 KB / 50	25.2 KB / 56	25.4 KB / 56	26.1 KB / 56	26.3 KB / 56
Shuffle Remote Reads	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(1)	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.1 min (0.3 s)	524.4 MB	206.5 KB	206.5 KB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.1 min (0.3 s)	524.4 MB	206.5 KB	206.5 KB	0

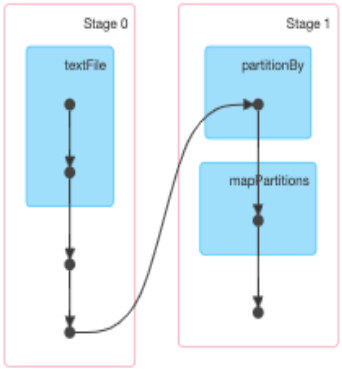
Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
driver	ip-172-31-19-119.ec2.internal:35395	Active	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.1 min (0.3 s)	524.4 MB	206.5 KB	206.5 KB

Figure 17 *DAGs, event timelines, aggregated metrics, details of the tasks per stage and executor details for the 3/512MB configuration at 2 executor cores.*

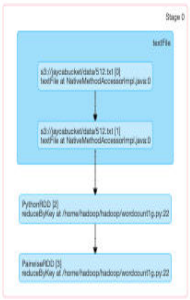
For 4 executor cores



Details for Stage 0 (Attempt 0)

Total Time Across All Tasks: 3.2 min
Locality Level Summary: Any: 8
Input Size / Records: 500.1 MB / 10242617
Shuffle Write: 201.6 KB / 436

▾ DAG Visualization



› Show Additional Metrics
› Event Timeline

Summary Metrics for 8 Completed Tasks

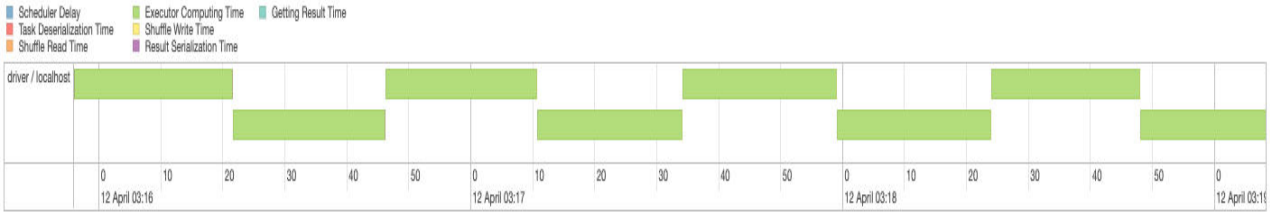
Metric	Min	25th percentile	Median	75th percentile	Max
Duration	20 s	24 s	25 s	25 s	26 s
GC Time	27 ms	34 ms	37 ms	48 ms	67 ms
Input Size / Records	52.0 MB / 1064793	64.0 MB / 1311176	64.0 MB / 1311193	64.0 MB / 1311201	64.0 MB / 1311205
Shuffle Write Size / Records	24.9 KB / 54	24.9 KB / 54	25.0 KB / 54	26.0 KB / 56	26.0 KB / 56

▾ Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Shuffle Write Size / Records	Blacklisted
driver	ip-172-31-19-119.ec2.internal:42981	3.2 min	8	0	0	8	500.1 MB / 10242617	201.6 KB / 436	false

▾ Tasks (8)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:15:55	26 s	67 ms	64.0 MB / 1311193	9 ms	24.9 KB / 54	
1	1	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:16:21	25 s	48 ms	64.0 MB / 1311201	3 ms	24.9 KB / 54	
2	2	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:16:46	24 s	36 ms	64.0 MB / 1311176	3 ms	25.0 KB / 54	
3	3	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:17:10	24 s	34 ms	64.0 MB / 1310658	3 ms	26.0 KB / 56	
4	4	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:17:34	25 s	32 ms	64.0 MB / 1311199	3 ms	24.9 KB / 54	
5	5	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:17:59	25 s	38 ms	64.0 MB / 1311205	3 ms	24.9 KB / 54	
6	6	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:18:24	24 s	37 ms	64.0 MB / 1311192	2 ms	25.0 KB / 54	
7	7	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:18:48	20 s	27 ms	52.0 MB / 1064793	3 ms	26.0 KB / 56	



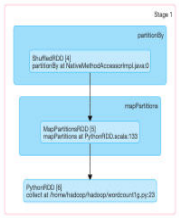
Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	20 s	24 s	25 s	25 s	26 s
Scheduler Delay	4 ms	5 ms	6 ms	11 ms	58 ms
Task Deserialization Time	4 ms	6 ms	6 ms	8 ms	29 ms
GC Time	27 ms	34 ms	37 ms	48 ms	67 ms
Result Serialization Time	0 ms	0 ms	0 ms	1 ms	2 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	52.0 MB / 1064793	64.0 MB / 1311176	64.0 MB / 1311193	64.0 MB / 1311201	64.0 MB / 1311205
Shuffle Write Size / Records	24.9 KB / 54	24.9 KB / 54	25.0 KB / 54	26.0 KB / 56	26.0 KB / 56

Details for Stage 1 (Attempt 0)

Total Time Across All Tasks: 0.5 s
Locality Level Summary: Any: 8
Shuffle Read: 201.6 KB / 436

- DAG Visualization



› Show Additional Metrics
› Event Timeline

Summary Metrics for 8 Completed Tasks

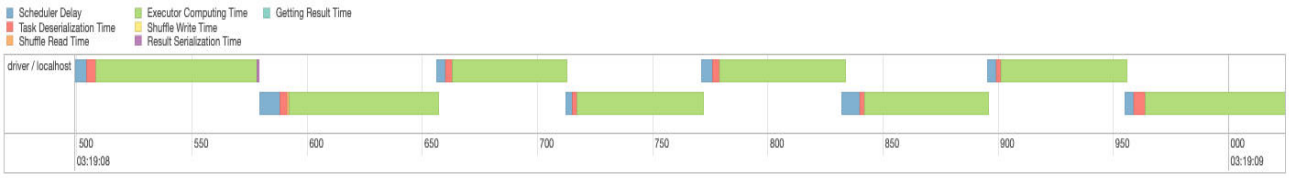
Metric	Min	25th percentile	Median	75th percentile	Max
Duration	50 ms	55 ms	55 ms	66 ms	70 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	24.1 KB / 50	25.2 KB / 56	25.4 KB / 56	26.1 KB / 56	26.3 KB / 56

- Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Read Size / Records	Blacklisted
driver	ip-172-31-19-119.ec2.internal:42981	0.5 s	8	0	0	8	201.6 KB / 436	false

- Tasks (8)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Errors
0	8	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:19:08	80 ms		24.1 KB / 56	
1	9	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:19:08	78 ms		26.3 KB / 50	
2	10	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:19:08	57 ms		25.4 KB / 56	
3	11	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:19:08	60 ms		25.2 KB / 56	
4	12	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:19:08	63 ms		26.1 KB / 56	
5	13	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:19:08	64 ms		25.2 KB / 56	
6	14	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:19:08	61 ms		25.4 KB / 56	
7	15	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:19:08	70 ms		24.1 KB / 50	



Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	50 ms	55 ms	55 ms	66 ms	70 ms
Scheduler Delay	3 ms	4 ms	5 ms	8 ms	9 ms
Task Deserialization Time	2 ms	2 ms	3 ms	4 ms	5 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	0 ms	0 ms	0 ms	0 ms	1 ms
Shuffle Read Size / Records	24.1 KB / 50	25.2 KB / 56	25.4 KB / 56	26.1 KB / 56	26.3 KB / 56
Shuffle Remote Reads	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(1)	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.2 min (0.3 s)	524.4 MB	206.5 KB	206.5 KB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.2 min (0.3 s)	524.4 MB	206.5 KB	206.5 KB	0

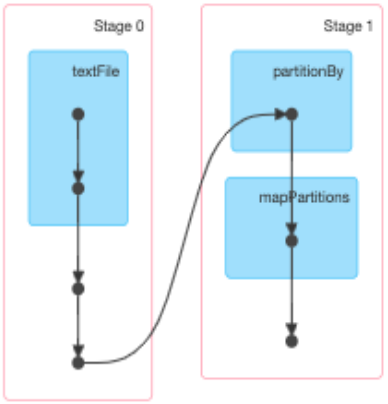
Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
driver	ip-172-31-19-119.ec2.internal:42981	Active	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.2 min (0.3 s)	524.4 MB	206.5 KB	206.5 KB

Figure 18 *DAGs, event timelines, aggregated metrics, details of the tasks per stage and executor details for the 3/512MB configuration at 4 executor cores.*

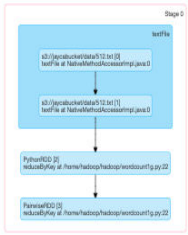
For 8 executor cores



Details for Stage 0 (Attempt 0)

Total Time Across All Tasks: 3.1 min
Locality Level Summary: Any: 8
Input Size / Records: 500.1 MB / 10242617
Shuffle Write: 201.6 KB / 436

- DAG Visualization



- Show Additional Metrics
- Event Timeline

Summary Metrics for 8 Completed Tasks

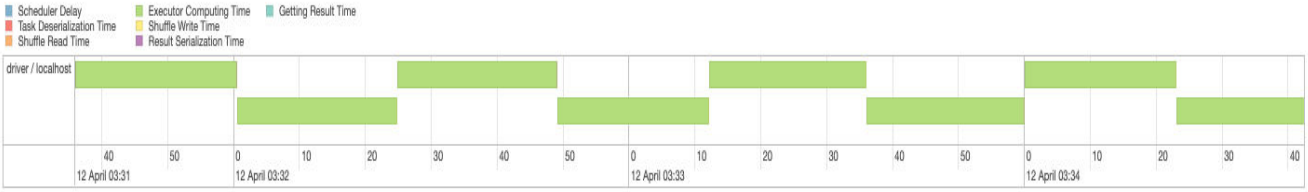
Metric	Min	25th percentile	Median	75th percentile	Max
Duration	19 s	23 s	24 s	24 s	25 s
GC Time	35 ms	37 ms	41 ms	51 ms	67 ms
Input Size / Records	52.0 MB / 1064793	64.0 MB / 1311176	64.0 MB / 1311193	64.0 MB / 1311201	64.0 MB / 1311205
Shuffle Write Size / Records	24.9 KB / 54	24.9 KB / 54	25.0 KB / 54	26.0 KB / 56	26.0 KB / 56

- Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Shuffle Write Size / Records	Blacklisted
driver	ip-172-31-19-119.ec2.internal:42103	3.1 min	8	0	0	8	500.1 MB / 10242617	201.6 KB / 436	false

- Tasks (8)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	0	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:31:35	25 s	67 ms	64.0 MB / 1311193	8 ms	24.9 KB / 54	
1	1	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:32:00	24 s	51 ms	64.0 MB / 1311201	3 ms	24.9 KB / 54	
2	2	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:32:24	24 s	38 ms	64.0 MB / 1311176	3 ms	25.0 KB / 54	
3	3	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:32:49	23 s	36 ms	64.0 MB / 1310658	3 ms	26.0 KB / 56	
4	4	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:33:12	24 s	37 ms	64.0 MB / 1311199	3 ms	24.9 KB / 54	
5	5	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:33:36	24 s	41 ms	64.0 MB / 1311205	3 ms	24.9 KB / 54	
6	6	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:34:00	23 s	35 ms	64.0 MB / 1311192	3 ms	25.0 KB / 54	
7	7	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:34:23	19 s	44 ms	52.0 MB / 1064793	3 ms	26.0 KB / 56	



Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	19 s	23 s	24 s	24 s	25 s
Scheduler Delay	4 ms	5 ms	10 ms	18 ms	45 ms
Task Deserialization Time	3 ms	4 ms	6 ms	9 ms	29 ms
GC Time	35 ms	37 ms	41 ms	51 ms	67 ms
Result Serialization Time	0 ms	0 ms	1 ms	1 ms	2 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	52.0 MB / 1064793	64.0 MB / 1311176	64.0 MB / 1311193	64.0 MB / 1311201	64.0 MB / 1311205
Shuffle Write Size / Records	24.9 KB / 54	24.9 KB / 54	25.0 KB / 54	26.0 KB / 56	26.0 KB / 56

Details for Stage 1 (Attempt 0)

Total Time Across All Tasks: 0.4 s
Locality Level Summary: Any: 8
Shuffle Read: 201.6 KB / 436

▸ DAG Visualization

Stage 1
partitionBy
ShuffledRDD [4]
partitionBy at NativeMethodAccessorImpl.java:0
mapPartitions
MapPartitionsRDD [5]
mapPartitions at PythonRDD.scala:133
PythonRDD [6]
collect at /home/hadoop/hadoop/wordcount1g.py:23

▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	52 ms	53 ms	54 ms	55 ms	58 ms
GC Time	0 ms	0 ms	0 ms	0 ms	3 ms
Shuffle Read Size / Records	24.1 KB / 50	25.2 KB / 56	25.4 KB / 56	26.1 KB / 56	26.3 KB / 56

▾ Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Read Size / Records	Blacklisted
driver	ip-172-31-19-119.ec2.internal:42103	0.5 s	8	0	0	8	201.6 KB / 436	false

▾ Tasks (8)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Errors
0	8	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:34:42	63 ms		24.1 KB / 56	
1	9	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:34:42	66 ms		26.3 KB / 50	
2	10	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:34:42	62 ms		25.4 KB / 56	
3	11	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:34:42	60 ms		25.2 KB / 56	
4	12	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:34:42	62 ms		26.1 KB / 56	
5	13	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:34:42	63 ms		25.2 KB / 56	
6	14	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:34:42	61 ms		25.4 KB / 56	
7	15	0	SUCCESS	ANY	driver	localhost	2019/04/12 03:34:43	60 ms	3 ms	24.1 KB / 50	

Scheduler Delay
Task Deserialization Time
Shuffle Read Time
Executor Computing Time
Shuffle Write Time
Result Serialization Time
Getting Result Time

driver / localhost

650 700 750 800 850 900 950 000 050 100
03:34:42 03:34:43

Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	52 ms	53 ms	54 ms	55 ms	58 ms
Scheduler Delay	2 ms	4 ms	5 ms	6 ms	7 ms
Task Deserialization Time	2 ms	3 ms	3 ms	3 ms	5 ms
GC Time	0 ms	0 ms	0 ms	0 ms	3 ms
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Shuffle Read Blocked Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	24.1 KB / 50	25.2 KB / 56	25.4 KB / 56	26.1 KB / 56	26.3 KB / 56
Shuffle Remote Reads	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B

Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(1)	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.1 min (0.4 s)	524.4 MB	206.5 KB	206.5 KB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.1 min (0.4 s)	524.4 MB	206.5 KB	206.5 KB	0

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
driver	ip-172-31-19-119.ec2.internal:42103	Active	0	0.0 B / 434.6 MB	0.0 B	1	0	0	16	16	3.1 min (0.4 s)	524.4 MB	206.5 KB	206.5 KB

Figure 19 *DAGs, event timelines, aggregated metrics, details of the tasks per stage and executor details for the 3/512MB configuration at 8 executor cores.*

Figures 16 to 19 are the Spark Web UI results obtained for the 3/512MB pair relative to the variation in the number of executor cores as listed in **Table 4**. Included in the figures are the DAG, the median durations of stages 0 and 1 along with information related to the tasks such as the scheduler delay, the event timelines for each stage and the task duration as shown in the executors summary screenshot in **Figure 19**. The actions performed in each stage of a job differ across many applications. In the case of this project however, stage 0 pertains to the reading of the text file and the reduction of the file, which involves the addition of all the words as per the algorithm in **Appendix A**. Stage 1 pertains to the partitioning of the text file, or shuffling, followed by the mapping as a result of the partitioning prior, where the mapping process is associated with the splitting of the words in the document at the whitespaces.

By tabulating the findings above:

Table 6 *3/512MB configuration duration details for varying executor cores.*

3/512MB configuration					
Executor cores	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	24	10	52	4	3.2
2	24	6	54	5	3.1
4	25	6	55	5	3.2
8	24	10	54	5	3.1

From the table above, it was shown that the duration in stage 0 ranged between 19 seconds and 20 seconds, whereas the scheduler delay decreased from 5ms to 4ms as the cores increased. The task duration was also observed to vary between 3.1 minutes and 3.2 minutes, all of which are equivalent to the execution times listed for this configuration in **Table 4**.

The table below summarizes the Spark Web UI results for the 3/1GB configuration listed in **Table 4**.

Table 7 *3/1GB configuration duration details for varying executor cores.*

3/1GB configuration					
Executor cores	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	24	5	55	4	6.4
2	24	6	54	4	6.2
4	24	5	56	4	6.3
8	24	4	53	3	6.4

The table above shows a consistency in the median duration of stage 0 and fluctuations of the median duration in stage 1, concluding with a decrease from 56ms to 53ms. It was also observed that a low value of task duration was experienced in the case where the number of executor cores was 2 and in that same scenario, a long scheduler delay occurred.

A table consolidating the values from the information detailed in **Table 4** for the 3/2GB configuration is as shown in **Table 8**.

Table 8 *3/2GB configuration duration details for varying executor cores.*

3/2GB configuration					
Executor cores	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	25	5	53	3	13
2	24	5	53	3	12
4	24	4	53	3	13
8	24	4	53	3	13

The table above showed decreases in the median duration and in the median scheduler delay for stage 0. The result also demonstrated consistencies in the median duration and median scheduler delay in stage 1. It as additionally observed that the second configuration, where the number of executor cores was 2, was the lowest in comparison with the other configurations.

3.5.3.2 Executor memory

This section lists the tabulated results obtained for the configuration in **Table 5**.

The table below comprises of the duration and scheduler delay for each executor memory configuration.

Table 9 *3/512MB configuration duration details for varying executor memories.*

3/512MB configuration					
Executor memory (GB)	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	25	6	54	5	3.3
2	25	7	54	6	3.2
4	25	10	54	6	3.2
8	24	8	55	5	3.2

The table above demonstrated observations of threefold. The first observation being the decrease of the stage 0 with respect to the median duration, the consistency in the overall task duration

and the increase of the median duration for stage 1 as a result of the increase in the executor memory.

The table below depicts the duration of each stage and the overall duration of a task for the 3/1GB configuration in **Table 5**.

Table 10 *3/1GB configuration duration details for varying executor memories.*

3/1GB configuration					
Executor memory (GB)	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	25	5	54	4	6.6
2	25	6	54	3	6.5
4	24	5	57	4	6.4
8	24	5	55	4	6.2

According to the table above, the duration of stage 0 decreased as the executor memory increased and the task duration corresponding to each configuration of the executor memory inversely decreased as a result of the usage of a larger text file.

The table below lists the results in a tabulated manner for the configuration listed in **Table 5**.

Table 11 *3/2GB configuration duration details for varying executor memories.*

3/2GB configuration					
Executor memory (GB)	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	24	4	53	3	13
2	24	4	53	3	13
4	24	4	53	3	13
8	24	4	53	3	13

The table above demonstrated consistencies in both the median duration and the scheduler delay of stage 0, similarly with stage 1 and in the task duration. For this particular scenario, no fluctuations in the data were observed.

3.5.4 Summarization of the results for the 7 nodes configuration

This section consists of conclusions and summarizations derived from the end result achieved for the 7 nodes configuration in **Tables 4 and 5**.

3.5.4.1 Executor cores

This section details the summaries acquired for the results of the configurations listed in **Table 4**.

7/512MB

Table 4 demonstrated a continuous decrease in the average utilization as the number of executor cores increased and a newly attained low average utilization value, 4% after 2 executor cores. It was also observed that the decrease was larger than the average utilization decrease in the 3/2GB pair. The results additionally indicated a fluctuation in the execution time.

7/1GB

Table 4 indicated a constant average utilization value across all number of executor cores and a newly attained average utilization value of 3%. The execution time also remained within the same range as the 3/1GB pair, indicating a miniscule difference between both configurations.

7/2GB

Table 4 demonstrated an increase in the average utilization for the 7/2GB pair compared to the values achieved for the 7/512MB and 7/1GB pairs, exhibiting observations of twofold. The first

being a limit to how much the average utilization could decrease to relative to the number of nodes and file size and the second being the desire to conduct further research into how many more executor cores will cause an increase in the utilization. It was also noted that the execution time remained within the same range as the 3/2GB pair however, with respect to the utilization, there was a notable improvement.

3.5.4.2 Executor memory

The graphs below illustrate the Ganglia graphs obtained for **Table 5**.

7/512MB

Table 5 demonstrated a consistency in the average utilization value from executor memories 1GB to 4GB and a decrease to 2% for 8GB executor memory, indicating a decrease in the average utilization in comparison with the ones attained for the 3/512MB pair.

7/1GB

Table 5 showed that the cluster average utilization remained constant at 3%. It was noted that, by increasing the file size to 1GB and the executor memory from 1GB to 8GB, the average utilization increased from 2% when the configuration was 7/512MB for 8GB executor memory to 3% when the configuration was changed to 7/1GB.

7/2GB

Table 5 demonstrated a consistency in the average utilization value across all executor memories and an increase in the execution times in contrast with the 3/2GB pair. The results also indicated

a better average utilization across all executor memories compared to **Table 4**, where the average utilization increased for 8 executor cores for the same 7/2GB pair. Furthermore, this led to the observation that increasing the executor memories compared to increasing the number of executor cores would cause a decrease in the average utilization.

3.5.5 Summary of the Spark Web UI results for the 7 nodes configuration

This section contains the encapsulated results for the 7 nodes configuration listed in **Tables 4 and 5** from the Spark Web UI.

3.5.5.1 Executor cores

This section consists of the results for the configurations in **Table 4**.

The table below lists the durations and delays recorded for each stage when the configuration was 7/512MB.

Table 12 *7/512MB configuration duration details for varying executor cores.*

7/512MB configuration					
Executor cores	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	25	9	54	5	3.2
2	25	9	54	5	3.3
4	26	13	54	5	3.4
8	24	8	54	6	3.2

The above illustrated a consistency in the duration of stage 1 and an increase in the scheduler delay of stage 1. Additionally, it was worth nothing that the highest number of executor cores, 8, experienced the lowest task duration among the rest, where the longest task duration was when the number of executor cores was 4.

The table below encapsulates all the durations of the stages, task and scheduler delay.

Table 13 *7/1GB configuration duration details for varying executor cores.*

7/1GB configuration					
Executor cores	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	24	5	56	4	6.2
2	25	6	54	3	6.5
4	24	7	58	3	6.4
8	25	5	54	4	6.4

The table above showed that the third configuration, where the number of executor cores was varied to 4, the longest scheduler delay was experienced, similarly with the median duration of the same configuration for stage 1. It was also observed that the task duration of the case when the number of executor cores was 2 was the longest and the task duration when the number of executor cores was 1 was the lowest among all the other configurations.

Below is a table listing the durations of each stage, as well as the scheduler delay and task duration.

Table 14 *7/2GB configuration duration details for varying executor cores.*

7/2GB configuration					
Executor cores	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	24	4	55	3	13
2	24	4	54	3	12
4	24	5	55	3	13
8	24	4	53	3	13

The table above illustrated a consistency in the duration of stage 0 and in the scheduler delay as a result of increasing the file size to 2GB.

3.5.5.2 Executor memory

This section details the results respective to **Table 5**.

The table listed below provides the duration of each stage, the task done and the delay of the scheduler.

Table 15 *7/512MB configuration duration details for varying executor memories.*

7/512MB configuration					
Executor memory (GB)	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	25	9	57	5	3.4
2	25	9	54	5	3.4
4	24	7	51	4	3.2
8	24	7	53	4	3.1

The table above showed decreases overall, with the exception of the median duration of stage 1. By comparing the results of the above with the results obtained in **Table 9**, the task duration experienced a minimum value of 3.1 when the number of nodes was increased to 7.

The table below shows the duration of the task, of each stage and the delay experienced by the scheduler per stage.

Table 16 *7/1GB configuration duration details for varying executor memories.*

7/1GB configuration					
Executor memory (GB)	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	24	5	55	4	6.4
2	25	5	54	4	6.5
4	25	6	55	4	6.4
8	24	5	55	4	6.4

The above demonstrated a constant scheduler delay in stage 1. It was also noted that, when the executor memory was 2GB, the longest task duration occurred and dropped back to 6.4 minutes

for the rest of the configurations. However, by comparing the above results with the ones obtained in **Table 10**, **Table 10** was shown to have a maximum value of 6.6 minutes with respect to the task duration when the executor memory was 1GB, whereas for the above, a task duration of 6.4 minutes was attained. It is also worth noting that the 8GB configuration in **Table 10** experienced the minimum value between **Tables 10 and 16** in spite of the increase in the nodes.

The table below details the duration of each stage along with the scheduler delays.

Table 17 *7/2GB configuration duration details for varying executor memories.*

7/2GB configuration					
Executor memory (GB)	Stage 0		Stage 1		Task Duration (minutes)
	Median duration (s)	Median scheduler delay (ms)	Median duration (ms)	Median scheduler delay (ms)	
1	24	4	53	2	13
2	24	4	60	3	12
4	24	4	53	2	13
8	24	4	53	3	12

The table above showed a consistency in the scheduler delay and median duration of stage 0. The above table also indicated an increase in the median duration of stage 1, namely, when the executor memory was 2GB in comparison to the same entry on **Table 14** and a decrease in the task duration when the executor memory was 8GB compared to **Table 14**.

3.6 Chapter summary

This chapter summarized the findings in the form of sample Ganglia graphs and Spark Web UI metrics for the 3/512MB pair listed in **Tables 4 and 5** along with observations derived for each configuration per table. The results indicated that by increasing the executor memory and keeping the executor cores constant, an increase in the execution time for the 3 nodes pairs was observed while on the other hand, a decrease in the average utilization for the 7 nodes pairs was exhibited. It was also shown for the 7 nodes pairs that as the executor memory was increased, the execution times increased compared to the pairs stated in **Table 4**. In addition to the consistency of the median durations of both stages 0 and 1 for the Spark Web UI results with respect to the range, the end results listed in **Table 4 and 5**, in a comparative perspective, indicated that minimal improvements in the execution times per pair were observed for certain execution memory and core values and a significant improvement in the average utilization. Additionally, it was shown that increasing the number of executor cores or executor memory for specific workloads demonstrated less impact on the overall performance and most importantly, an introduction to a limit to how much further the average utilization or execution time, or both, could change respective to the number of executor cores and executor memory per workload as indicated in the trends of **Tables 4 and 5**. Expounding on this, the results in this chapter brought

several interesting insights, as well as prompts for further work, which will be detailed in the following chapter.

Chapter 4

Discussions

The sample Ganglia graphs obtained above demonstrated the usage statistics in terms of CPU, network, memory and load, where the memory utilization remained fairly consistent across the configurations, as expected per the features of Spark, where memory utilization is low due to the in-memory processing feature, further reinforced by the durations recorded in **Tables 6 to 17**, where the delays did not exceed 10ms and per the details illustrated in the Spark Web UI results for both **Tables 4 and 5**.

Also recorded in the results section were the execution times and average utilization values of each configuration for each workload, where the data propagation delay was not factored into the resulting execution time due to the prior upload of the files to the cloud. In accordance with the results obtained, it was observed that, when the number of executor cores was at the default value and the executor memory was varied from 1GB to 8GB, an average utilization of 2%, which was lower than the average utilization value of 3% in **Table 4**, was achieved. In **Table 5,** it was also shown that, when the number of nodes was changed to 7, several increases and decreases were observed with respect to the execution time, similarly for the 3 nodes configuration. Another observation relates to the increase in the average utilization time for the 3 nodes configuration when the executor memory was varied while the number of executor cores is 1. A possible reason behind the miniscule improvement in performance, with respect to the execution time, and simultaneously, degradation in performance, arises from the default

configuration of the spark cluster, static allocation, instead of dynamic allocation, where the resources are returned to the cluster in the event of nonuse [33]. Another reason behind the decrease in the performance lies in the cluster type. As observed in **Table 4**, the execution times across each configuration fluctuated. Notably, there was an increase in the execution time when the configuration was 2 executor cores for 7/512MB to the next and following that, a decrease was observed as well. Similarly, when the executor memory was varied across the 1GB to 8GB range, while the execution time dropped for the 7 nodes configuration, the execution time increased for the 3 nodes configuration. Although, in spite of the fluctuations in the execution times for all the configurations, the average utilization dropped significantly in **Table 5** in contrast with **Table 4**.

Chapter 5

Conclusions and Future Work

In contrast to the results provided in the related ventures conducted on the subject of performance evaluation of big data applications using Spark, the results of this project presented several conclusions. One of the conclusions drawn from the observations of the results included how a specific configuration of executor core and executor memory for a pair differs among others and provides the best execution time. For example, by referring to **Table 4**, it was shown that the highest number of executor cores, 8, provided the best execution time for the 3/512MB pair, whereas for the 3/1GB and 3/2GB pairs, the execution time worsened for higher number of executor cores. For the 7 nodes configuration on the other hand, it was observed that the lowest execution time was obtained for the case where the number of executor cores was 8 for 7/512MB and conversely, 1 and 2 executor cores provided the lowest execution time for the 7/1GB and 7/2GB pair. This observation illustrated that it was unnecessary to use higher configurations for certain file sizes and node configurations and that, conclusively, the best results were attained for lower file sizes that have a higher number of executor cores while maintaining the executor memory at a default value, 1GB. Contrary to this conclusion, it was shown that for larger file sizes, a low number of executor cores provided low execution times. Additionally, with respect to the variation of the executor memory in **Table 5**, it was observed that higher executor memory provided better execution time across all configurations, but to a certain extent. Elaborating on this, by referring to **Table 5**, for the pairs 3/512MB, 3/1GB, 3/2GB, 7/512MB and 7/1GB, the highest number of executor memory provided the best results in terms of execution time and in

some cases, the best average utilization, as was observed for the 3/512MB and 7/512MB. However, for 7/2GB, it was shown that a low value of executor memory in comparison provided the best execution time. Furthermore, by comparing the changes between the execution time respective to a certain pair for varying executor memory in **Table 5**, such as 3/1GB as an example, the change in the execution time when the executor memory was increased from 1GB to 2GB was approximately 5 seconds, whereas a decrease of about 9 seconds was observed during the increase of executor cores from 1 to 2. This observation thus presented another conclusion – the changes made to the executor memory had less of an impact in comparison with the changes made to the number of executor cores. The aforementioned conclusions led to the notion that the execution time and average utilization of a pair depends on the value of certain parameters, where the selection of the number of executor cores or executor memory would provide the best results in terms of execution time and average utilization for a pair, all of which can be used as a form of guidance for developers and researchers to optimize the performance of a WordCount application, since the subjection of another big data application can present different results, by appropriately selecting the parameters respective to the observations made. However, the selection of the configuration parameters in this project facilitated the analysis of the application on a small scale, hence, as part of the future directions in order to further improve the results obtained and to draft extensive conclusions, the file size range respective to the text file to be analyzed can be increased so that the analysis will transform to that of a large scale one, as will be discussed in the **Future Work** section. As a whole, these conclusions, along with the analysis of a single big data application as opposed to the analysis of many listed in the literature works, are what sets this project apart from the related ventures done on performance evaluation using Apache Spark.

Overall, this project demonstrated a form of tradeoff between the execution time and average utilization, where an increase in the number of nodes and executor memory provided better average utilization compared to a lower number of nodes and different executor core values and longer execution times. It was further observed that altering the executor memory had less of an effect on the overall performance of the application—the execution time for some configurations decreased by less than 10 seconds approximately and the average utilization per configuration decreased by about 1% to 2%. The configurations facilitated the indication that the performance was dependent on the selection of the configurations chosen for the parameters for a pair. However, it is through these configurations that a minimal average utilization of 2% was attained. This observation and conclusion prompts future directions in order to achieve better results by pursuing a variety of routes and options, as shown in the following section.

5.1 Future Work

In order to further improve the results obtained in **Tables 4 and 5**, several approaches can be attempted as a means of future directions for this project, as shown below:

- Since the default configuration of a cluster is set to static allocation, dynamic allocation is an option worth exploring for the purpose of observing any significant changes in the performance as opposed to minute ones.
- A different hardware configuration of the cluster apart from m3.xlarge can be attempted to view any improvements on the end result, since various clusters have different numbers of virtual CPUs and could potentially lead to better performance by changing the cluster configuration.

- Broader range of configuration parameters with respect to the file size, for example, some file size configurations can be in the form of 4GB, 8GB and 12GB.
- Calibrating a degree of parallelism per configuration may lead to a performance improvement.
- Incorporating partitions.

Through the use of one, more or all of the approaches above, the performance can improve in comparison. [34] briefly discussed how the number of partitions per core can imply a speedup depending on the application, most in particular when the application involves intensive shuffling. Additionally, the parallelism configuration parameter of Spark, spark.default.parallelism, which involves the specific number of tasks to be used in the event that a partition parameter is unspecified, can be used to further tune Spark in a beneficial manner to provide improved results [35]. However, determining the parallelism value can prove to be difficult to determine, as discussed in [35]. Another route that can be taken relates to the change in the cluster configuration, as different clusters have different amounts of virtual CPUs and network performance rates. In the case of this project, a cluster of type m3.xlarge was used, consisting of 4 virtual CPUs [36], which is an outdated cluster. To further improve the performance, a cluster of type m5.2xlarge for example can be used, which consists of 8 virtual CPUs[37]. In spite of a potential performance change by modifying the cluster to be used, a higher memory utilization rate can be obtained as a result of the m5.2xlarge having a memory of 32GB, which is slightly more than twice the amount than that of the m3.xlarge cluster.

As observed above, several expenses can be acquired depending on the option taken. In this scenario, the vital question worth delving into is the following - which option will provide better results despite the tradeoffs present? Is it worth it to trade the memory utilization for better

performance? Or is it worth consuming time to determine the optimal parallelism parameter that would provide heightened statistics in the result? The answers to these questions form the future directions of this project.

Appendix A Python WordCount algorithm [38]

```
import sys
from operator import add
from pyspark import SparkConf, SparkContext

conf = (SparkConf()
        .setMaster("local")
        .setAppName("WordCounter")
        .set("spark.executor.memory", “2g"))
sc = SparkContext(conf = conf)

print("Launch App..")
if __name__ == "__main__":
            print("Initiating main..")

            inputFile = "s3://jaycabucket/data/512.txt"
            print("Counting words in ", inputFile)
            lines = sc.textFile(inputFile)

            lines_nonempty = lines.filter( lambda x: len(x) > 0 )
            counts = lines_nonempty.flatMap(lambda x: x.split(' ')) \
                        .map(lambda x: (x, 1)) \
```

```
            .reduceByKey(add)
output = counts.collect()
for (word, count) in output:
    print("%s: %i" % (word, count))


sc.stop()
```

Bibliography

[1] N. Nguyen, M. M. H. Khan, Y. Albayram, and K. Wang. "Understanding the influence of configuration settings: An execution model-driven framework for apache spark platform." In 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), pp. 802-807. IEEE, 2017.

[2] D. Chettiar, A. Das, O. Das: Performance Modeling of Cloud-based Web Systems to Estimate Response Time Distribution. Workshop on Software Architectures for Adaptive Autonomous Systems (SAAAS 2016) colocated with ISEC 2016, Goa, India, February 2016, pp. 41-46.

[3] O. Das, A. Das: Estimating Response Time Percentiles of Cloud-based Tiered Web Applications in presence of VM failures. 12th International ACM SIGSOFT Conference on the Quality of Software Architectures (QoSA 2016), Venice, Italy, April 2016, pp. 1-10.

[4] A. Das and O. Das: Effect Of Human Learning On Performance Of Cloud Applications. 10th IEEE International Conference on Cloud Computing (CLOUD 2017), Hawaii, USA, June 2017, pp. 778-781.

[5] I. Mavridis and H. Karatza, "Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark," Journal of Systems and Software, vol. 125, pp. 133–151, 2017.

[6] J. Veiga, R. R. Exposito, X. C. Pardo, G. L. Taboada, and J. Tourifio, "Performance evaluation of big data frameworks for large-scale data analytics," 2016 IEEE International Conference on Big Data (Big Data), pp. 424-431, Dec. 2016.

[7] O. Yildiz and S. Ibrahim, “On the Performance of Spark on HPC Systems: Towards a Complete Picture,” Supercomputing Frontiers Lecture Notes in Computer Science, pp. 70–89, 2018.

[8] L. Liu, “Performance comparison by running benchmarks on Hadoop, Spark, and HAMR,” UDSpace Home, 2015. [Online]. Available: http://udspace.udel.edu/handle/19716/17628. [Accessed: 15-Apr-2019].

[9] D. Cheng, J. Rao, Y. Guo, C. Jiang, and X. Zhou, “Improving Performance of Heterogeneous MapReduce Clusters with Adaptive Task Tuning,” IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 3, pp. 774–786, Mar. 2017.

[10] “What is Hadoop?,” SAS. [Online]. Available: https://www.sas.com/en_ca/insights/big-data/hadoop.html. [Accessed: 15-Apr-2019].

[11] “Apache Hadoop,” Apache Hadoop. [Online]. Available: https://hadoop.apache.org/. [Accessed: 15-Apr-2019].

[12] Apache Hive TM. [Online]. Available: https://hive.apache.org/. [Accessed: 15-Apr-2019].

[13] “Apache Oozie Workflow Scheduler for Hadoop,” Oozie, 14-Feb-2019. [Online]. Available: https://oozie.apache.org/. [Accessed: 15-Apr-2019].

[14] A. Oussous , F.-Z. Benjelloun , A. A. Lahcen, and S. Belfkih, “Big Data technologies: A survey,” *Journal of King Saud University - Computer and Information Sciences*, vol. 30, no. 4, pp. 431–448, Jun. 2017.

[15] N. C. TOKALA, “ADVANTAGES OF HADOOP,” International Journal of Scientific & Engineering Research, vol. 6, no. 1, pp. 2134–2135, Jan. 2015.

[16] A. Talwalkar, “HadoopT - breaking the scalability limits of Hadoop,” Thesis, Rochester Institute of Technology, 2011. [Online]. Available:

[https://scholarworks.rit.edu/cgi/viewcontent.cgi?referer=https://www.google.ca/&httpsredir=1&article=1470&context=theses]. [Accessed: 15-Apr-2019].

[17] V. N. Inukollu, S. Arsi, and S. R. Ravuri, "Security Issues Associated with Big Data in Cloud Computing," International Journal of Network Security & Its Applications, vol. 6, no. 3, pp. 45–56, May 2014.

[18] M. A. Memon, S. Soomro, A. K. Jumani, and M. A. Kartio, "Big Data Analytics and Its Applications," ResearchGate, vol. 1, no. 1, pp. 45-54, 2017.

[19] "Apache Spark™ - Unified Analytics Engine for Big Data," Apache Spark™ - Unified Analytics Engine for Big Data. [Online]. Available: https://spark.apache.org/. [Accessed: 15-Apr-2019].

[20] R. Guo, Y. Zhao, Q. Zou, X. Fang, and S. Peng, "Bioinformatics applications on Apache Spark," *GigaScience*, pp. 1-10, Aug. 2018.

[21] Z. Han and Y. Zhang, "Spark: A Big Data Processing Platform Based on Memory Computing," 2015 Seventh International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), pp. 172-176, Dec. 2015.

[22] Powered By Spark | Apache Spark. [Online]. Available: https://spark.apache.org/powered-by.html. [Accessed: 15-Apr-2019].

[23] S. Tang, B. He, C. Yu, Y. Li, and K. Li, "A Survey on Spark Ecosystem for Big Data Processing," pp. 1-21, Nov. 2018.

[24] A. G. Shoro and T. R. Soomro, "Big Data Analysis: Ap Spark Perspective," Global Journal of Computer Science and Technology: C Software & Data Engineering, vol. 5, no. 1, pp. 7–14, 2015.

[25] “Cluster Mode Overview,” *Cluster Mode Overview - Spark 2.4.2 Documentation*. [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview.html. [Accessed: 06-May-2019].

[26] “Cluster Mode Overview,” Cluster Mode Overview - Spark 2.4.1 Documentation. [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview.html. [Accessed: 15-Apr-2019].

[27] “Ganglia Monitoring System,” Ganglia Monitoring System RSS. [Online]. Available: http://ganglia.sourceforge.net/. [Accessed: 15-Apr-2019].

[28] Ganglia:: ClusterUY Grid Report. [Online]. Available: https://www.cluster.uy/ganglia/?cs=&ce=&m=load_one&tab=m&vn=&hide-hf=false. [Accessed: 15-Apr-2019].

[29] M. Massie, B. Li, B. Nicholes, and V. Vuksan, Monitoring with Ganglia, 1st ed. Sebastopol, CA: OReilly, 2012, pp. 181-184.

[30] M. L. Massie, B. N. Chun, and D. E. Culler, “The ganglia distributed monitoring system: design, implementation, and experience,” Parallel Computing, vol. 30, no. 7, pp. 817–840, 2004.

[31] “What is AWS? - Amazon Web Services,” Amazon. [Online]. Available: https://aws.amazon.com/what-is-aws/. [Accessed: 15-Apr-2019].

[32] “Configure Spark,” Amazon. [Online]. Available: https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-configure.html. [Accessed: 15-Apr-2019].

[33] “Job Scheduling,” Job Scheduling - Spark 2.4.1 Documentation. [Online]. Available: https://spark.apache.org/docs/latest/job-scheduling.html. [Accessed: 15-Apr-2019].

[34] R. Tous, A. Gounaris, C. Tripiana, J. Torres, S. Girona, E. Ayguade, J. Labarta, Y. Becerra, D. Carrera, and M. Valero, “Spark deployment and performance evaluation on the MareNostrum

supercomputer," 2015 IEEE International Conference on Big Data (Big Data), pp. 299-306, 2015.

[35] A. K. Paul, W. Zhuang, L. Xu, M. Li, M. M. Rafique, and A. R. Butt, "CHOPPER: Optimizing Data Partitioning for In-memory Data Analytics Frameworks," 2016 IEEE International Conference on Cluster Computing (CLUSTER), pp.110-119, 2016.

[36] "Previous Generation Instances," Amazon. [Online]. Available: https://aws.amazon.com/ec2/previous-generation/. [Accessed: 15-Apr-2019].

[37] "Amazon EC2 M5 Instances - general purpose compute workloads," Amazon. [Online]. Available: https://aws.amazon.com/ec2/instance-types/m5/. [Accessed: 15-Apr-2019].

[38] Aliga8or, "Aliga8or/csds-spark-emr," *GitHub*, 13-Apr-2017. [Online]. Available: https://github.com/Aliga8or/csds-spark-emr. [Accessed: 21-Apr-2019].