Journal of Technological Advancements
Volume 1 • Issue 1

Spark Performance Optimization Analysis With Multi-Layer Parameter Using Shuffling and Scheduling With Data Serialization in Different Data Caching Options

Deleli Mesay Adinew, UESTC, China & Dilla University, Ethiopia
Ayall Tewodros Alemu, UESTC, China & Dilla University, Ethiopia

ABSTRACT

As social networking services and e-commerce are growing rapidly, the number of online users is also dynamically growing. They contribute content to the digital world. In such a dynamic environment, meeting the demand of computing is very challenging especially with existing computing models. Although Spark is recently introduced to alleviate the problems with the concept of in-memory computing for big data analytics with many parameter configurations that allow the configuration and improvement of performance, it has a performance bottleneck which requires investigating the performance improvement mechanism by focusing on the combination of scheduling and shuffle manager with data serialization with intermediate data caching options. The standalone cluster computing model was selected as experimental methodology with submit command line for data submission. Three Spark applications, WorkCount, TeraSort, and PageRank, were selected and developed for experiment. As a result, 2.45% and 8.01% performance improvement are achieved in OFFHEAP and Memory Only Ser data caching options, respectively.

KEYWORDS

Big Data Analytic, In-Memory Computing, Performance Improvement, RDD Caching Mechanism, Scheduling, Shuffling

DOI: 10.4018/JTA.290326

1. INTRODUCTION

As the digital universe increase in size due to the daily transformation of everything related to people, enterprises, and environments to the digital universe, it becomes challenging to analyze it(H. Zhang et al., 2018). According to a study observed and predicted by IDC, the size of data in the digital universe will exceed around 44 zettabytes by 2020(Aggarwal et al., 2014; Tsai et al., 2018). As the use of the Internet is growing with the daily activities of users, data are growing daily with increasing data availability in a variety of features. However, these multidimensional features make design and analysis more complicated with intricate performance. This kind of complex data requires big data analytic which facilitates the performance of data analysis to reduce cost and facilitate instant decisions.

Consequently, Big Data technologies get more attention from both academia and industry as data are growing in volume. This helps people to understand how it is important to improve the efficiency of data analytic software in order to improve data processing capability (H. Zhang et al., 2018). Although Hadoop became one of the Big Data computational tools designed for clusters computing in the past two decade to run on commodity hardware to process huge data in a distributed ways across different machines(Aggarwal et al., 2014; Choi et al., 2015; Asuncion & Newman, 2007) with MapReduce implementation framework(Qiu et al., 2018;Gantz et al., 2007) for large scale distributed data processing(Wu & Gokhale, 2013), it is not efficient for data analytic applications, especially for those require interactive and iterative computational tasks(Herodotou et al., 2011; Kambatla et al., 2009) due to its implementation read data from a disk that hinder its' capability.

Spark is recently designed as in-memory computing (Li et al., 2017; Ewart et al., 2015) to overcame the limitation of Hadoop by avoiding low access of disks (Shi et al., 2015; Jiang et al., 2014; Adinew et al., 2019). However, sometimes disk access is also required for reading the RDDs, if they are not fully cached (K. Zhang et al., 2017; Garg & Janakiram, 2018). Moreover, Spark has the ability to persist recomputed data in memory (Baig et al., 2018; Zhou et al., 2018) which eliminates a significant amount of I/O that happens in the case of reading disk (Xu et al., 2016; Adinew et al., 2020). Although Spark overcame the limitation of Hadoop (K. Wang & Khan, 2015), still it has a performance bottleneck, which has been trying to address by different researcher. However, still very challenging to get the best performance (Rahman et al., 2018; Koliopoulos et al., 2016) so that it requires to find a way to optimize its performance based on the optimization framework defined in R. Zhang et al. (2015). Accordingly, we focus on a combination of Scheduler and Shuffle manager with data serialization and data caching mechanism to improve performance of Spark application.

Consequently, from this research work, we identified the following lists as contributions of our work.

- Scheduling and Shuffling with data serialization and data caching option was proposed to improve performance of Spark application.
- The execution performance of Spark applications in Scheduling and Shuffle Manager with data serialization in different intermediate data caching options on different Algorithms was investigated.
- The relationship between Scheduler and Shuffler Manager with data serialization, and data caching options were identified.
- It was confirmed that FIFO Scheduler and Sort Shuffle Manager in the OffHeap data caching option slightly shows high performance than others combinations regardless of data serialization.
- It was confirmed that Tungsten-Sort Shuffler with FIFO Scheduler shows good performance regardless of data serialization in serialized data caching options in all algorithms and in all datasets.
- It was confirmed that Memory Only Ser data caching option shows good performance than Memory And Disk Ser data caching option, almost in all combinations

- Overall, it was confirmed that the performance of Scheduler and Shuffle Manager varies and depends on data serialization and data caching options and algorithms.

The rest of this research is organized in a section based. Where Section 2 describes the proposed model. Section 3 describes research Methodology. Section 4 describes Experimental process. Section 5 illustrate Experimental Results. Section 6 discusses Experimental result. Section 7 mentions related work of the research. Finally, Section 8 Summarizes the conclusion of the research.

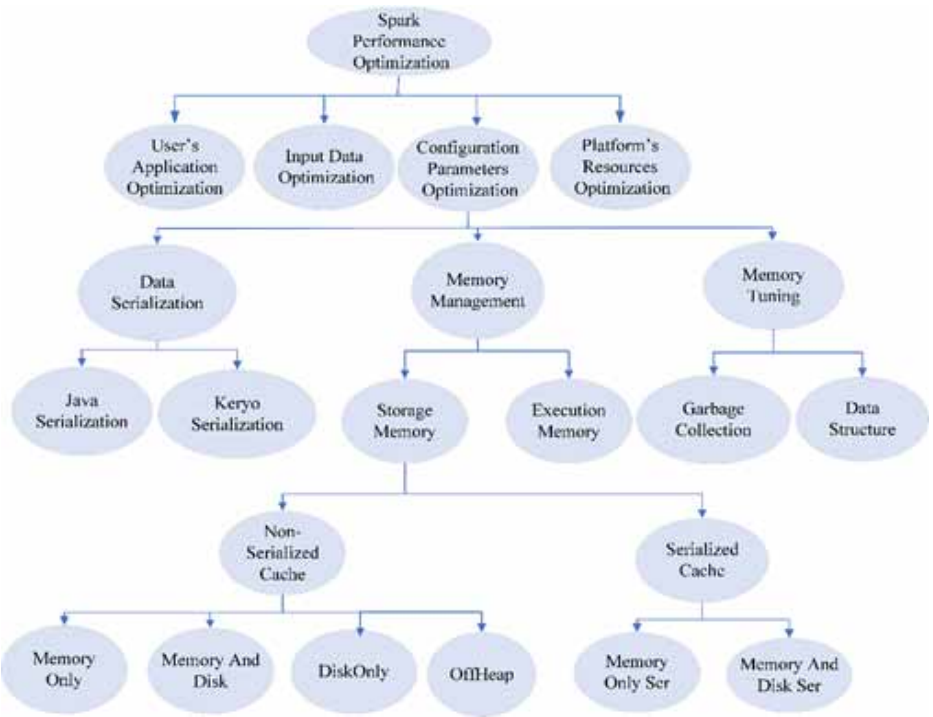
2. PROPOSED MODEL

This research focuses on investigating the execution performance of Spark applications in using scheduling and shuffle Manager with data serialization in different intermediate data caching options. The experiment focuses on Spark configuration parameters that were defined as one of performance optimization parameters, which were defined as

$$SparkPerf = Func(A, D, R, C),$$

Where SparkPerf denotes the performance of Spark platform, A denotes the user's application, D denotes the input data, R denotes the platform's resources, and C denotes the Spark platform parameters configuration, Func denotes a performance function of Spark performance in using A, D, R, and C as parameters (G. Wang et al., 2017). However, this experiment particularly focuses on Configuration parameter c which may require the selection of certain parameters with careful selection due to improper selection may lead to significantly degrade its performance (Du et al., 2019)[8, 23]. The performance optimization category used for this experiment is illustrated in Figure 1.

Figure 1. Spark Performance Optimization Categories



2.1. Scheduling

As Spark application is submitted to cluster computing, submitted application will get its' independent executor to process the tasks. The scheduler decomposes a job into different stages according to RDD dependencies and submits the stages one by one to the executors. It uses a FIFO scheduler as a default scheduler when applications are computed in the standalone cluster. However, it can be changed at any time when an application is running (Zhou et al., 2018).

2.2. Data Shuffling

Spark keeps data distributed across cluster nodes while data being partitioned. So that, during the execution of an application, each data partition is fed into corresponding Worker nodes. However, during data processing, input data (RDDs) are shuffled across machines as a job progresses (Choi et al., 2015). Therefore, data shuffling is one of the very important parameter configurations, which guarantee to shuffle data persisted in the storage to provide better reliability after successful completion of execution task.

2.3. Data Serialization

Spark can be configured to use either the default Java serialization or the Kryo serialization libraries (H. Zhang et al., 2018; Koliopoulos et al., 2016). According to a study2020(Tsai et al., 2018, Zhou et al., 2018) Kryo offers high-performance improvements over Java serialization. However, its effect is not assessed in such a multi-layer experimental structure with a combination of other parameters.

2.4. RDD Caching Mechanism

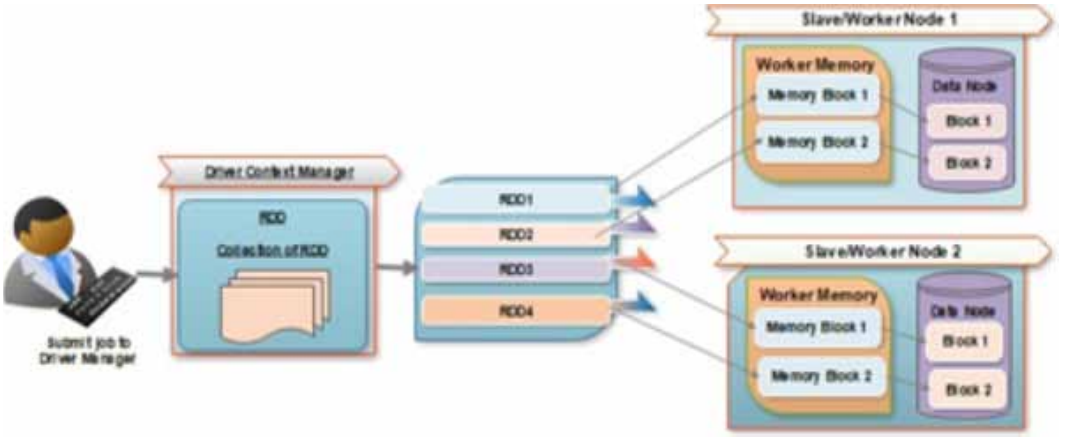
Data Caching is the way to keep data in memory to create a mechanism to bring the best performance and speed up data retrievals processing mechanism. Data Caching is simplified using localizing RDDs to make every node maintain their own cache without considering the problems of coherency. Moreover, Spark adds its caching semantics with persist (). Although the default way of the data caching option is to store Java objects directly into memory, users can also by themselves explicitly can cache an RDD according to their requirement in different data caching options. Accordingly, RDDs are stored for while in JVM heap memory bypassing Storage Level object with MEMORY ONLY data caching option. However, persisting RDDs in memory by itself causes significant overhead in GC time. To alleviate this problem, Spark offers various data caching options and enables a user to implement custom caching strategies. However, this practice demands expert knowledge of the underlying platform and extensive experimental evaluation of the different options with other different parameters such as Data serialization, Scheduler, and Shuffler as a combination to simplifies task for users while selection.

3. METHODOLOGY

This experiment follows the Spark standalone cluster computing model by selecting as best deployment model, approaches, and methods as shown in architectural defined in Figure 2, which emphasis an RDD computation approach. Moreover, a cluster computing component and their interaction were also identiðed as the Driver, the Master, the Cluster Manager, and the Executor(s), which run different Worker. Application anatomy and workloads were also identified, developed, and used for evaluating Spark performance under different combinations of these parameters that were listed in table 2.

For data submission, submit command line is used that allows submitting application and configuration to Spark. Architectural Model is designed to show how different RDD on different workers is processed as shown in Figure 2.

Figure 2. Show RDD submission to workers with block allocation



4. EXPERIMENTS

4.1. Environment Variable

Environment variable that is used to support our experimental activities are categorized and listed as Hardware and Software environment variables. The hardware that is used for our experiment is a standalone laptop where the details specifications are listed and described in Table 1 together with details of Software used for experiments.

Table 1. Hardware and software configuration environments

Hardware	Software
Memory:4GB with additional 2GB GPU	Window 10 OS with 64-bits
Hard Disk:750GB	Spark 2.4.4 with Hadoop 2.7
Intel(R) Core(TM)i5-5200U CPU@2.20GHz	Scala 2.11.12 JDK 11.0.1 Python3.7.1

4.2. Configuration Variables Parameters

For this experiment, six different configuration parameters were used. Their details are listed and described in Table 2 with their default and new value.

4.3. Parameter Selection

Spark has almost more than 180 parameters that can be easily adjusted (configured) by users according to their own application requirements to increase its performance. These large parameters option creates very complex interactions between parameters in their combination which required appropriate ways of parameter tuning. If it is carefully tuned, such a large space in parameter configuration enables a lot of opportunities for performance improvement. Accordingly, six main parameters were selected, which independently has a high impact on Spark performance but not checked yet in their combination in any of research. The Choice of optimal Shuffling and Scheduling with data serialization on different storage levels requires in-depth systems-level knowledge which was not assessed yet through experimentation. The reasons for selecting these parameters are individually they have a

Table 2. Parameters configuration used for experiment with default and new value

Categories	Parameters	Description	Default Value	New Value
Shuffle Related Parameter	Spark.shuffle. manager	Manage shuffle metadata on the driver(Shuffle Handle) and running tasks on executors to access the metadata	Sort	Sort and Tungster-sort
	spark.shuffle.service. enabled	Enables the external shuffle service	false	True
Scheduling mode	spark.scheduler.mode	Scheduling mode submitted to SparkContext	FIFO	FAIR
Data Serialization	spark.serializer	It is used for caching serialized objects	Java Serializer	Java Serializer Kryo Serializer
Data Storage	Storage Level	All cached RDD can be stored in different data caching levels in deserialized format	None	MEMORY ONLY MEMORY AND DISK,DISK ONLY, OFFHEAP
		All cached RDD can be stored in different data caching levels in serialized format	None	MEMORY ONLY SER MEMORY AND DISK SER

highly significant impact on Spark application performance, but their combined performance is not checked yet in any of the researchers.

4.4. Model Selection

Although several common models are available under machine learning for data analytic techniques which are used to measure the performance of a different application.We select some Spark benchmarks for this experiment such as WordCount, Sort, and PageRank for applying different parameter tuning in order to investigate the effect of Scheduling and Shuffling in different data serialization and in different intimidate data caching options. We selected those benchmarks because having a join operation appears frequently in such applications which creates a lot of shuffling. They are also the real representation of Spark application, which are used alone as well as by integrating with different applications which are implemented in a wide range of applications. They are implemented in Spark Scala code.

4.5. Programming Language

Although Spark supports several common programming languages to write a code such as Scala, Java, Python, etc, our BenchMark is implemented in Scala using IntelliJ ideal as environment IDE by integrating with SBT. Python was used to draw the bar chart and others software were used as experimental support.

4.6. Experimental Steps

In order to investigate their performance improvement. We first install and configure Spark according to our deployment model requirements for huge data analytics. IntelliJ idea was also install by integrating with Scala and SBT to write different Spark application that helps to do this experiment. Then we create a Spark cluster that holds one master that launches the driver program and two workers that launch different Executors as JAVA processes. After forming a cluster, applications were submitted to the Spark cluster with different parameters configuration values with dataset

from the command line using submit command. Then, the driver program was launched to run in JVM by creating a SparkContext object. Then, Master assigns tasks with resources to workers in the cluster for computation. The submission mode is using cluster deploy mode with different parameters setting. The submitted applications were decomposed into tasks and wait in queue for execution as the JAVA process is launched. Executors were execute tasks assigned by Worker according to resources allocated to them. In the end, the results were replayed to the Master by driver program by collecting and combining all results from each Worker. Finally, execution time was collected as one of the performance evaluation mechanisms for Spark application in Spark optimization which were directly accessed from its web as the performance of the applications at the end of each computation. Then, experimental results were visualized in different forms such as figures and tables, which show performance comparison between the default value and new value of those configuration parameters.

Sample commands that were submitted with different parameters configuration as job are shown in below, which is taken from PageRank algorithm sample code.

```
C: Users mesay IdeaProjects untitled6>spark- submit --master
spark://113.54.216.149:7077
--deploy-mode cluster --conf "spark.rpc.askTimeout=10000s"
--conf"spark.network.timeout=80000s" -- conf"spark.shuffle.
service.enabled =True"
--conf "spark.shuffle.manager=tungsten- sort" --conf "spark.
storage.level= MEMORY ONLY" --class Spark-PageRank PageRank.jar
file:D: Messay 360 PageRank web.txt spark://113.54.216.149:7077 2
```

Each application run under one driver and multiple executors to allow parallel process that run under different executors. Accordingly, tasks were submitted three times for computing average of its computational time for every parameter's configuration. So that experiments were done repeatedly to make sure that the experiments are well evaluated.The experiment was done in two-phase where the first phase focuses on non-serialized data caching options with other parameters. Whereas the second phase focuses on Serialized data caching options.

4.7. Workloads and Datasets

For this experiment, a number of different dataset samples were used. Datasets that were used in phase one was accessed from different data sources such as the Stanford dataset [3] and the UCI of Machine Learning Repository. Where some of the datasets that were used in phase two were constructed manually from the dataset used in phase one which was accessed from the above data

Table 3. Dataset used in phase one

Applications	Input size
PageRank	31.3MB,71.8MB
TeraSort	11KB,22KB,43KB
WordCount	2MB,4MB,16MB

Table 4. Dataset used in phase two

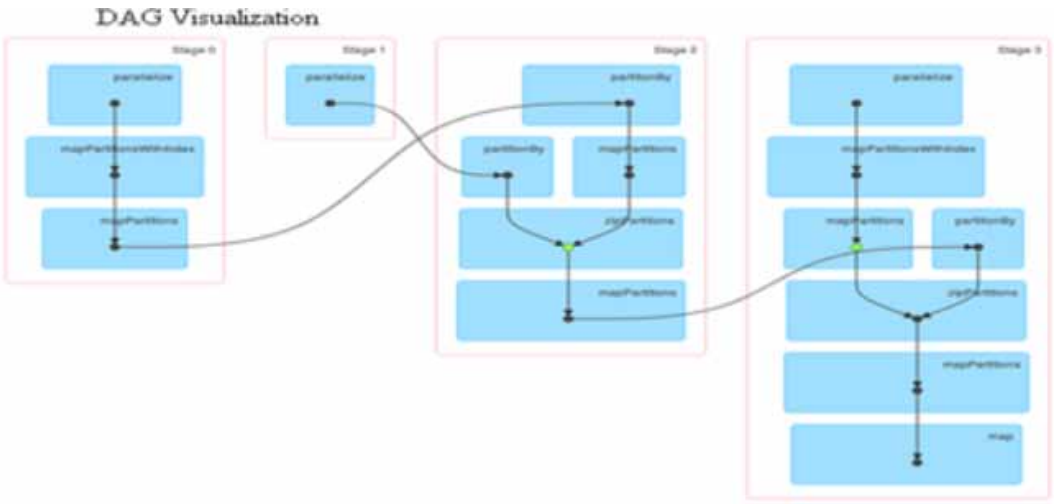
Applications	Input size
PageRank	32MB, 72MB, 500MB, 750MB, 1GB
TeraSort	11KB, 22KB, 43KB, 252KB, 531MB, 735MB
WordCount	2MB, 8MB, 16MB, 1GB, 2GB, 3GB

sources by customizing it to increase the data size. A WordCount, TeraSeort, and PageRank algorithms were selected and used to evaluate Spark performance under different parameters value and different dataset sizes. The algorithms and datasets that were used for the experiment one are listed in Table 3. The datasets that were constructed and used for experiment phase two are listed in table 4.

5. EXPERIMENTAL RESULT

The experiments were done alternatively by combining different Scheduler and Shuffle managers along with data serialization in different algorithms in different data caching options in two-phase.

Figure 3. Job Graph (DAG) in cluster computing from PageRank algorithms



The experimental activities pass through the different stages during data computation. Depending on computational activities, each stage has its own operations in each computational task like map, join, groupBykey, reduceByKey and so on which include different scheduling and shuffling operations. These data computation steps are called RDD transformation and action which is visualized as RDD DAG or job graph. It shows how jobs are executed. A job graph for PageRank was illustrated in Figure 4 which include different details of transformations and actions under the different stage of its task.

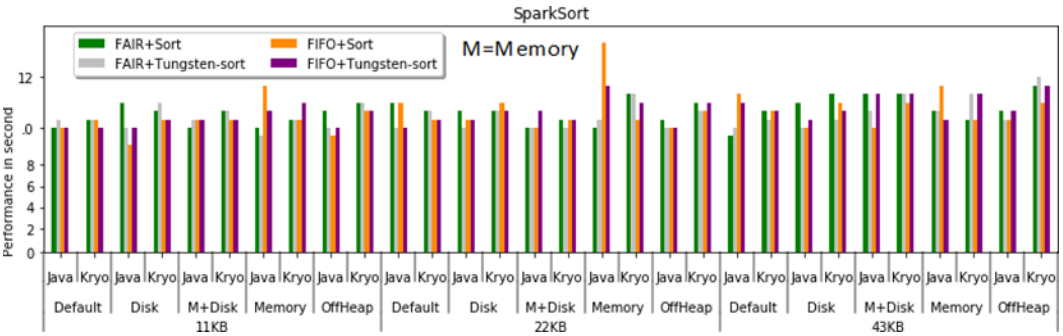
5.1. Experimental Result Visualization

This section describes performance comparison between different applications and datasets for non-serialized data caching options with other Spark parameters. The result is visualized with different Schedulers with different Shuffler in different data serialization in different dataset sizes in different machine learning applications as is shown in Figures 4 to 6.

Performance comparison between Memory Only Ser and Memory and Disk Ser storage level with the combination of other parameters configuration are shown in Figures 7 to 9.

Figure 4: Illustrate Performance Achieved in Scheduling and Shuffling with Data Serialization in Different

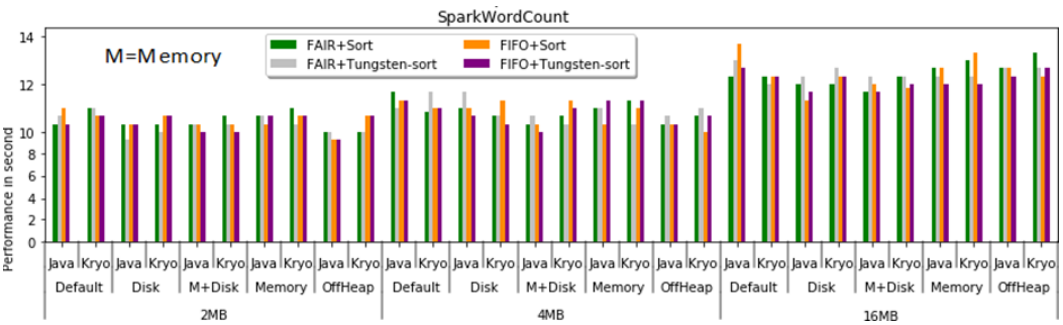
Figure 4. Combination of other parameters



Storage Level in Spark Sort Algorithm

As shown in Figure 4, the FIFO scheduler with Sort shuffler together with Java serialization on OffHeap data caching show best performance than other different combinations in Spark Sort Algorithm. Whereas Memory and Disk data caching option also show good performance by following next to OffHeap data caching option.

Figure 5. Illustrate Performance Achieved in Scheduling and Shuffling with Data Serialization in Different Storage Level in Spark Word Count Algorithm



As shown in Figure 5, the FIFO scheduler and Sort shuffle manager with java serialization on Offheap show good performance than other combinations in Spark Word Count Algorithm. Whereas the OffHeap data caching option also shows good performance next to the Disk Only data caching option regardless of data serialization and data size.

As shown in Figure 7, FIFO Scheduler and Tungsten- Sort Shuffler have higher performance improvement on Memory Only Ser data caching option than Memory And Disk Ser data caching option regardless of data serialization on Spark Sort Algorithm in all datasets. Whereas specifically in Memory And Disk Ser data caching option still FIFO Scheduler Tungsten-Sort Shuffler with Java data serialization shows higher performance improvement in all datasets.

As shown in Figure 8, again FIFO Scheduler and Tungsten-Sort Shuffler have higherperformance improvement on Memory Only Ser data caching option than Memory And Disk Ser data caching option regardless of data serialization on Spark Word Algorithm in all datasets.

Figure 6. Illustrate performance achieved in scheduling and shuffling with data serialization in different storage level in spark page rank algorithm

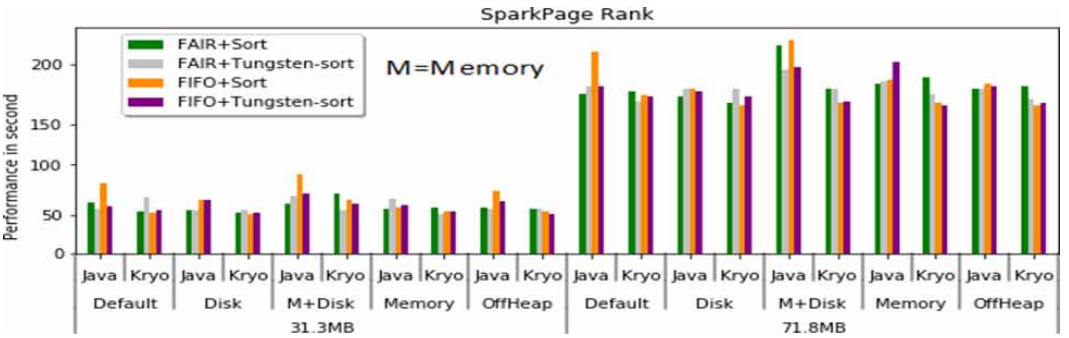


Figure 7. Illustrate Performance Evaluation between Memory Only Ser and Memory And Disk Ser in Scheduling and Shuffling in Different Data Serialization in Spark Sort Algorithm

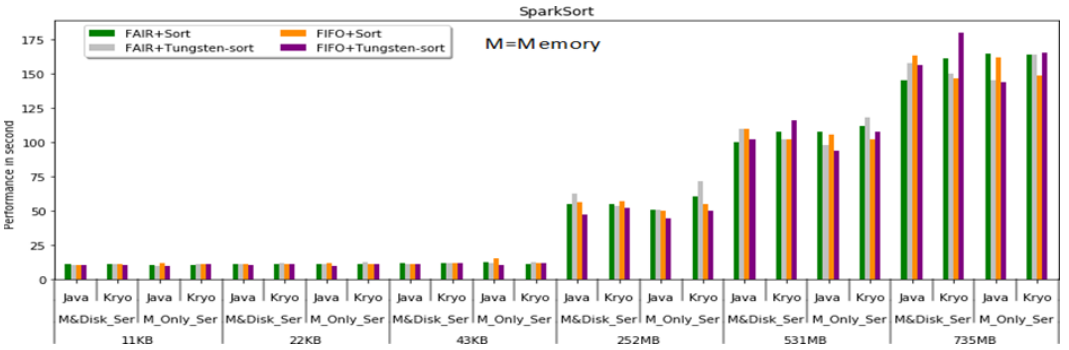
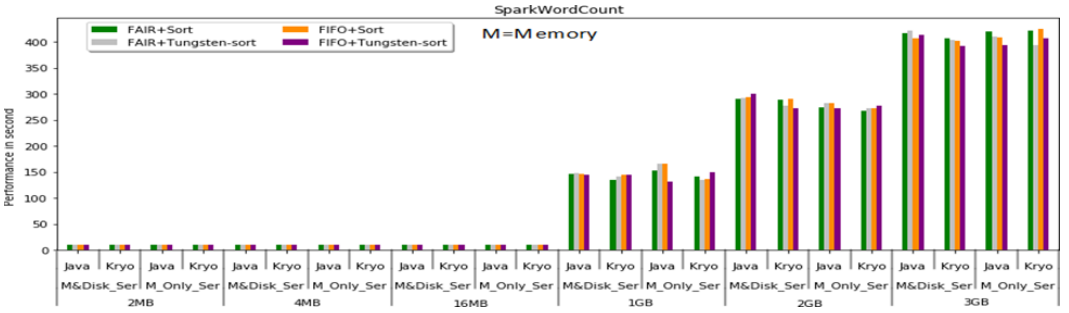


Figure 8. Illustrate Performance Evaluation between Memory Only Ser and Memory And Disk Ser in Scheduling and Shuffling in Different Data Serialization in Spark Word Count Algorithm



As shown in Figure 9, again FIFO Scheduler and Tungsten-Sort Shuffler show higher performance improvement on Memory Only Ser data caching option than Memory And Disk Ser data caching option regardless of data serialization on Spark PageRank Algorithm in all datasets.

Figure 9. Illustrate performance evaluation between memory only ser and memory and disk ser in scheduling and shuffling in different data serialization in spark page rank algorithm

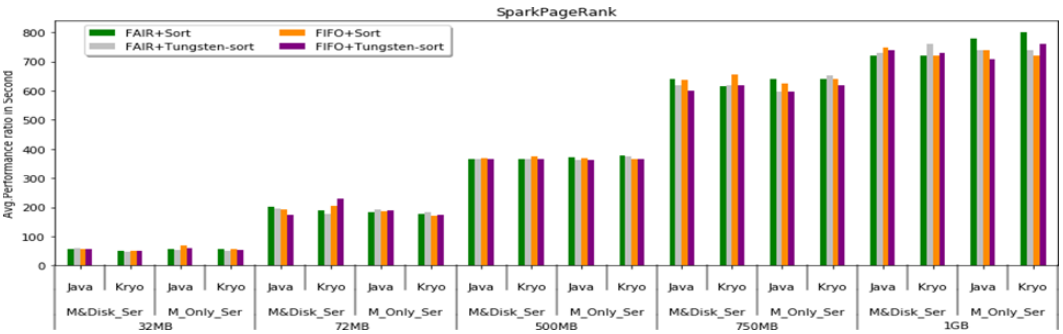


Table 5. Performance improvements result for non-serialized data caching options

Application	Storage Level	Java				Kryo			
Sort	Scheduler + Shuffler	FF + Sort	FF + T-Sort	FR + Sort	FR + T-Sort	FF + Sort	FF + T-Sort	FR + Sort	FR + T-Sort
	Memory Only	-5.38	0	-1.1	1.07	-7.53	-1.06	-0.54	17.28
	Memory&Disk	-4.31	-2.18	-2.2	-1.07	-3.23	-2.11	-0.22	19.1
	Disk Only	1.08	-6.53	1.1	-3.2	-2.16	-3.16	-0.22	18.19
	Heap	1.08	-3.27	0	-3.2	-7.53	-6.32	-0.75	1.82
WordCount	Memory Only	0.98	0.01	1.93	-2.95	0	-3.42	4.81	2
	Memory&Disk	7.77	5.83	3.85	1.48	2.95	1.47	4.81	2
	Disk Only	3.48	2.72	2.69	-0.99	1.57	3.32	3.65	2.01
	Heap	5.83	3.89	3.85	2.95	0	0.49	2.89	-1
PageRank	Memory Only	15.2	-8.79	-1.28	-5.07	2.8	4.1	-6.36	3.14
	Memory&Disk	-5.81	-11.3	-16.67	-10.13	-1.62	-0.59	-7.52	0.15
	Disk Only	15.67	-0.7	3.39	1.27	4.42	1.03	4.77	0.15
	Heap	11.84	-1.82	0	0.99	3.68	4.1	-2.46	3.28

Table 5 and 6 show details of performance improvement result for Non-Serialized Data Caching Options and Serialized Data Caching Options respectively. Every computation was done by using default configuration result as base result. So that the performance improvement was calculated as the difference between new configuration result and default value result. Where default value and new configuration of these parameters are described in Table 2.

Table 6. Performance improvement result for serialized data caching options

Data Caching Options	Data Serialization	Scheduler + Shuffler	Sort			WordCount			PageRank		
			D-1	D-2	D-3	D-1	D-2	D-3	D-1	D-2	D-3
Memory Only SER	Java	FF+T-Sort	11.92	11.33	11.14	20.5	3.55	3.44	1.64	4.49	4.06
		FR+Sort	-1.32	-1.89	-2.07	8.44	2.84	-2.95	-1.09	-2.57	-5.41
		FR+T-Sort	-0.66	7.55	10.11	0	0	-0.5	1.64	4.17	0
	Kryo	FF+Sort	-9.94	3.78	8.05	18.1	3.55	-3.93	1.09	-2.57	2.71
		FF+T-Sort	0	-1.89	-2.48	9.64	1.42	0.5	1.09	0.65	-2.71
		FR+Sort	-19.87	-5.67	-1.24	14.5	4.97	-3.44	-2.72	-2.57	-8.11
		FR+T-Sort	-43.03	-11.33	-1.45	19.3	3.55	3.44	-2.18	-4.81	0
Memory and Disk SER	Java	FF+T-Sort	15.89	7.28	4.29	1.37	-2.05	-1.98	1.09	5.96	1.34
		FR+Sort	3.53	9.1	10.11	0	1.37	-2.47	0.55	-0.32	4
		FR+T-Sort	-10.01	0	2.48	-1.37	0.69	-3.95	1.09	2.83	2.67
	Kryo	FF+Sort	-1.19	7.28	9.28	1.37	1.37	0.99	-1.64	-2.83	4
		FF+T-Sort	7.65	-5.46	-11.14	1.37	7.49	3.45	1.09	2.83	2.67
		FR+Sort	2.95	1.82	0.41	8.22	2.05	0	1.09	3.45	4
		FR+T-Sort	5.3	7.28	7.01	2.74	5.45	0.5	1.09	3.14	-1.34

6. DISCUSSION

Experiments were done in two phases on three representatives of a Spark application such as WorkCount, TeraSort, and PageRank. They were used as workloads to evaluate performance achieved in changing the value of the above six experimental parameters configuration. The experimental results are shown in Figure 4 to 9 which describe, which Scheduler and Shuffle Manager in which data serialization such as Java and Kryo serialization with which intermediate data caching options such as Memory Only, Memory And Disk, Disk Only, OffHeap, Memory Only Ser, and Memory And Disk Ser in Spark data computing show high-performance improvement, that can seen in two categories.

The results in experimental phase one were depicted and shown in graphs as they are illustrated in Figures 4 to 6. The experimental results observed in different algorithms are summarized as follows.

1. In Spark Sort Algorithm, FIFO scheduler with Sort shuffler together with Java serialization in OffHeap show the best performance than others different combination, whereas Disk Only data caching option also show good performance than other different combinations follow OffHeap data caching option.
2. In Spark Word Count Algorithm, FIFO scheduler and Tungsten-sort shuffle manager with java serialization in Memory and Disk Only show good performance than others combination, whereas OffHeap also shows good performance follow Memory and Disk data caching option.
3. In Spark Page Rank Algorithm, FIFO scheduler and Sort shuffle manager with Kryo serialization in OffHeap show good performance than other different combination, whereas Disk Only data caching option also show good performance than others different combinations follow OffHeap data caching option.

6.1. Performance From Scheduler and Shuffler Perspective

```
If(FIFO Scheduler and Sort Shuffle Manager) Performance is good
with Kryo Serialization
in Offheap Data Caching.
ElseIf(FIFO Scheduler and Tungest Shuffle Manager)
Performance is good with Kryo Serialization in Offheap Data
caching.
ElseIf(FAIR Scheduler and Sort Shuffle Manager)
Performance is good with Java Serialization in Default Data
caching almost in all
algorithms.
ElseIf(FAIR Scheduler and Tungest Shuffle Manager) Performance is
good with Java
Serialization regardless of the Data caching mechanism.
```

In general, the result shows FIFO Scheduler and Sort Shuffle Manager with kryo Serialization in Offheap data caching relatively has good performance.

Overall, as we observed from the above three experimental results which were done in phase one, performance in Scheduling and Shuffling varies and depends on data serialization and data caching options as well as it also varies from algorithm to algorithm.

In general, although the result varies across algorithms, FIFO Scheduler and Sort Shuffle Manager in OffHeap data caching option slightly show high performance than others combinations regardless of data serialization in most of the algorithms.

The results from experimental phase two are shown in Figures 7 to 9, which compare performance between Memory Only Ser or Memory And Disk Ser in different Scheduler and Shuffler with different dataset sizes. Accordingly, the result of performance comparison between Memory Only Ser and Memory And Disk Ser with different data serialization in different Scheduler and Shuffler, Memory Only Ser with FIFO Scheduler &TungstenSort Shuffler shows overall high performance regardless of data serialization in all three workloads and in all datasets. When we compare performance between Java data serialization and Kryo serialization with the above combinations of parameters configuration, Java data serialization with FIFO Scheduler and TungstenSort Shuffler shows overall high performance regardless of data caching options in all three workloads and in all datasets. As we also compared performance between Scheduler and Shuffler, FIFO Scheduler and TungstenSort Shuffler show high performance improvement regardless of data serialization and data caching options in all workloads and datasets. However, in Memory And Disk Ser data caching option, Kryo show overall good performance improvement in all Scheduler and Shuffler in all workloads and in all dataset next to Java data serialization with FIFO Scheduler. TungstenSort Shuffler in both Memory Only Ser and Memory And Disk Ser data caching options show good performance. In general, TungstenSort Shuffler shows best performance improvement in all combination, FIFO Scheduler with TungstenSort Shuffler also show best performance improvement. From a data serialization perspective, Java data serialization is slightly high performance than kryo in all data caching options, in all workloads, and all datasets. Memory Only Ser data caching option show good performance almost in all combinations than Memory And Disk Ser data caching options.

7. RELATED WORK

Although many researchers were tried to analyze the parameters configuration of Spark platforms in order to improve Spark performance by modifying the default configuration in a way to increase the performance of Spark applications, they were focused on certain common individual parameters. Consequently, authors of (K. Zhang et al., 2017) investigate the influence of different intermediate data caching options on execution performance of Spark applications, by a comparison of disk-based

caching and other data caching options. The experiment also extends to compared serialization of RDD with an encoding of DataFrame in intermediate data caching The authors (Xu et al., 2016) examine how to improve overall memory utilization and also work in the way to minimize performance degradation by designing MEMTUNE to improve the performance of Spark. It is designed to monitors memory consumption in garbage collection and memory paging frequency, during tasks computation using a statistical data collection mechanism. It also uses this information to dynamically adjust data cache space which is implemented under Spark options such as MEMORY ONLY and MEMORY AND DISK. MEMTUNE will use APIs to manage the RDD cache automatically. The authors of(Koliopoulos et al., 2016) propose an automatic data caching selection mechanism which helps to explores the memory impact of different workloads using different storage levels. It is a framework designed for in-memory distributed systems that automatically selected the optimal storage level based on data size and cluster characteristics. In authors (Tsai et al., 2018) identify the existence of more effective memory resource requirements for computing tasks on large data analytic applications. An author proposed a prediction model which forecasts an application memory requirement based on its service level agreement which is implemented in machine learning algorithm in effective ways.. In authors (Zhou et al., 2018), it was proposed an I/O- aware data analytical model to reason out the performance improvement of Spark applications. It also focus on the effective utilization of I/O bandwidth together with different data access sizes and different numbers of CPU cores. In authors (Baig et al., 2018) they evaluated the performance of the Spark platform using different algorithms on NUMA hardware. They performed their experiments on a single machine that hold and manage the number of executors in each worker, number of cores, and memory allocated to each worker together with other different parameters. In authors (Rahman et al., 2018), it was proposed and developed an effective, self-tuning approach, namely SMBSP, based on Artificial Neural Network (ANN) to avoid the drawbacks of manual parameters tuning. The experiment was done using five parameters and 5 different dataset sizes. In reference (Wu & Gokhale, 2013), the authors designed Profiling and Performance Analysis based System (PPABS) framework in the way to automate the tuning strategies of the Hadoop application based on its configuration requirement. In authors (R. Zhang et al., 2015), it was proposed an engine that will recommend configurations for those newly submitted data analytic jobs. The aim of designing an engine is to alleviate the issue shown in a modified k nearest neighbor algorithm, which finds desirable configurations from similar past jobs that have performed well. In authors (K. Wang & Khan, 2015), the authors were tried to model simulation-driven prediction that can predict job performance in high accuracy prediction for Spark platform. The model is designed in a too simple way to predict memory usage and to predict the overall execution time of Spark applications in default parameters as well as in changing the default value of different parameters configuration.

Although several research efforts were carried to investigate and mitigate the impact of different RDD cache options, they did not investigate the relationship of Scheduling and Shuffling with different RDD caching options with different data serialization mechanism. The most related works to our work are the authors(K. Zhang et al., 2017) and authors(Xu et al., 2016). In the former, the authors focus on the influence of different intermediate data caching options by a comparison of one storage caching option with other caching options without taking into consideration others parameters such as scheduler and Shuffler as well as data serialization. In the latter, the authors examine the way to improve overall memory utilization and reduce performance degrading mechanism which follows GC overhead improvement approach.In both works, it does not investigate the relationship of data serialization in different cache storage options.

In general, some of the related work followed a normally GC overhead approach in measuring the performance improvement and none of them did focus to investigate performance impact of scheduling and shuffling with data serialization and data caching options.

8. CONCLUSION

As data grows in different dimensions it requires appropriate data analytic. Among several big data analytic tools, Apache Spark is the most dominant that requires further optimization in the parameter's configuration domain. Accordingly, we proposed and configure different scheduling and shuffling with data serialization on different intimidate data caching options to improve performance of Spark application. Experiments are done in two phases as experiments one and two on three representative Spark applications to evaluate performance achieved in changing the default values. In experimental one, although the performance of Scheduler and Shuffle Manager varies and depends on data serialization, data caching options, and algorithms, it was observed FIFO Scheduler and Sort Shuffle manager in OffHeap data caching options slightly show the best performance than others different combinations in almost all of algorithms. In experimental two, Tungsten-Sort Shuffler with other combinations and performance in all combinations especially with FIFO Scheduler it shows good performance. In data caching options, the Memory Only Ser data caching option shows the best performance than Memory And Disk Ser data caching option almost in all combinations. In a conclusion, even though performance varies and depends on data serialization and data caching options and algorithms, 2.45% and 8.01% performance improvement are achieved in experiments one and two, respectively.

REFERENCES

Adinew, D. M., Shijie, Z., & Liao, Y. (2019). Spark Performance Optimization Analysis in Memory Tuning on GC Overhead for Big Data Analytics. *ACM International Conference Proceeding Series*, 75–78. doi:10.1145/3375998.3376039

Adinew, D. M., Shijie, Z., & Liao, Y. (2020). Spark performance optimization analysis in memory management with deploy mode in standalone cluster computing. *Proceedings - International Conference on Data Engineering*, 2049–2053. doi:10.1109/ICDE48307.2020.00242

Aggarwal, C., Subbian, K., Butler, K., Stephens, M., Stephens, M., Chakrabarti, D., Kumar, R., Tomkins, A., Clauset, A., Moore, C., Newman, M. E. J., Csardi, G., Nepusz, T., Decelle, A., Krzakala, F., Moore, C., Zdeborov, L., Eisinga, R., Te Grotenhuis, M., … Cov, E. R. (2014). SNAP Datasets: Stanford Large Network Dataset Collection. *Physical Review Letters, Complex Sy*, (1).

Asuncion, A., & Newman, D. J. (2007). *UCI Machine Learning Repository: Data Sets*. University of California Irvine School of Information.

Baig, S. U. R., Amaral, M., Polo, J., & Carrera, D. (2018). Performance characterization of spark workloads on shared NUMA Systems. *Proceedings - IEEE 4th International Conference on Big Data Computing Service and Applications, BigDataService 2018*, 41–48. doi:10.1109/BigDataService.2018.00015

Choi, I. S., Yang, W., & Kee, Y. S. (2015). Early experience with optimizing I/O performance using high-performance SSDs for in-memory cluster computing. *Proceedings - 2015 IEEE International Conference on Big Data*, 1073–1083. doi:10.1109/BigData.2015.7363861

Du, H., Han, P., Chen, W., Wang, Y., & Zhang, C. (2019). Otterman: A Novel Approach of Spark Auto-tuning by a Hybrid Strategy. *2018 5th International Conference on Systems and Informatics, ICSAI 2018, Icsai*, 478–483. doi:10.1109/ICSAI.2018.8599304

Ewart, T., Yates, S., Cremonesi, F., Kumbhar, P., Schürmann, F., & Delalondre, F. (2015). Performance evaluation of the IBM POWER8 architecture to support computational neuroscientific application using morphologically detailed neurons. *Proceedings of the 6th International Workshop in Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, PMBS 2015 - Held as Part of ACM/IEEE Supercomputing 2015, SC 2015*. doi:10.1145/2832087.2832088

Gantz, B. J., Reinsel, D., & Shadows, B. D. (2007). Big data, bigger digital shadow s, and biggest grow th in the far east executive summary: A universe of opportunities and challenges. *IDC*, 1–16.

Garg, N., & Janakiram, D. (2018). Sparker: Optimizing spark for heterogeneous clusters. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, 1–8. doi:10.1109/CloudCom2018.2018.00017

Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., & Babu, S. (2011). Starfish: A self-tuning system for big data analytics. *CIDR 2011 - 5th Biennial Conference on Innovative Data Systems Research, Conference Proceedings*.

Jiang, T., Zhang, Q., Hou, R., Chai, L., McKee, S. A., Jia, Z., & Sun, N. (2014). Understanding the behavior of in-memory computing workloads. *IISWC 2014 - IEEE International Symposium on Workload Characterization*, 22–30. doi:10.1109/IISWC.2014.6983036

Kambatla, K., Pathak, A., & Pucha, H. (2009). Towards optimizing hadoop provisioning in the cloud. *Workshop on Hot Topics in Cloud Computing, HotCloud 2009*.

Koliopoulos, A. K., Yiapanis, P., Tekiner, F., Nenadic, G., & Keane, J. (2016). Towards automatic memory tuning for in-memory big data analytics in clusters. *Proceedings - 2016 IEEE International Congress on Big Data, BigData Congress 2016*, 353–356. doi:10.1109/BigDataCongress.2016.56

Li, C. S., Franke, H., Parris, C., Abali, B., Kesavan, M., & Chang, V. (2017). Composable architecture for rack scale big data computing. *Future Generation Computer Systems*, *67*, 180–193. Advance online publication. doi:10.1016/j.future.2016.07.014

Qiu, D., Zhou, W., & Liu, J. (2018). Performance evaluation and optimization of join operation in spark for big data processing. *2017 3rd IEEE International Conference on Computer and Communications, ICCC 2017*, 2295–2298. doi:10.1109/CompComm.2017.8322944

Rahman, M. A., Hossen, J., & Venkataseshaiah, C. (2018). SMBSP: A Self-Tuning Approach using Machine Learning to Improve Performance of Spark in Big Data Processing. *Proceedings of the 2018 7th International Conference on Computer and Communication Engineering, ICCCE 2018*, 274–279. doi:10.1109/ICCCE.2018.8539328

Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., & Özcan, F. (2015). Clash of the titans: Mapreduce vs. spark for large scale data analytics. In *Proceedings of the VLDB Endowment* (Vol. 8, Issue 13). doi:10.14778/2831360.2831365

Tsai, L., Franke, H., Li, C. S., & Liao, W. (2018). Learning-Based Memory Allocation Optimization for Delay-Sensitive Big Data Processing. *IEEE Transactions on Parallel and Distributed Systems*, *29*(6), 1332–1341. doi:10.1109/TPDS.2018.2800011

Wang, G., Xu, J., & He, B. (2017). A Novel Method for Tuning Configuration Parameters of Spark Based on Machine Learning. *Proceedings - 18th IEEE International Conference on High Performance Computing and Communications, 14th IEEE International Conference on Smart City and 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016*, 586–593. doi:10.1109/HPCC-SmartCity-DSS.2016.0088

Wang, K., & Khan, M. M. H. (2015). Performance prediction for apache spark platform. *Proceedings - 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security and 2015 IEEE 12th International Conference on Embedded Software and Systems, H*, 166–173. doi:10.1109/HPCC-CSS-ICESS.2015.246

Wu, D., & Gokhale, A. (2013). A self-tuning system based on application profiling and performance analysis for optimizing hadoop mapreduce cluster configuration. *20th Annual International Conference on High Performance Computing, HiPC 2013*, 89–98. doi:10.1109/HiPC.2013.6799133

Xu, L., Li, M., Zhang, L., Butt, A. R., Wang, Y., & Hu, Z. Z. (2016). MEMTUNE: Dynamic Memory Management for In-Memory Data Analytic Platforms. *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, 383–392. doi:10.1109/IPDPS.2016.105

Zhang, H., Liu, Z., & Wang, L. (2018). Tuning performance of spark programs. *Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018*, 282–285. doi:10.1109/IC2E.2018.00057

Zhang, K., Tanimura, Y., Nakada, H., & Ogawa, H. (2017). Understanding and improving disk-based intermediate data caching in Spark. *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, 2508–2517. doi:10.1109/BigData.2017.8258209

Zhang, R., Li, M., & Hildebrand, D. (2015). Finding the big data sweet spot: Towards automatically recommending configurations for hadoop clusters on docker containers. *Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015*, 365–368. doi:10.1109/IC2E.2015.101

Zhou, P., Ruan, Z., Fang, Z., Shand, M., Roazen, D., & Cong, J. (2018). Doppio: I/O-Aware Performance Analysis, Modeling and Optimization for In-memory Computing Framework. *Proceedings - 2018 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2018*, 22–32. doi:10.1109/ISPASS.2018.00011