International Journal on Advanced Science Engineering Information Technology

Vol.7 (2017) No. 5
ISSN: 2088-5334

Optimizing Parallelism of Big Data Analytics at Distributed Computing System

Rohyoung Myung[#], Heonchang Yu[#], Daewon Lee[*]

[#] *Dept. Computer Science & Engineering, Korea University, 145 Anam-ro, Seongbuk-gu, Seoul, 02841, Korea*
E-mail: {mry1811, yuhc}@korea.ac.kr

**Dept. of Computer Engineering, Seokyeong University, 124 Seogyeong-ro Seongbuk-gu, Seoul, 02173, Korea*
E-mail: daelee@skuniv.ac.kr

***Abstract*— Since advent of information revolution, there have been a lot of interest at big data analytics as well as big data. In the big data analytics, it is essential that not only extracting valuable information from the big data but also processing the data rapidly. Therefore, the distributed computing systems which process the analytics concurrently with parallel programming model based distributed processing framework as well as provide data analytics related libraries get attention of researchers. Several big data analytics programming models are studied that implemented for processing and generating huge data sets. However, developing the big data analytics in the distributed computing systems with utilizing parallel processing framework needs expertise in each area. In this paper, we demonstrate there is huge gap among usages of processing units if the big data analytics are naively executed at the distributed system. And we also prove that applying proper parallelism of those methods results in 1.5 to 3.3 times improvement of execution time compared to default parallelism.**

***Keywords*— big data analytics; distributed computing system; distributed processing framework; parallel programming model**

I. INTRODUCTION

Recently, since there has been huge interest in big data, researchers are getting more interest in big data analytics and systems for executing those analytics. Also, as the volume of the data is sharply increased, enhancing processing performance of existing big data analytics and systems which process them is highly issued by researchers [1]. For processing the data rapidly, distributed computing systems [2,3,4,5] that support distributed parallel processing at multiple computing nodes cluster and Distributed processing frameworks [2,5] that support implement of big data analytics at the systems are widely used.

The distributed processing frameworks provides programmers to high level interfaces in the distributed computing systems for taking advantage of multiple computing nodes of the systems and multicore processor at each computing node. The distributed computing systems manages necessary resources for executing a program as well as supports fault tolerance for making a program be processed normally. The systems also provide built in libraries for developers to implement programs more easily. However, developing the big data analytics in the distributed computing systems with utilizing parallel processing framework needs expertise in each area. Because it requires designing programs with concerning available resources of the systems, parallelizing the program with programming model and related tools of the framework, and reflecting characteristics of the data such as volume and structures.

Several big data analytics programming models are studied that implemented for processing and generating huge data sets. Mapreduce is one of them that has been successfully implemented for large-scale data-intensive applications on commodity clusters. Mapreduce consists of two functions. The map function processes a key/value pair to create a set of intermediate key/value pairs, and the reduce function merges all intermediate values. However, most of big data analytics programming models such as Mapreduce are built around an acyclic data flow model. It's is not suitable for other popular applications. Spark [2] is new cluster computing framework, that supports applications while retaining the scalability and fault tolerance of MapReduce.

In this paper, we conduct a research on enhancing performance of multiple big data analytics by optimizing parallelism of those techniques at Spark which is one of the most famous distributed computing systems. The first result of our research demonstrates that default implementations of the big data analytics cannot fully utilize processing resource

of the system. The amount of every core's utilization which process partitions of the analytics is different from the amounts of others. And then we found there can be at least 1.56 to at most 3.3 times reduction in execution time of those techniques if proper parallelism is applied to the implementations.

II. Material and Method

A. Related Works

For implementing the big data analytics and executing them concurrently, the distributed processing framework and the distributed computing system should be needed. The distributed systems not only support the resources for executing the analytics but also provide various libraries for convenient development of those techniques. The distributed processing framework provides high level programming interface for implementing the general parallel programming models which are based on concurrent parallel processing in the distributed systems. However, the components which constitute the system are too complex to optimize performance of those techniques at the distributed systems.

There are many big data analytics and they have a very wide range of execution performance difference. If input data of them are too large or computation resources are not appropriately utilized some of which have really higher computation complexity can largely be reduced by those factors. Others may also get affected by those reasons which could result in significant difference between the expected execution performance and the real execution performance of them. Therefore, when the big data analytics are implemented at the distributed systems, optimization of execution performance is required for not being largely affected by the size of the data but also scale of the system's available computing resources.

MapReduce programming model[6] is an algorithm for distributed parallel execution of big data analytics. It consisted of map and reduce procedures. The map procedure performs distributed filtering and sorting and the reduce procedure performs a combine operation as a summary. The model also has redundancy and fault tolerance so that it is commonly used to implement big data analytics. However, if the appropriate number of task partitions is not applied to the implementations, efficiency of the system's resources and execution performance of the implementations can be severely degraded.

Most of MapReduce systems are built around an acyclic data flow model which is not suitable to some cases of applications. To overcome such flaw, [2] introduced Spark. Spark is in-memory based framework which is suitable to cyclic data flow model especially machine learning algorithm. In the experiment, it outperforms Hadoop by 10x. To process iterative jobs resiliently, Spark use resilient distributed datasets (RDDs) which is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost.

In [8], 'CHOPPER' is proposed which dynamically optimize the number of parameters and partition scheme in the middle of execution. In general, the performance of in-memory based framework is affected by the method of data partition because it is subordinated to the memory performance. In its experiment, it boosts up to 1.35x as compared with the default option of Spark. One of the reasons why it is faster than the baseline is it minimizes the stage execution time and shuffle traffic. However, it is training-based model that when the available resources are changed, it needs re-training. Therefore, it is not appropriate to adapting cloud-based system.

Marcu [9] compares two of the most famous and fast big data analytics frameworks: Spark vs Flink. In the experiment, Spark is about 1.7x faster than Flink for large graph processing, while the latter outperforms Spark up to 1.5x for batch and small graph workloads using sensitively less resources and being less tedious to configure. The difference of the performance is derived from design choices like memory management, optimizations and parameter configuration.

There are representative two workloads were developed to enhance the batch-oriented Hadoop with iterative support.

- Batch workloads: Word count, grep and sort are used in various real applications such as LHC[10], google[11], amazon[12], and so on.
- Iterative workloads: K-Means, Page Rank and Connected Components are frequent in machine learning algorithms [13,19,20] and social graphs processing (at Facebook [14] or Twitter [15]).

TABLE I
Operators used in each workload

Operators	Batch (one pass)			Iterative (caching)		
	WC	G	S	KM	PR	CC
map			○	○	○	○
flatMap	○				○	○
mapToPair	○					
reduceByKey	○			○		
collectAsMap				○		
filter → count		○				
distinct					○	○
repartitionAndSort WithinPartitions			○			
Graph specific operators					○	○
coalesce, mapPartitionsWithIndex					○	○
save	○		○	○	○	○

Table I shows the most important operators' usage by each workload. These operators have basic core and specific modules by the libraries of each framework.

- Word Count(WC): a simple metric for measuring article quality by counting the total number of occurrence of each word.
- Grep(G): a common command for searching test data sets.
- Sort(S): a sorting algorithm suitable for measuring the I/O and the communication performance of the two engines.
- K-Means(KM): an unsupervised method used in data mining to group data elements with a high similarity.
- Page Rank(PR): an unsupervised method used in data mining to group data elements with a high similarity.
- Connected Components(CC): an important topological invariant of a graph

B. *Optimizing Parallelism of Big Data Analytics at Distributed System*

In this section, we describe execution procedure of big data analytics and optimizing big data analytics' parallelism. We focused on how to optimize big data analytics' parallelism on general execution procedure of big data analytics at distributed system.

1) Execution procedure of big data analytics

Distributed computing system[2] consists of a master node which manages overall processing procedure of a program and multiple computing nodes which compute small, partitioned tasks in parallel. Developers take advantage of parallel programming model based distributed processing framework to implement the big data analytics. Fig. 1 shows the general execution procedure of big data analytics at distributed system.

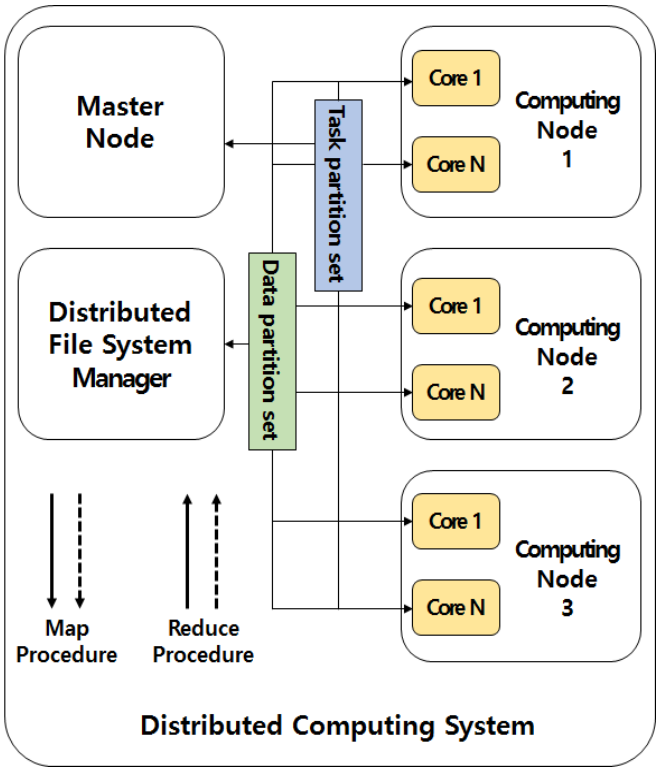

Fig. 1 General execution procedure of big data analytics at distributed system

The execution procedure detail of the analytics is given as follow. The input data which is required for each analytic is partitioned to small data partitions and the number of them is decided by the framework's parallelism policy. Then, the partitioned data is allocated into Computing Nodes(CN) by Master Node(MN)'s scheduler(In spark, the default policy is FIFO). The real data is transferred to CNs by distributed file system manager(e.g. HDFS) and the manager sometimes replicate the whole data into the CNs for enhancing processing performance of the system. After that, CNs process transferred data in parallel with multiple cores and merge the outputs of the processes when results should be synchronized. Finally, the analytic is terminated with returning the merged outputs of CNs.

TABLE II
AN IMPLEMENTATION EXAMPLE OF 'WORD COUNT' AT SPARK

```
lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])
counts = lines.flatMap(lambda x: x.split(' ')) \
          .map(lambda x: (x, 1)) \
          .reduceByKey(add)
output = counts.collect()
```

We show a specific example 'Word count' which is one of the most common data analytics in table II. The system proceeds the analytics' tasks related to map procedure with 'map' and 'flatMap' API supported by the framework and reduce procedure with 'reduceByKey' and 'collect' API [16].

TABLE III
AN IMPLEMENTATION EXAMPLE OF 'K-MEANS' AT SPARK

```
from numpy import array
from math import sqrt

from pyspark.mllib.clustering import KMeans, KMeansModel

# Load and parse the data
data = sc.textFile("data/mllib/kmeans_data.txt")
parsedData = data.map(lambda line: array([float(x) for x in
line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared
Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point))
.reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load model
clusters.save(sc,"target/org/apache/spark/PythonKMeans
Example/KMeansModel")
sameModel = KMeansModel.load(sc, "target/org/apache/spark/
PythonKMeansExample/KMeansModel")
```

Table III shows an example 'K-means' which is more complex than 'Word count'. After loading and parsing data, this example use the K-means object to cluster the data into two clusters. The number of supposed clusters is passed to the algorithm. Then, this example computes within set sum of squared error (WSSSE). By increasing k, we can reduce this error measure. Usually, the optimal k is one in the WSSSE graph.

2) Optimizing big data analytics' parallelism

A minimum processing unit of the CN is a task partition. An implementation of single data analytic is divided into multiple task partitions and processed at multiple CNs in parallel. However, if the number of task partition is improperly decided, execution time deviation of each CN is extremely large. There are 3 case of processing.

TABLE IV
TOTAL EXECUTION TIME OF ALL CNS IN CASE OF PROCESSING 'WORD COUNT', 'K-MEANS CLUSTERING', AND 'SORT'

Time unit: minute		CN 1	CN 2	CN 3	CN 4
Word count	5GB	15	14	23	14
	10GB	34	29	37	27
	15GB	52	50	50	44
K-means clustering	10GB	80.9	81.4	115	81.4
	15GB	163	157	231	169
Sort	0.01GB	0.68	0.72	0.75	0.77
	0.05GB	1.91	2.43	1.83	2.32
	0.1GB	4.45	4.07	3.7	4.2

Table IV shows that in case of 'K-means clustering' which handles 15GB, CN 2 processed it 157 minutes while CN 3 processed it 231 minutes which is 74 minutes longer than that of CN 2. As a result, the parallelism of the analytics' implementation so severely affects the usage of CNs that usage of CNs can be extremely different.

We linearly increased the total number of partitions for optimizing execution performance of the implementations and found the lowest bound of execution time for each case. Moreover, we verified each analytic has different appropriate number of task partitions and the optimal number of every analytics' task partitions is different if input data of them vary in size. We also recognized that after the point which has minimum execution time, there is almost no difference whether the number of task partitions is increased. And there is large performance decrease when the number of task partitions is increased too much.

It means that although increasing the number of task partitions makes usage of the system resources high, there is also additional overheads such as scheduler delay of task partitions and context switching overhead as more number of task partitions are uploaded to cores. In our future work, we will analysis factors which are closely related to performance of processing task partitions and find the optimal number of task partitions which makes execution time minimized after only one execution of specific implementation. We also have plan to research scheduler of task partitions in the distributed computing system for promoting efficiency of CNs.

III. RESULT AND DISCUSSION

The experiment environment of our research is progressed with single cluster with 4 desktops. Each one consists of Intel i7 8 cores(4 hyperthread cores included) 3.4GHz processor, 16GB memory, 256GB SSD. And we used Ubuntu 14.04 as OS, Spark 2.0.2, Java 1.7, Python 2.7, and Scala 2.11 for our experiment. The Spark consists of 1 MN and 4 CNs and we added 8GB memory to the desktop which has the MN. In this experiment, we are focused on total execution time. For the batch workloads, our goal was to validate strong and weak scalability. We further analyze the resource usage that focused on scalability, caching and pipelining performance.

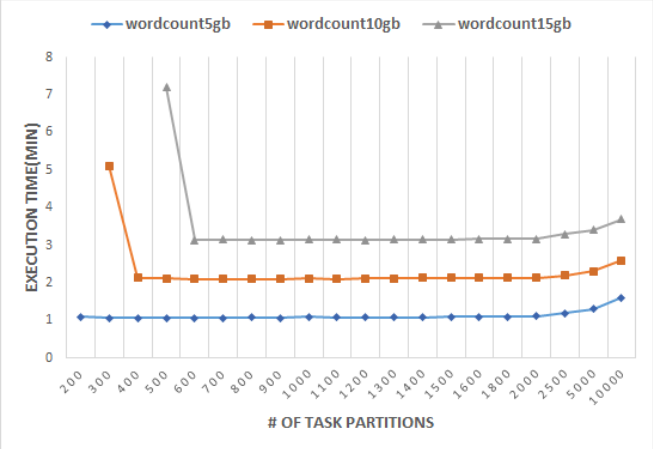

Fig. 2 Execution time curves of the big data analytics as the number of word count is increased

Fig. 2 shows execution time curves of each method as the number of word count is increased when the size of input data varies. The 'word count' 5gb, 10gb, 15gb has minimum execution time when the number of task partitions are 400, 600, 900 and the execution time of them is increased after the number of task partitions becomes 2000. This means that even though the input data is increased, increasing the number of task partitions not always decreases the executions time of the method.

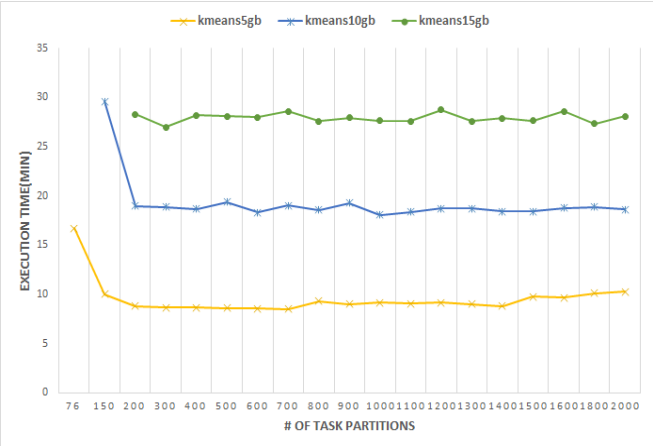

Fig. 3 Execution time curves of the big data analytics as the number of K-means clustering is increased

Fig. 3 shows execution time curves of each method as the number of K-means clustering is increased when the size of input data varies. The 'K-means' 5gb, 10gb, 15gb has minimum execution time when the number of task partitions are 700, 1000, 300 and the average execution time of them is 9.12, 18.70, and 27.93. Each of them is generally over each average after the number of task partitions becomes 1200. This also shows the input data is increased, increasing the number of task partitions not always decreases the executions time of the method.

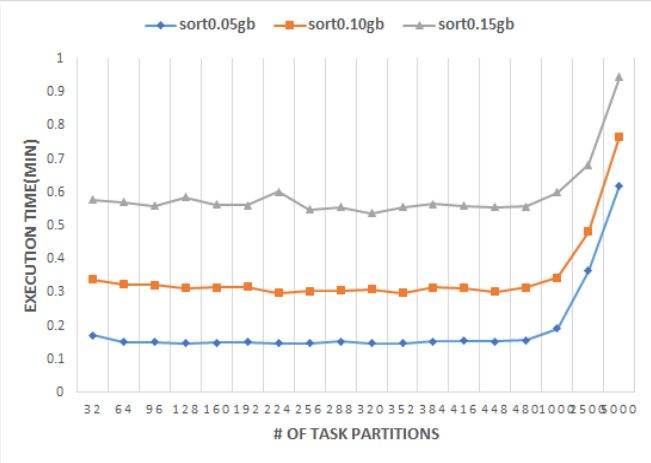

Fig. 4 Execution time curves of the big data analytics as the number of sort is increased

Fig. 4 shows execution time curves of each method as the number of sort clustering is increased when the size of input data varies. The 'sort' 0.05gb, 0.10gb, 0.15gb has minimum execution time when the number of task partitions are 224, 224, 320 and the execution time of them is increased after the number of task partitions becomes 500. This means that reordering the operators drastically reduces the execution time and enables more efficient resource usage.

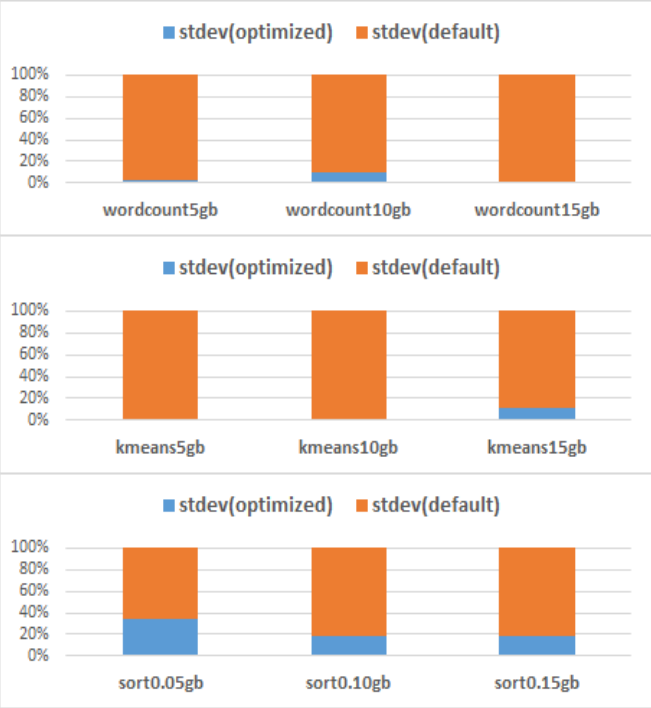

Fig. 5 Standard deviation rate of optimized and default implementations' processing time for all CNs

Fig. 5 shows standard deviation(stdev) rate of optimized and default implementations' processing time for all CNs when the input size of the data varies. In case of optimized ones, all the implementations has relatively low processing time to that of default ones. Specially, 'Wordcount' 5gb, 15gb, and 'K-means' 5gb, 10gb have nearly 0 when they are compared to default ones. It means that if proper parallelism is applied, usage of the system resource would be higher than when it is not applied.

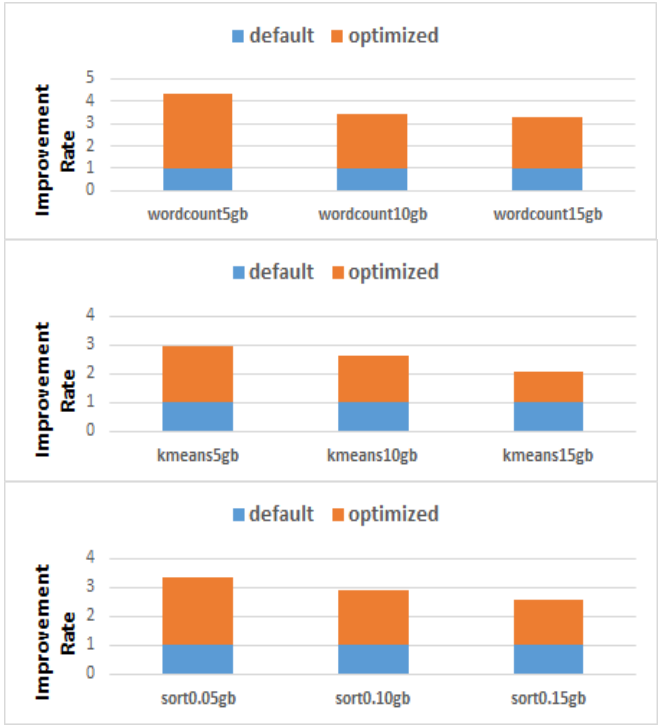

Fig. 6 Improvement rate of optimized and default implementations' processing time for all CNs

Fig. 6 shows improvement rate of optimized and default implementations' processing time when the input size of the data varies. In case of wordcount, there is at most 3.33 times higher than execution performance of the default with 5gb input data and at least 2.29 times performance improvement of the default with 15gb input data. Although kmeans with 15gb has 1.04 times performance enhancement, but most cases have been improved more than 1.5 times execution performance of the default.

IV. Conclusion

Nowadays, by information has increased explosively, enhancing processing performance of big data analytics and systems is highly issued by researchers. Several big data analytics programming models are studied that implemented for processing and generating huge data sets. However, developing the big data analytics in the distributed computing systems with utilizing parallel processing framework needs expertise in each area.

In this paper, we researched on enhancing performance of processing time by applying parallelism optimization when implementing big data analytics with parallel programming model based distributed processing framework at distributed computing system. Our study proved that if the optimization of parallelism is not applied, the implementations of the methods cannot fully utilize resources of the system. And we also demonstrated if proper parallelism is applied to the implementations, there can be at least 1.5 to at most 3.3 times performance enhancement.

In this paper, we only focused on optimizing parallelism of spark by configuring partitioning parameter. However, there is plentiful parameters that have close relationship on optimizing overall performance of spark. Since parameters have dependency among them, simple idea or single heuristic which tunes all of them is almost impossible as

well as inefficient at the aspect of executing performance. In our future works we will deal with more parameters such as network shuffle parameters, system resource utilization parameters, etc which can seriously impact on the Spark. We will also tune those parameters taking advantage of the latest machine learning techniques which are variety of deep learning models: “Deep Belief Network”, “Convolutional Neural Network”, and “Recurrent Neural Network”.

ACKNOWLEDGMENT

This Research was supported by Seokyeong University in 2014. *Corresponding Author: Daewon Lee (daelee@skuniv.ac.kr)

REFERENCES

[1] Alsheikh MA, Niyato D, Lin S, Tan H-P, Han Z. Mobile big data analytics using deep learning and apache spark. IEEE Network. 2016;30(3):22-9.

[2] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster Computing with Working Sets. HotCloud. 2010;10(10-10):95.

[3] Power R, Li J, editors. Piccolo: Building Fast, Distributed Programs with Partitioned Tables. OSDI; 2010.

[4] Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, et al., editors. Pregel: a system for large-scale graph processing. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data; 2010: ACM.

[5] Isard M, Budiu M, Yu Y, Birrell A, Fetterly D, editors. Dryad: distributed data-parallel programs from sequential building blocks. ACM SIGOPS operating systems review; 2007: ACM.

[6] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM. 2008;51(1):107-13.

[7] Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, et al., editors. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation; 2012: USENIX Association.

[8] Paul AK, Zhuang W, Xu L, Li M, Rafique MM, Butt AR, editors. CHOPPER: Optimizing Data Partitioning for In-memory Data Analytics Frameworks. Cluster Computing (CLUSTER), 2016 IEEE International Conference on; 2016: IEEE.

[9] Marcu O-C, Costan A, Antoniu G, Pérez-Hernández MS, editors. Spark versus flink: Understanding performance in big data analytics frameworks. Cluster Computing (CLUSTER), 2016 IEEE International Conference on; 2016: IEEE.

[10] Aad, G., Abat, E., Abdallah, J., Abdelalim, A. A., Abdesselam, A., Abdinov, O., Acharya, B. S, et al., CERN: The Large Hadron Collider. Choice Reviews Online. 2009;46(08):46-4503-46-4503.

[11] Knill, Emanuel. "Physics: quantum computing." nature 2010;463(7280): 441-443.

[12] Singh L, K. Bharti R. Comparison among different Cryptographic Algorithms using Neighborhood-Generated Keys. International Journal of Computer Applications. 2013;73(5):40-43.

[13] Anjeneya Swami Kare. A Simple Algorithm For Replacement Paths Problem. Electronic Notes in Discrete Mathematics. 2016;53:307-318.

[14] Lee C. Find Them on Facebook: Using Facebook to Reach Students Where They Already Go. SSRN Electronic Journal.

[15] Azar P. The Wisdom of Twitter Crowds: Predicting Stock Market Reactions to FOMC Meetings via Twitter Feeds. SSRN Electronic Journal.

[16] Aleksiyants A, Borisenko O, Turdakov D, Sher A, Kuznetsov S. Implementing Apache Spark jobs execution and Apache Spark cluster creation for Openstack Sahara. Proceedings of the Institute for System Programming of RAS. 2015;27(5):35-48.

[17] Song J. Performance and Energy Optimization on Terasort Algorithm by Task Self-Resizing. Information Technology And Control. 2015;44(1).

[18] Kadhum AM, Hasan MK. Assessing the Determinants of Cloud Computing Services for Utilizing Health Information Systems: A Case Study. International Journal on Advanced Science, Engineering and Information Technology. 2017;7(2):503-10.

[19] Yun, Y., Hooshyar, D., Jo, J. Lim, H. Developing a hybrid collaborative filtering recommendation system with opinion mining on purchase review. Journal of Information Science; 2017: Chartered institute of library and information association.

[20] Lee, S., Hooshyar, D., Ji, H. Nam, K. Lim, H. Mining biometric data to predict programmer expertise and task difficulty. Cluster Computing; 2017:1-11.