Performance Analysis of Big Data Frameworks on Virtualized Clusters

Amil Ahmad Ilham
Department of Informatics
Hasanuddin University
Makassar, Indonesia
amil@unhas.ac.id

Muhammad Niswar
Department of Informatics
Hasanuddin University
Makassar, Indonesia
niswar@unhas.ac.id

Andi Muhammad Ryanto
Department of Informatics
Hasanuddin University
Makassar, Indonesia
contact.amrjo@gmail.com

***Abstract*—Research on Big Data applications has become increasingly important for institutions and researchers worldwide. This trend is triggered by the increasingly use of systems and devices that leads to generate massive of electronic data each day. The implementation of conventional algorithms has been considered to be less efficient on managing and processing large datasets. In Big Data computation, Hadoop and Apache Spark are two open source frameworks that are commonly used and run on physical clusters. Since running these frameworks on a physical cluster costs more energy and rigid in management, in this research we evaluated their performance on virtualized clusters. Virtualization technology offers flexibility on managing cluster by sharing the resources to multiple instances. Our experiments show that in general Apache Spark is about 2-9 times better in execution time and throughput compared with Hadoop running on a virtualized environment.**

Keywords—virtualization, Big Data, frameworks, execution time, throughput

I. INTRODUCTION

In the last decade, the amount of electronic data has growing in a large scale. According to the an IDC on 2013, the size of world data that generated was 4.4 zettabytes (1 zettabytes ≈ 10^{21} bytes) and is forecasting to grow tenfold times to 44 zettabytes by 2020 [1]. This phenomenon will present challenges not only on managing and storing massive volume of data, but also to perform large scale data processing.

Hadoop [2] is a framework for distributed programming which capable on managing large datasets with simple programming model called MapReduce that allows to process and analyze the data in parallel manner. Spark [3] framework proposed by enabling to perform more kind of computation such as interactive applications and stream data processing by enhancing the MapReduce model and came through the in-memory concepts and techniques. Both of the frameworks are recommended to run on physical servers.

However, designing a physical cluster for big data computation and analysis is not easy. Most of failures are caused by misconfigurations like permission and option configuration in operating system [4]. Virtualization technologies become a solution by giving scalability, reliability, availability and better performance through approach on multiple layers [5]. Virtualization technologies allow resource sharing and allocating of a physical machine into multiple environments. In [6] Apache Hadoop is built atop NAS Storage and utilizing iSCSI, perform ETL (Extract, Transform, and Load) process and then analyze the results. Virtualization provides more flexibility on building clusters.

In this research, we evaluate and analyze the performance of both big data frameworks based on their I/O throughput and average execution time when running on a virtualized environment. We used Wordcount and TestDFSIO benchmarks to compare the performance of the two frameworks.

II. TECHNOLOGICAL BACKGROUND

A. Hadoop

Hadoop is a framework for processing and storing large data sets through cluster computing by a simple programming model on distributed manner. Hadoop is an open-source framework which is being developed by Apache Software Foundation for reliable, fault tolerance, scalable, and distributed computing [2].

There are two main components in Hadoop. The first component is Hadoop Distributed File System (HDFS) which is a file system that works in a distributed way. The second is MapReduce which is a distributed programming model and a system for jobs scheduling and resource management called YARN [7].

1) HDFS is a filesystem that manages Hadoop to be capable running on commodity platforms. It is fault-tolerant and requires low hardware spefecification to run. It adopts a master and slave architecture model. Figure 1 shows a single HDFS cluster. Only one single namenode exists. It acts as a master, controlling the namespaces and the metadata of file. It also controls slaves when accessing a file. Several datanodes manage storages attached to nodes. HDFS discloses the namespace of filesystem. It permits files to be stored in the filesystem. A file then splitted into blocks with specified size internally and store them in the all datanodes distributedly. Some operations of the namenode on the file system are opening, renaming, and closing files and directories. It also decides the mapping of blocks to datanodes which are responsible for serving the read and write requests from the file system's clients. The datanodes also execute block

replication, creations and deletion upon instruction from the namenode [8] which make it fault-tolerants.

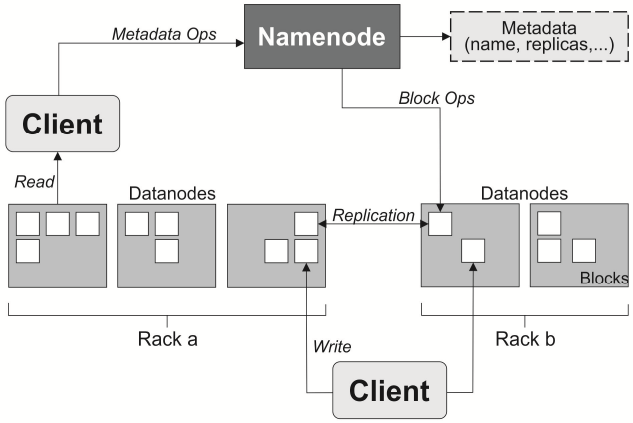


Figure 1: An HDFS Architecture

2) MapReduce [9] is a programming tool designed for generating and processing big data. It has a capability to process large volume of data on thousand of computing nodes in one cluster, handling failures, duplicating tasks, and aggregating results. MapReduce computation (Figure 2) proceeds a set of *input* key and value pairs to yield a set of *output* key and value pairs. It consists of two functions, a user-defined map function and a user-defined reduced function. The process can be done in parallel on multiple machines. The map function takes a key and value pair as input, applies the user-specific code, and produce intermediate results. These results are then aggregated by the reduce function to output the final results.

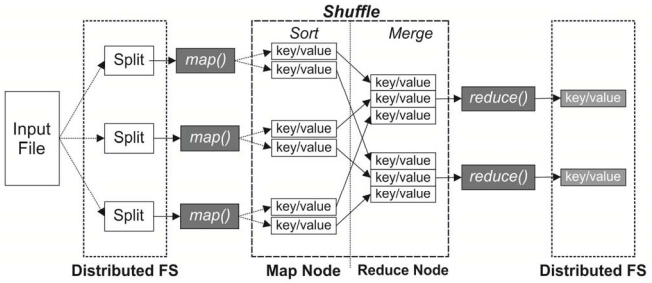


Figure 2: MapReduce Model

3) Yet Another Resource Negotiator (YARN) is a system to manage Hadoop's cluster resources. Hadoop 2 implements YARN to improve the MapReduce performnace. YARN provides its services by two daemons, a resource manager which manages the resources across the cluster and a node manager which launches and monitors the container [1].

B. Spark

Spark is a big data framework designed to be fast and general-purpose for big data processing [3]. Spark emerges to overcome the deficiencies of Hadoop on iterative jobs and interactive analytics [10]. It extends the MapReduce model to support more computation types including interactive queries and stream processing. Spark provides high-level APIs in Scala, R, Python and Java. It also supports general execution graphs by optimizing an engine called Direct Acyclic Graph (DAG) [3]. Spark consists of several components: a main component that contains basic functionality of Spark called Spark Core, Spark's default cluster manager called the *Standalone Scheduler,* and higher-level libraries to perform various types of workloads (Spark SQL, Spark Streaming, MLlib and GraphX) [11]. Spark supports many file systems for reading and writing task like HDFS, Amazon S3, Cassandra, or in a local file system.

The Spark main abstraction is a feature called *resilient distributed datasets* (RDDs). It holds data objects in memory for reducing overhead incurred by the operation of networks and disks. An RDD represents a read-only collection of objects partitioned across a set of machines. It can be rebuilt if the partition is lost [10]. We can split each RDD into multiple partitions. We may also compute it on different nodes of the cluster. RDD performs an operation called *Transformations* which build a new RDD from a previous one e.g. filter() which returns a new RDD of filtered RDD. It also performs operation called *Actions* which calculate a result based on an RDD. It returns the value either to the driver program or saves it to the file system e.g. count() [11].

Spark has a feature called *in-memory computation.* This concept allows Spark to store data in RAM. This feature gives performance improvement by an order of magnitudes by keeping the data in memory. Spark also has a feature called *lazy evaluation* which not start the execution until an action operation is triggered [11].

C. Proxmox Virtual Environment

Proxmox Virtual Environment (VE) is an open source-tool based on Debian GNU/Linux distribution. Proxmox is designed for managing virtual machines, containers, virtualized networks, storage, and high-availability clustering. It has a well-design that can be managed using a web-based or a command-line interface. Proxmox VE is a project developed by Proxmox Server Solutions GmbH and licensed under GPLv3. Proxmox VE is recognized as a type 2 hypervisor because the virtualization layer is built upon an operating system. Proxmox VE v4.2 provides LXC, KVM and QEMU virtualization technologies through a web-based interface. Proxmox VE is considered as *type 2 Hypervisor* [12].

III. Experimental Method

A. System Overview

To analyze the performance of both frameworks on a virtualized cluster, we established Proxmox VE virtualized environment. We setup the environment on Intel Core i7-4770 of 3.40 GHz (with 4 cores, hyper-threading and Intel VT technology) processor, 16 GB of RAM, 1 x 1TB of storage and one network interface card. The Proxmox VE version used is 4.2.

We run our experiments on two types of clusters. Type 1 cluster is a single-node cluster which consists of one virtual machine with configuration: 6 vCPUs and 12 GB of vRAM.

Type 2 cluster is a multi-node cluster using a configuration of 3 VMs, 1 as a master and 2 as slaves. The environmental set up is 2 vCPUs and 4 GB of vRAM for each VMs. All VMs are using CentOS 7 as operating system and KVM as a virtualization technology.

We used Hadoop 2.7.3, Spark 2.2.0, and Oracle Java Development Kit 8u121 in all virtual machines. For experiment in a single-node cluster, we deployed pseudo-distributed mode. For experiment in multi-node cluster, we deployed fully-distributed mode. The HDFS block size was 128MB. Hadoop is using YARN as resource manager and Spark is using its default Standalone Scheduler.

B. Benchmarks

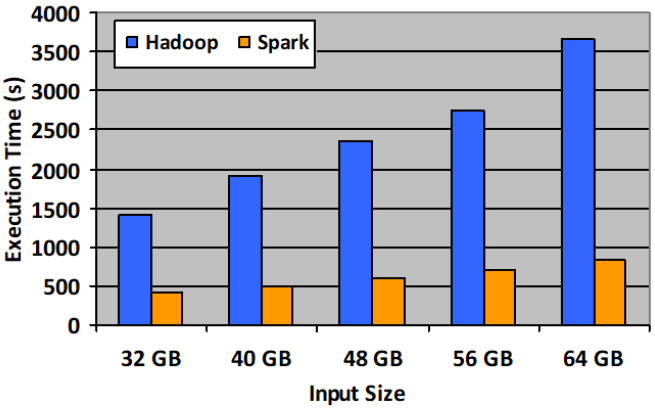
1) Wordcount: A benchmarking program that takes a text file as input. It then processes and computes the number of occurences of each word in a file. The results are written in the output file. This file which is a text file contains a word and the number of times it occurs. It is a mapper that proceeds a line as input and breaks it into words. It then emits a key and value pair of word as the key and assign the value to 1. The reducer then sums the counts for each word. It emits a single key and value pair with the word and sum [13]. On Hadoop, the Wordcount is already provided by default installation and on Spark by adapting RDD API Examples from [14]. We used this benchmark to compare the *execution time* measured in seconds (s) for the two frameworks.

2) TestDFSIO: A benchmarking program for testing the read/write performance of HDFS. The goal of TestDFSIO benchmark is to test the parameter of NameNode and DataNodes for discovering the performance bottleneck in network. It evaluates the cluster I/O rate by performing a stress testing on the HDFS. It consists of two tests. The first one is a write test. This test is performed by generating and writing the files in the HDFS environment. The second one is a read test. This test is performed by reading the files created by the write test while measuring it's performance. The prosess of reading or writing each file is performed in separate map task, and the output is collected for measuring the performance of the process. The performance measurements are collected in the reduce task to yield the summary of the test. In our experiments, we built both Hadoop and Spark on the same filesytem which is HDFS cluster. Using TestDFSIO is helpful to find out which framework gives HDFS better performance in terms of I/O by stress testing on it. In Hadoop, TestDFSIO application is already included in its default installation. However, TestDFSIO is not provided as default for Spark. Its source codes have been adapted from [15]. We set the TestDFSIO to create 5 files and 6 files of 8GB respectively and store it in HDFS. We used this benchmark to compare the *throughput* measured in megabytes per seconds (MB/s) for the two frameworks.

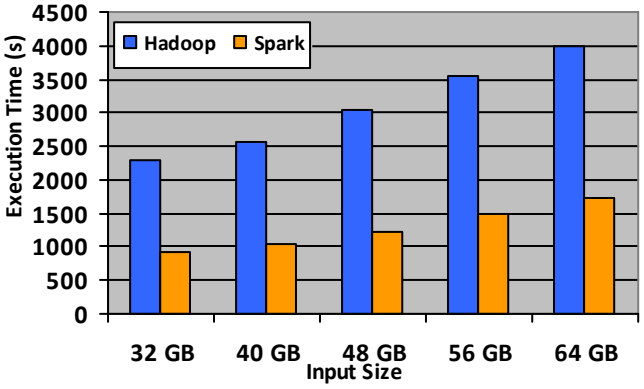
IV. Results and Discussion

A. Execution Time

Figure 3 shows the execution time of Wordcount for various input sizes, using both Spark and Hadoop on a single-node cluster (type 1 cluster) and on a multi-node cluster (type 2 cluster). Spark outperformed Hadoop 3x-4x faster on the type 1 cluster and 2x faster on the type 2 cluster for 32, 40, 48, 56, and 64 GB input sizes respectively. Spark executes Wordcount faster than Hadoop due to in-memory computation feature of Spark. This feature allows Sparks to hold the *intermediate data* from the map phase in the Random Access Memory (RAM) before passing it to the reduce phase while Hadoop persists to return it back to disk which will cost more time.



(a)



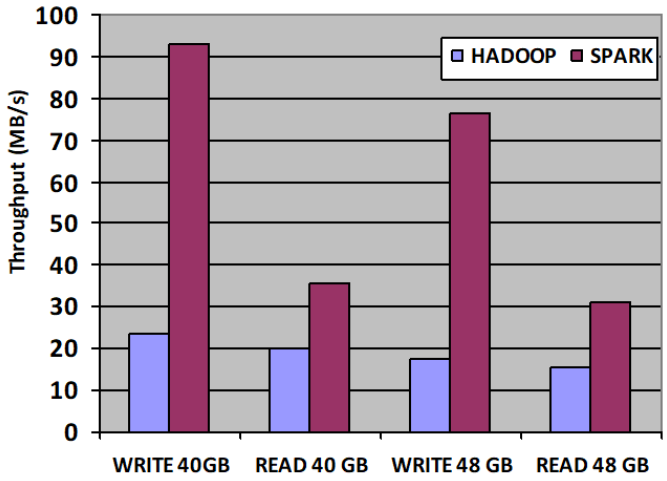
(b)

Figure 3 Wordcount Execution Time on the (a) Type 1 Cluster (b) Type 2 cluster

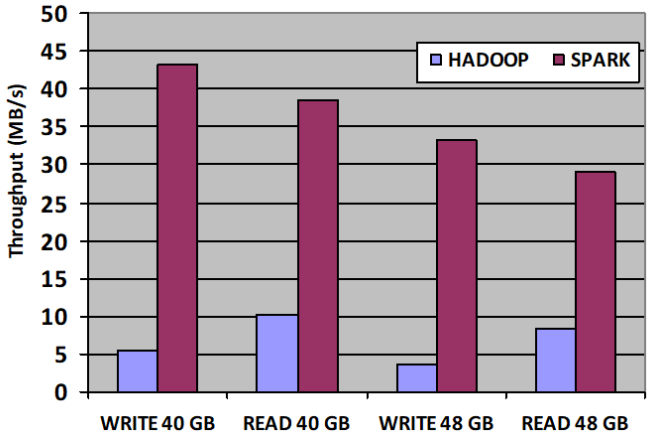
B. Throughput

Figure 4 shows the throughput of TestDFSIO using both Spark and Hadoop on the type 1 cluster and on the type 2 cluster. Spark gives better throughput in performing *write* and *read* tasks than Hadoop on the two types of clusters for each 5 files and 6 files of 8GB. Spark gives better throughput about 4x in write task and about 2x in read task than Hadoop on the type 1 cluster. On the type 2 cluster, the throughput of Spark is

up to 9x better in write task and about 3.5x in read task compared with Hadoop.



(a)



(b)

Figure 4 : TestDFSIO Throughput on the (a) Type 1 Cluster (b) Type 2 Cluster

Spark obtains better results due to the direct acyclic graph feature. This feature optimizes the *execution plan* by utilizing the cache and reducing the number of read/write processes which saving more time.

V. CONCLUSION

This paper evaluates and analyzes the performance of two big data frameworks, Hadoop and Spark, based on their average execution time and I/O throughput when running on a single-node cluster and on a multi-node cluster in virtualized environment. Using Wordcount benchmark, the experiments show that Spark outperformed Hadoop 3x-4x faster on a single-node cluster and 2x faster on multi-node cluster. Using TestDFSIO benchmark, Spark gives better throughput about 4x in write task and about 2x in read task on single-node cluster, about 9x better in write task and about 3.5x in read task on a multi-node cluster compared with Hadoop.

Spark provides better execution time compared with Hadoop by taking advantages of processing data in-memory while Hadoop persists to return it back to slow disk drives after the map or reduce phase. Spark optimizes the *execution plan* to get better throughput by utilizing the cache and reducing the number of read/write processes which saving more time. However, since Spark is relying on processing data in-memory and caching them for a while, it needs a lot of memory allocation for demanding good performance.

Our plan in future work is to add some cluster types and do experiments with more workload types to perform.

REFERENCES

[1] T. White, Hadoop: The Definitive Guide, 4th ed. O'Reilly, 2015.

[2] "Apache Hadooop." [Online]. Available: http://hadoop.apache.org.

[3] "Apache SparkTM - Lightning-Fast Cluster Computing." [Online]. Available: http://spark.apache.org/.

[4] A. Rabkin and R. H. Katz, "How Hadoop Clusters Break," IEEE Softw., vol. 30, no. 4, pp. 88–94, 2013.

[5] D. Kusnetzky, Virtualization: A Manager's Guide. O'Reilly, 2011.

[6] Adnan, A. A. Ilham, and S. Usman, "Performance analysis of extract, transform, load (ETL) in apache Hadoop atop NAS storage using ISCSI," in Proceedings of the 2017 4th International Conference on Computer Applications and Information Processing Technology, CAIPT 2017.

[7] D. DeRoos, P. C. Zikopoulos, R. B. Melnyk, B. Brown, and R. Coss, Hadoop for dummies. 2014.

[8] "HDFS Architecture Guide." [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[9] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Commun. ACM, vol. 51, no. 1, p. 107, 2008.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark : Cluster Computing with Working Sets," HotCloud'10 Proc. 2nd USENIX Conf. Hot Top. cloud Comput., p. 10, 2010.

[11] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, Learning Spark. O'Reilly, 2015.

[12] R. Goldman, Learning Proxmox VE. Packt Publishing, 2016.

[13] "Apache Hadoop, WordCount - Hadoop Wiki." [Online]. Available: https://wiki.apache.org/hadoop/WordCount..

[14] "Apache Spark - Examples." [Online]. Available: https://spark.apache.org/examples.html

[15] "BBVA/Spark-nenchmarks: Benchmarking suite for Apache Sparks." [Online]. Available: https://github.com/BBVA/spark-benchmarks.