Spark Performance Optimization Analysis In Memory Management with Deploy Mode In Standalone Cluster Computing

1st Deleli Mesay Adinew 2nd Zhou Shijie†and 3rdYongjian Liao‡
*School of Information and Software engineering
*University of Electronic Science and Technology College, Chengdu, China.

***Abstract*—As data is growing in different dimensions, it is difficult to get appropriate data analytic tools. Spark is one of high speed "in-memory computing" big data analytic tool designed to improve the efficiency of data computing in both batch and real-time data analytic. Spark is memory bottleneck problem which degrades the performance of applications due to in memory computation and uses of storing intermediate and output result in memory. Investigating how performance is increased in relation to spark executor memory, number of executors, number of cores, and deploy mode parameters configuration in a standalone cluster model is our primary goal. Three representative spark applications are used as workloads to evaluates performance in relation to changing these parameters value. Experimental result show, submitting the job in cluster deploy mode is faster to finish than a submitting job in client deploy mode under two workloads. This implies spark performance does not depend on deploy mode rather it depends on types of application. However,increasing number of executor per worker, a number of core per executor and memory fraction will increase spark performance under all workloads in any deploy mode.**

***Index Terms*—Spark Performance Tuning, Parameter configuration, Standalone cluster, Deploy mode, Memory Management, Performance improvement**

I. Introduction

As the internet became popular, data grows exponentially in volume which becomes known as big data. Big data need better management and proper data analytic tools to make more effective decision and better benefit in different organizational and industrial area. It is a technique used in different organizations and industry to extract the most and better valuable information that supports better decision making processes[4].Hadoop is one of the most popular big data analytic platform which has its own limitation. Apache Spark is one of recently designed big data analytic tool to overcome certain Hadoop's limitation.It emerged as good fault tolerance and support a distributed computing framework designed based on principles of Hadoop MapReduce algorithms which uses in "memory computation" to improve the efficiency of data computing to minimize disk read to reduce data loading latency[4]. It is well suitable for iterative applications used in both batch and real-time which has more than 180 configuration parameters which helps to optimize its performance according to user's requirements[2]. It's performance is depend on combinations of these configuration parameters values [4]. Memory management and other parameters such as a number of executors, a number of cores, and deploy mode are one of the parameters need to investigate whether it improves performance of the application or not.

The contribution of this paper is described as follows.

- We studied big-data analytic approaches and tools with its deployment techniques.
- We identified and selected appropriate big data analytic tools suitable for data analytic.
- We propose those selected parameters to improve spark performance.
- We confirm spark performance doesn't depend on job submission mode rather it depends on types of workloads. values.
- We confirm deploy mode does not affects spark performance.
- We measure and confirm increasing execution memory fraction, a number of executors per Worker and number of cores per executor increase spark performance under any deploy mode regardless of applications workloads.

This paper is organized as follows. Section 2 Background. Section 3 Experimental Methodology. Section 4Environmental and configuration setup. Section 5 Experimental Result and Discussion. Section 6. Related work. Section 7 Conclusion.

II. Background

In this section, we will discuss big data computational tools including their limitation and usage

A. Hadoop

In the past few years, Hadoop is well known as most dominant platforms for structural Big Data analytic using MapReduce computation model[13,16]. It is designed to scale up the performance of commodities machines with a very high degree of fault tolerance and efficient. However, analyzing big data is still a challenging job, unless it is analyzed properly. Hadoop framework faces performance bottlenecks for iterative machine learning algorithms and interactive data queries [3].

B. Apache Spark

Spark is high-speed computing framework which use in "memory computation" to improve the efficiency of data computation, designed as fault tolerant and more powerful for

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00242

big data Analytic which is designed based on the principles of Hadoop MapReduce algorithms. However, unlike MapReduce, the intermediate and output results of its jobs can be stored in memory[1] and more suitable for iterative applications and real-time data processing[5,7].

C. Spark performance Model

Although, spark has high processing speed, it requires optimization framework due to daily increasing data in volume. Framework is designed and defined as follows[4].
Spark Perf =F (A, D, R, C) where A denotes the user's application, D denotes the input data, R denotes the resources, and C denotes the configuration parameters of Spark platform, F is a function that is performed on A, D, R, and C parameters. Even-though, Spark performance depends on the above parameters, Spark performance bottlenecks are moved to resources utilization such as CPU, memory, and network communication layer that requires to adjusting settings to memory, cores, and instances to guarantees that the Spark has optimal performance and prevents resource bottle-necking during run-time of an application. Proper Spark tuning can ensure the following properties. Proper use of all resources in an effective and efficient manner will support:

- Avoid those jobs that run a long time in the execution of tasks.
- Improves the performance time of the system in the execution of tasks.
- Guarantees jobs are on a correct execution engine in the process of execution activities.

1) Memory management : is grouped into two categories execution and storage memory[6]. The execution memory is usually used for handling computation in shuffles, joins, sorts and aggregations, while the storage memory is usually used for handling the caching and propagating internal data across the cluster of spark. In Spark, execution and storage memory share a unified region (M) in the case when one does not require more space, that means when execution memory is no more used(free), storage can acquire all the available memory and vice versa[9]. if necessary, sometimes execution may enforce to drive out storage until total storage memory usage or capacity falls under a certain threshold(R)[6].

D. Spark Memory Management Policy

Spark has two memory area named execution and storage memory which used for computation in shuffles, joins, sorts and aggregations, and for caching and propagating internal data across the cluster. In spark, execution, and storage memory share a unified memory region which means when no execution memory is used, storage can acquire all the available memory and vice versa. If necessary, execution may force moving out storage until total storage memory usage falls under a certain threshold[6,14]. This requires managing the efficient utilization of memory between the two fractions. The best way to size the amount of memory consumption transforming datasets into RDD, like easily put into cache, and look at the storage[6,15]. We can handle RDDs affect on the performance of workloads by changing memory Faction ratio and changing different resource configuration of spark executors. Different spark executors parameter and deploy mode is selected with a different experimental approach to investigate how memory and other parameters are efficiently utilized under different

III. Experimental Methodology

To do our experiment, we identified different experimental approaches, tools, components and deployment approaches, methods, and parameters configuration as experimental criteria based on our experimental goal. Spark standalone cluster deployment model is selected as the best deployment approaches and methods. Environmental variable and parameters configuration was selected along with their deployment mode. Spark standalone cluster component interaction were identified as the Driver, the Master, the Cluster Manager, and the Executor(s), which run different Worker. The experimental model is designed as an architectural model to show its component interaction. Application interaction anatomy was also identified and designed as one of application interaction model that computer tasks in the model. Different spark workloads were identified, developed and used for evaluating performance under different deploy mode and parameters value.

IV. Environment and Parameter Configuration setup

This section describes the Environment variable and Parameters configuration setup that can be identified, configured, and used for our experiment.

A. Environment setup

Our experimental activities and results depend on environmental variables configuration so that it requires to identify and configure the environment variable. Our experimental environment is categorized as Hardware and Software environment. Hardware used for this experiment is a standalone laptop where the details are listed in table 1. Among software used for this experiment spark,python, scala, SBT, and IntelliJ idea are some of them that were installed on a standalone laptop where the details are listed1. Spark is installed and configured according to deploy the model for data analytics. Scala, SBT and IntelliJ idea are installed to support developing and writing different spark application that was used for doing this experiment.

TABLE I
Show Hardware and Software Configuration Environments

Hardware	Software
Memory :4GB Hard Disk :750GB Processor: Intel(R) Core(TM)i5-5200U CPU@2.20GHz	Window 10 OS with 64-bits Spark 2.4.0 with Hadoop 2.7 Python,Scala 2.11.12,JDK 11.0.1 SBT Intellij idea 2019.1

B. Parameter Configuration Setup

To evaluate our experiments we identify and configure spark parameters those appropriate parameters for our experiments. These selected spark parameters for this experiment were listed with the default and new value as spark parameters configuration in table 2.

TABLE II
SHOW SPARK PARAMETER CONFIGURATION WITH THE DEFAULT AND NEW VALUES

Catagories	**Parameters**	**Default Value**	New Value
Application Properties	spark.executor .memoryOverhead	1000M	2048M
	spark.executor.memory	1000M	2000M
Deployment Env't Run Modes	spark.submit.deployMode	None	Client
			Cluster
Memory Management	spark.shuffle.memoryFraction	0.2	0.9
	spark.storage.memoryFraction	0.6	0.9
Execution Behavior	spark.executor.cores	4	8
Runtime Environment	InitiatingHeap OccupancyPercent	45	85
	num-executors	1	8

C. Experiment

1) Procedure: To do this experiment we install and configure different software according to our experimental model. After the installation is completed, we developed different spark application workloads in Scala and create a jar file for each application. Once jar files are completed, we start one master server with two workers to connecting them to the Master via Master URL to create spark standalone cluster with one Master and two workers. The master launch a master daemon process for manages all the worker processes. The worker launches a worker daemon process which is responsible for communicating with the master and managing executors. Finally, we submit a different application jar as spark job to cluster with different parameters values and different deploy mode to measure performance spark related to these parameters configuration.

2) Architectural Overview: Our experimental architectural used for this research is shown in figure 1, which describe how to spark a standalone cluster communicates with each other. The interaction describes, as whole the cluster waiting to receive job requests from a client with different parameters configuration options to pass it into the master and worker as resources for executors. When submitting a job every client is required to specify as whole resource requirement such as how many executors, CPU, and memory is required for each executor. Spark scheduler then allocates resources to the client as wishes to run the job. Such kind of allocation requires every client to calculate resource volume carefully in order to run.

3) Performance Evaluation: Performance evaluation was done by categorizing our experimental view as performance in job submission mode and performance in parameters values

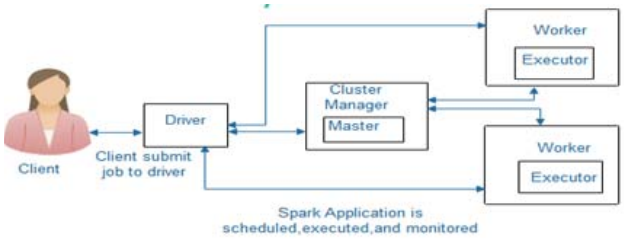

Fig. 1. Shows spark components in a standalone cluster with their interaction.

under different workloads and dataset.

1) Impact of job Submission Mode: Client vs Cluster Deploy Mode. The experiments were done by submitting different workloads by specifying –deploy-mode as client or cluster to evaluate performance difference in job submission mode.

2) Impact of changing parameters value: Default vs New value The experiments were done by submitting different workloads under default or new parameters value to evaluate spark performance in different parameters value.

In both the above cases experiments were done repeatedly with the same parameters values and deploy mode with the same workload, then average performance is calculated as the final performance of spark application under that workload.

4) Performance Improvement Calculation: We calculated performance improvement based on one of the principles used to calculate performance improvement which defined as

$$Perf.Improve = \frac{NewValue}{OldValue}) * 100\%$$

by assuming New value is the short response time than Old value.

i) Performance improvement rate in deploy mode can be calculated as

$$\frac{ClusterValue}{ClientValue} * 100\%$$

by taking change parameter as reference for those workloads that have shown higher value in cluster deploy mode but for Tera Sort we calculated as

$$\frac{ClientValue}{ClusterValue} * 100\%$$

because Tera Sort shows higher performance in client deploy mode. the result is shown in table 1. ii) Performance improvement rate among default and a new value can be calculated as

$$\frac{NewValue}{DefaultValue} * 100\%$$

by taking deploy mode as reference. the result is shown in table 2.

5) Workloads and Datasets: We used 3 representative spark applications as workloads to evaluate the impact of memory limitation and job submission mode on performance. The workloads selected for our experiments are implemented in spark machine learning algorithms named as WordCount, TeraSort, and PageRank to evaluate performance under different parameters value and deploy mode. Datasets used for

TABLE III
SHOW SPARK PARAMETER CONFIGURATION WITH THE DEFAULT AND NEW VALUES

Application	Parameter Value	Deploy Mode	Avg.Perf/sec in deploy mode	Perf.Improve
Page Rank	Default	Cluster	532.8	(533/2016)*100 =26.43%
		Client	2,016	
	New	Cluster	276	(276/1012)*100 =27.27%
		Client	1,012	
wordCount	Default	Cluster	12	(12)/35*100 =34.29%
		Client	35	
	New	Cluster	6.5	(6.5/11.2)*100 =58.04%
		Client	11.2	
TeraSort	Default	Cluster	160.8	(63/160.8)*100 =39.38%
		Client	63	
	New	Cluster	32.2	(28.2/32.2)*100 =87.58%
		Client	28.2	

TABLE IV
SHOW SPARK PARAMETER CONFIGURATION WITH THE DEFAULT AND NEW VALUES

Application	Deploy Mode	Parameters Value	Avg. Perf/sec	Perf.Improve.in Para. value change
Page Rank	Client	Default	2016	(1012/2016)*100 =50.2%
		New	1,012	
	Cluster	Default	532.8	(276/532.8)*100 =51.8%
		New	276	
wordCount	Client	Default	35	(11.2/35)*100 =32%
		New	11.2	
	Cluster	Default	12	(6.5/12)*100 =54.2%
		New	6.5	
TeraSort	Client	Default	63	(28.2/63)*100 =44.8%
		New	28.2	
	Cluster	Default	160.8	(32.2/160.8)*100 =20%
		New	32.2	

these experiments were accessed from different data sources such as Stanford dataset[9].

TABLE V
SHOW DATASET USED FOR PERFORMANCE COMPUTING

Workloads	Input size
PageRank	31.3MB
TeraSort	11KB
WordCount	4KB

V. EXPERIMENTAL RESULT AND DISCUSSION

A. Experimental Result

The experiment result show, different performance rate for different workloads under different deploy mode and parameters value. The result is visualized in different workload with

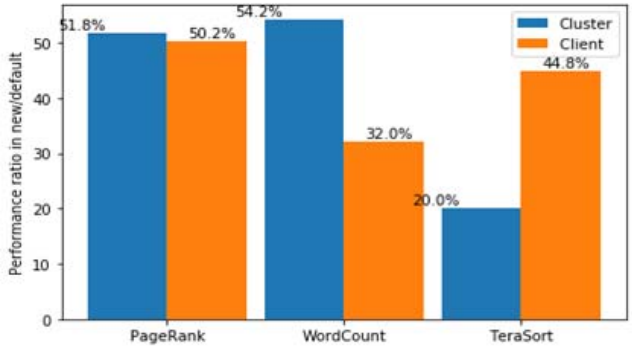

Fig. 2. Performance Comparision among deploy mode

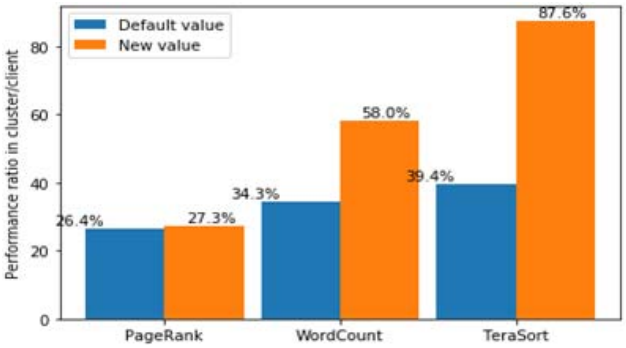

Fig. 3. Performance Comparision among parameter value: default vs new value

different deploy mode and parameters value. PageRank and WordCount application perform higher in cluster deploy mode than client mode regardless of parameter change whereas TeraSort performs higher in client deploy mode than cluster deploys mode regardless of parameter change. Increasing default value increase performance of all application depending on deploy mode.

1) Performance results in deploy Mode: This section describes performance improvement comparison between cluster and client deploy mode by taking default parameter value as a base reference. The result shows submitting the job in cluster deploy mode has high performance than a submitting job in client deploy mode in case of WordCount and PageRank. However, spark performance in TeraSort, client deploy mode has high performance than cluster deploy mode.

2) Performance result in changing parameters value: This section describes performance improvement comparison between default and new parameter value by taking deploy mode as a base reference. The result shows performance in new parameters configuration is better than default parameters values under both deploy mode in all workloads.

B. Discussion

The result shown in figure 3 indicate a performance improvement among deploy mode in both default and new values that means increasing default configuration parameters listed

in table 2 will increase performance of spark under both deploy mode in all workloads. The result shown in Figure 2 indicate performance of spark increased in cluster deploy mode under PageRank and WordCount but decreased for TeraSort. This implies performance in deploy mode depends on types of applications which is directly related to internal structure of the application.

VI. Related Work

Spark configuration parameters require to identify which combination increase performance in big data analytics. In[10] K. Zhang et al. proposed in MEMORY ONLY, DISK ONLY, OFF HEAP MEMORY, AND DISK for intermediate data caching in case of operating as RDD and DataFrame in K-means benchmark under different datasets size to increase performance. The result shows when using memory-only for intermediate data caching, serialization process is not executed in the RDD but the encoding is executed in the DataFrame that make RDD is faster than the DataFrame. However, the encoding of DataFrame is faster than serialization of RDD when storing the cache on disk or off-heap memory. In[11] L. Xu et al. MEMTUNE are proposed for Memory Management Policy which implements to strives as best utilization of memory resource that ensures MEMTUNE improves individual allocated memory utilization of each application. It uses storage memoryFraction to manage OutOfMemory and evaluate execution time of different workloads under the default Spark, MEMTUNE, MEMTUNE with prefetch only, and dynamic RDD cache tuning only in Logistic Regression, Linear Regression, Page Rank, and Shortest Path to study the impact of garbage collection and Cache Hit Ratio. In [12] H. Du et al. Proposed Otterman to parameters optimization approach based on Simulated Annealing algorithm and Least Squares method, which required to adapt them to the MapReduce paradigm to evaluate the impact of these parameters in the available resources such as CPU, Memory, and Disk in the cluster. In reference[11] L. Xu et al focus to manage OutOfMemory, which use garbage collection strategies to evaluate the performance of spark using different memory fraction under different workloads. In reference[12] H. Du et al don't focus to identify which parameters and job submission mode will affect spark performance and in which aspect. All the above papers don't focus on deploy mode and even use different methodology to evaluate in which way performance can be increased under different workloads. As we observe from the above reason, research work on the performance optimization of the Spark platform is still not addressed from a different aspect. This free space initiates us to focus on a certain group of parameters that were not covered with such methodology by any of them.

VII. Conclusion

Big data analytics requires the proper tools to extract valuable information. Spark is one of high-speed big data analytic tool with efficient resource utilization such as memory under different parameters configuration. Increasing default value of a number of executor per worker and number of core per executor, executors memory fraction increase spark application performance under all deploy mode and workloads. However, deploy mode doesn't determine spark performance rather it depends internal structure of the application.

References

[1] 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), 2015.

[2] A.-K. Koliopoulos, P. Yiapanis, F. Tekiner, G. Nenadic, and J. Keane, "Towards Automatic Memory Tuning for In-Memory Big Data Analytics in Clusters," 2016 IEEE International Congress on Big Data (BigData Congress), 2016.

[3] K. Grolinger, M. Hayes, W. A. Higashino, A. Lheureux, D. S. Allison, and M. A. Capretz, "Challenges for MapReduce in Big Data," 2014 IEEE World Congress on Services, 2014.

[4] V. M. Bande and G. K. Pakle, "CSRS: Customized service recommendation system for big data analysis using map reduce," 2016 International Conference on Inventive Computation Technologies (ICICT), 2016.

[5] Z. Matei, C. Mosharaf, D. Tathagata. "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing" NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation[C].2012, 2-

[6] Apache Spark™ - Unified Analytics Engine for Big Data," Apache Spark™ - Unified Analytics Engine for Big Data. [Online]. Available: https://spark.apache.org/. [Accessed: 29-Feb-2019].

[7] Z. Yang, D. Jia, S. Ioannidis, N. Mi, and B. Sheng, "Intermediate Data Caching Optimization for Multi-Stage and Parallel Big Data Frameworks," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018.

[8] "Stanford Large Network Dataset Collection." [Online]. Available: https://snap.stanford.edu/data/index.html. [Accessed: 10-Mar-2019].

[9] I. S. Choi, W. Yang, and Y.-S. Kee, "Early experience with optimizing I/O performance using high-performance SSDs for in-memory cluster computing," 2015 IEEE International Conference on Big Data (Big Data), 2015.

[10] K. Zhang, Y. Tanimura, H. Nakada, and H. Ogawa, "Understanding and improving disk-based intermediate data caching in Spark," 2017 IEEE International Conference on Big Data (Big Data), 2017

[11] L. Xu, M. Li, L. Zhang, A. R. Butt, Y. Wang, and Z. Z. Hu, "MEMTUNE: Dynamic Memory Management for In-Memory Data Analytic Platforms," 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2016.

[12] H.Du, P. Han, W. Chen, Y. Wang, and C. Zhang, "Otterman: A Novel Approach of Spark Auto-tuning by a Hybrid Strategy," 2018 5th International Conference on Systems and Informatics (ICSAI), 2018.

[13] J. Dean and S. Ghemawat, MapReduce: simplified data processing on large clusters" International Journal of Research and Engineering, vol. 5, no. 5, pp. 399–403, 2018.

[14] W.Guolu, X.Jungang, and H. Ben, "A Novel Method for Tuning Configuration Parameters of Spark Based on Machine Learning", 2016 IEEE 18th International Conference on High-Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems,2016

[15] Z. Zhu, Q. Shen, Y. Yang, and Z. Wu, "MCS: Memory Constraint Strategy for Unified Memory Manager in Spark," 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), 2017.

[16] D. M. Adinew, Z. Shijie, and Y. Liao, "Spark Performance Optimization Analysis in Memory Tuning On GC Overhead for Big Data Analytics," Proceedings of the 2019 8th International Conference on Networks, Communication and Computing, 2019