EasyChair Preprint

№ 2567

Distributed Computing for Advanced Smart Meter Data Management with focus on Electrical Utility Applications

Ameema Zainab, Shady S. Refaat, Haitham Abu-Rub and Othmane Bouhali

February 5, 2020

Distributed Computing for Smart Meter Data Management for Electrical Utility Applications

Ameema Zainab
Electrical & Computer Engineering
Texas A & M University
College Station, Texas, USA
azain@tamu.edu

Shady S. Refaat
Electrical & Computer Engineering
Texas A & M University
Doha, Qatar
shady.khalil@qatar.tamu.edu

Haitham Abu-Rub
Electrical & Computer Engineering
Texas A & M University at Qatar
Doha, Qatar
haitham.abu-rub@qatar.tamu.edu

Othmane Bouhali
Electrical & Computer Engineering
Texas A & M University
Doha, Qatar
othmane.bouhali@qatar.tamu.edu

***Abstract*— With the advent of Internet-of-Things (IoT) devices, including smart meters and sensors in the smart grid, there has been immense research interest in big data management, analytics, and parallel processing of data. However, complex hardware and software parameters configurations and in-depth understanding of the data processing design are essential for efficient utilization of big data analytics platforms. In this work, we analyze the parallelization of load prediction by utilizing spark regression python library to assess the performance with workloads of up to 8 nodes. The results of different configurations have been studied and analyzed against the performance of Apache Spark. It was found that a trade-off between the number of nodes and cores is necessary to perform efficient parallel computing. Multiple sets of combinations of nodes and cores are considered in this paper to evaluate the performance. The work also signifies the importance of high-performance computing capability for smart meters big data management. The obtained results indicate that the computational time is not only dependent on the data size but also on the number of compute nodes and the number of cores assigned to run the job.**

Keywords— Smart grid, load forecast, big data, parallel computing, Apache-Spark, high performance computing

I. INTRODUCTION

Smart meters play an important role in the smart grid big data. The data is collected normally at a rate of 5, 10 or 15 minutes, which accounts for approximately 35040 data points per meter per year [1]. Considering the number of smart meters installed in a grid, the amount of data generated is increasing tremendously. Performing load forecast on such huge amounts of data is challenging. Our work focuses on performing load forecasting on the smart meters data in a distributed computing environment. The proposed work intends to understand the performance with respect to distributed allocation of resources (computational nodes and cores) to perform the analysis.

Big data cannot be analyzed with the traditional tools and techniques but needs big data technologies because of the volume, variety, and veracity. Spark's parallelization has been used to propose the shortest path graph computing to solve the problem of finding the shortest path in a large number of vertices [2].

The data analyst would rely on the snapshots of data and hoped to capture the characteristics of the whole data in the snapshot. However, due to the recent innovation and emerging trends, the availability of the computing resources has increased and led to a decrease in the price of the resources. The breakthroughs in hardware and software have led to the revolution in the big data management industry. The currently available innovative software tools accommodate computing and data nodes to be used as one big pool of resources where computing, storage, networking, and processes can be moved from one node to another with ease and efficiency.

With the virtualization technology over resources, it has been possible to deal with the problem of latency that existed whilst managing big data. The real-time applications and processing of big data are possible because of the support of distributed computing and parallel processing.

Apache Spark is an open source cluster computing framework for big data management, processing and analytics with various built in tools [4]. It has rightfully succeeded the Apache Hadoop MapReduce framework with few improvements to it. Apache Spark is reported to be 100 times faster than Apache Hadoop when configured correctly and used with appropriate job scheduler [5] [6] [7] [8]. It provides data abstraction, distributed data frame, linear scalability, and fault tolerance. Spark framework is not just significant in batch processing of data but also in iterative querying and real-time streaming of data.

However, a thorough analysis of the Spark framework is crucial to guide on the selection and configuration of big data analytics framework like Apache Spark. In this work, a cluster is set up with a varying number of nodes and a varying number of cores to evaluate the performance of Apache Spark for parallel processing with respect to the application of load forecasting in smart grids. By varying the number of nodes, memory per node and number of cores per node, we examine the scalability of the Spark computing engine when additional computational resources are made available. This cross-performance evaluation will identify the factors leading to the potential performance degradation of parallel computing. The main contributions of this paper are (1) emphasis on the importance of parallel processing to perform load forecasting, and (2) trade-off between the optimal number of nodes and cores to perform distributed processing.

The remainder of the paper is organized as follows. Section II presents the information on the background work and also, provides literature review on the performance evaluation of big data analytics platform. Section III details the proposed methodology and illustrates the results of the performance evaluation of apache Spark for different combinations of nodes, cores and memory. Finally, concluding remarks are given in section IV of the paper.

II. RELATED WORK

Due to the widespread adoption of the big data platforms such as Apache Hadoop and Spark, several research works have been done to improve the performance of these platforms [9] [10] [11] [7] [12] [13]. Most of the research works tried to improvise the execution of the jobs on the platforms using the common knowledge of the working of these platforms. For instance, the optimizations in the network are obtained by increasing the network throughput [14]. The optimizations in the storage infrastructure are obtained by the use of large in-memory cache [15]. In addition to the factors of network throughput and large in-memory cache, the tradeoff between the number of cores, number of nodes and memory utilization for the optimized execution of jobs in faster times, is also an important factor to be considered. If the jobs are executed on hundreds of cores without any thought on the appropriate parameterization and optimization, this yields poor efficiency and high cost of computations with big data platforms [16]. Also, according to literature, it may give even degraded results owing to overheads [17]. In this paper, a systematic analysis has been performed to evaluate the performance of Spark to define optimized parameters of number of cores and number of nodes for load forecasting in the electric power utility.

In [18], Green et al. dealt with the applications and trends of high-performance computing for electrical power grids. The usage of high-performance computing in the smart grid, specifically with regards to the analysis, control, and design of electrical systems. They mention a large amount of parallelism that is available in algorithms commonly connected with the electrical power systems analysis. They describe the methodologies, technologies, and algorithms like cloud computing, GPU computing, parallel meta-heuristic algorithms, etc. that will provide enhanced distributed processing performance and efficiency. In [19], Marcu et al. evaluated the performance of Spark and Flink for different architectures and different parameter configurations. They compared the performance of Spark and Flink for different case study problems of word count, evaluation of text search, bulk iterations performance, and tera sorting. The performance has been compared in the metrics of scalability, resource usage, and CPU utilization. However, their results presented that it is not possible to conclusively present that one platform outperforms the other. Also, their work did not generalize the effect of the number of cores, number of nodes and memory utilizations with a vast majority of workloads.

Apache Spark tries to the maximum to avoid the usage of file system and uses the in-memory implementation i.e. it stores the intermediate results of the operations across phases in a job so that the machine learning and clustering applications and methods have the results from each iteration available for further iterations. This gives an edge to Apache Spark over Apache Hadoop. Apache Spark uses an abstraction paradigm called Resilient Distributed Dataset (RDD) which abstracts the details of distribution. RDD is based on the transformations (namely map, filter, and join) on the datasets and these transformations are applied multiple times on a dataset. In this work, the memory for the nodes is selected in such a way that the memory can hold the RDD input in the main memory.

III. METHODOLOGY AND RESULTS

A. Software specification

Python programming has been utilized to run the Sparks MLib machine learning library Linear regression [20].

- Spark/2.4.0-intel-2018b-Hadoop-2.7-Java-1.8-Python-3.6.6
- SLURM job scheduler

B. Cluster specification

The experiments are performed on the Terra cluster1 provided by Texas A&M High Performance Research Computing. The configuration of the cluster is given below:

- Total nodes: 320
- Operating system: Linux (CentOS 7)
- Memory/Node: Intel Xeon E5-2680 v4 2.40GHz 14-core
- Cores/Node: 9,632
- Processor Type: IBM Power7+ , 8-core, 4.2 GHz
- File system: General Parallel File System (GPFS)
- Interconnect: Intel Omni-Path Fabric 100 Series switches.

Three scenarios have been considered to evaluate the performance of apache spark with respect to the different number of nodes, cores, and memory. Fig. 1 shows a schematic view of the test cases.

Fixed amount of memory: In this set of experiments, the total number of cores for all the nodes considered together is fixed and the number of nodes or number of cores per node is varied as: 8 node - 1 core/node, 4 nodes - 2 cores/node, 2 nodes - 4 cores/node and 1 node - 8 cores/node.

Fixed number of cores per node: In this set of experiments, the number of cores per node is fixed, and the number of nodes is varied. i.e. Each node has fixed four cores per node and the number of nodes is varied.

Fixed number of nodes: In this set of experiments, the number of nodes is fixed to 8, and the number of cores per node is varied.

1) Fixed amount of memory:

Test Case I

[1]Portions of this research were conducted with the advanced computing resources provided by Texas A&M High Performance Research Computing https://hprc.tamu.edu/wiki/Terra

This work aimed to evaluate the performance of the big data platform, Apache Spark and compare its performance with no Spark Machine Learning. This quantitatively evaluates and compares the performance of standalone machine learning libraries against the distributed machine learning models.

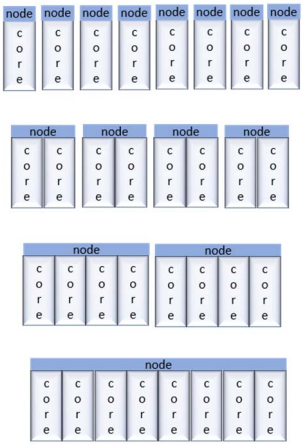

Fig. 1. Distribution of nodes and cores/node

Also, it has always been expected that the performance of no Spark is better than the Spark when the data set is not big. The methodology of performance evaluation is tested on different sizes of data. The data set used in our methodology ranges from ~9000 rows to ~ 2250000 rows.

In the initial experiments, the total number of cores considering all the nodes are kept constant while the number of nodes is altered. The experiments are performed on different data sizes; 1) 9000 rows data, and 2) 2250000 rows data.

Figure 1, represents the performance of no spark and spark on a smaller data set with 9000 rows. It is very evident from the graph that the performance of no spark is much better when compared to the spark run time. The spark data frame performs data abstraction before the analysis of data, and the profiling of different execution steps revealed that the data abstraction of spark methodology usually takes a lot of time and this is very evident in the case of smaller datasets.

Figure 2 represents the performance of no spark and spark on a dataset of 2250000 rows. The figure illustrates that even though the runtime for no spark is still less comparatively for a dataset that has increased ~250 times, the spark run time in seconds has not increased accordingly. The increase in the performance is seen in the reduction of the run time given in seconds. The reduction in run time is from 30 secs to 15 secs for data size of 9000 rows and from 180 secs to 40 secs for data size of 2250000 rows. According to the literature, when the memory is sufficient to include the processing of the data set, no spark always performs better than a spark. Spark would perform better when the data size is beyond the threshold of the capacity of no spark applications explained in the later sections. Also, Spark performs better when the overhead of distribution is less than the overhead of the processing itself.

Key take away from Fig. 2 and Fig. 3 is that as the number of cores is increasing from 1 to 8, the performance of Apache

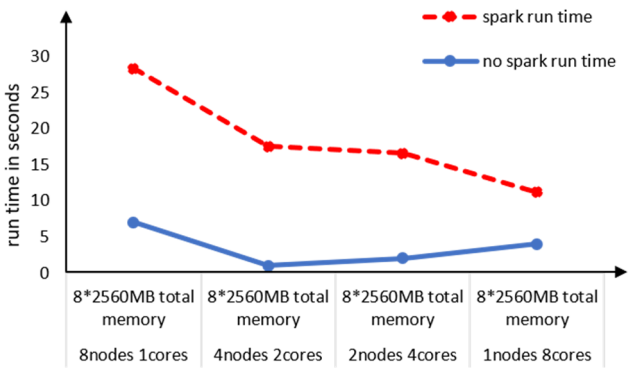

Fig. 2. Performance comparison of spark and no spark for a fixed number of total cores where the number of nodes is a variant

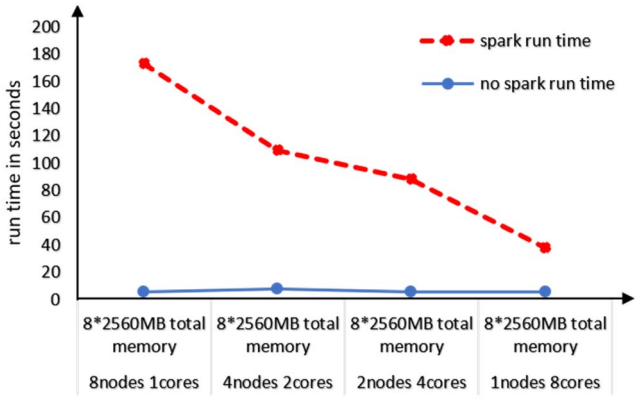

Fig. 3. Performance comparison of spark and no spark for a fixed number of total cores where a number of nodes is a variant

Spark gets better holding onto the essence of distributed computing.

Test Case II

The data size has been further increased to evaluate the performance of spark and no spark keeping the total number of cores to be fixed across nodes. The data size is varied from a range of 10161600 rows (~1 GB) to 203232000 rows (~18 GB). For small amounts of data, the performance for the standalone machine learning library without spark is better than distributed machine learning using Spark. However, as the data size increases, distributed computing using Spark performs better than the non-distributed library as providing sufficient amount of memory is necessary in the latter case. This signifies the importance that for big data with large data sizes, load forecasting needs to be performed using parallel processing or distributed computing.

During the experiments, it was noticed that for the data size of 203 million rows, standalone no-spark machine learning does not successfully perform computations because the data does not fit in the memory of one node (Table 1). However, for the Spark machine learning library, the fitting of data in the memory of one node is not required and hence, Spark performs computations on the large volume of data without out of memory error. Table 1, shows that for a same amount of memory provided, distributed computing executes the job of load forecasting successfully, but the standalone runs out of memory and fails the job execution. This justifies

the statement mentioned in the previous sections. The failure of running the load forecasting on no-Spark platforms clearly calls for a need of distributed computing to perform load forecasting in the smart grid. The no spark jobs have run successfully when the allocated memory has been increased. Upon testing, if the same job is to be successful on the no-spark standalone platform, then it requires about 40 GB of memory per core for a configuration of 4 cores per node. While spark performs the job successfully with 8 GB of memory per core.

TABLE 1 RUN TIME FOR SPARK AND NON-SPARK EXECUTIONS

Total memory 64GB[a]				
	Nodes	**Cores per node**	**Memory per core (GB)**	**Run time (s)**
Spark (distributed)	2	4	8	3657.3
Sklearn (non-distributed)	2	4	8	Out of memory

[a.] Data size of 203,232,000 rows (~18 GB)

[b.]

Fig. 4 shows the comparison between the Spark and no Spark libraries in terms of execution time in seconds when the data size is increased between the scale of 10.2 million rows to 203.2 million rows. As seen in Fig. 4, the run time for Spark is still large compared to no Spark, but this is due to the data size not being beyond the threshold of memory of one node. Moreover, when the data size increases beyond a threshold, the no-spark does not yield results, when the spark gives a good performance. Also, there are many other factors that need to be optimized to acquire better performance in Spark. The optimizing factors include the number of stages, driver and executor memory assigned, number of cores, memory partition, etc.

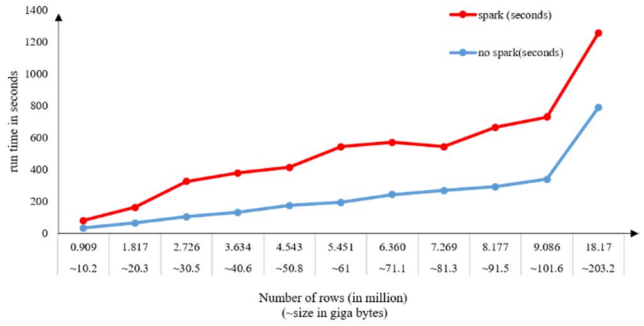

Fig. 4. Comparison of execution time for distributed spark library with standalone with increasing data size

Test Case III

Assigning any number of cores per node is not optimal. This test case justifies this condition.

Distribution of memory: Table 2 shows the distribution of 64GB memory into various nodes and cores. Observations:
8 cores per node performs better than 4 cores per node but the CPU efficiency is cut down to almost half.

Increasing the number of cores per node reduces time but affects the CPU and memory efficiency is an unexpected trend. Almost the same amount of run time is observed if the number of nodes is increased to 2 for 8 cores per node. Restricted the memory size to 20GB the memory efficiency has increased from close to 30% up to 90%.

Rows 2 and 9 in Table 2 clearly show that only 20GB memory is enough to run the same job achieving similar run times.

TABLE II RUN TIME FOR EXECUTIONS WITH DIFFERENT NUMBER OF NODES, CORES, AND MEMORY

Total of 64 GB memory[c]						
nodes	**Cores per node**	**Memory per core (GB)**	**CPU eff.**	**Memory eff.**	**Memory Utilized (GB)**	**Run time (s)**
1	4	16	80.36%	26.75%	17.12	622.5
1	8	8	58.49%	28.00%	17.92	364.5
1	10	6.4	53.95%	28.84%	18.46	351.3
1	12	5.3	00.09%	0.00%	01.10	261.5
1	16	4	00.06%	0.00%	01.09	274.9
2	4	8	37.17%	26.08%	16.69	668.9
2	8	4	23.60%	27.84%	17.82	392.6
3	8	2.6	19.44%	27.64	17.69	366.5
Total of 20 GB memory						
1	8	2.5	48.47%	88.20%	17.64	377.5
1	4	5	72.46%	88.21%	16.44	649.3

[c.] Data size of 30,484,800 rows ~ 2.7GB

For the given data size and computational needs, based on the observations from Table 2 it can be clearly stated that having 8 cores per node to run the job is optimal.

2) Fixed number of cores per node:

The number of cores per node have been fixed in this case and the number of nodes has been increased. Fig. 5 and Fig. 6 clearly show that increasing the number of nodes reduces the run time.

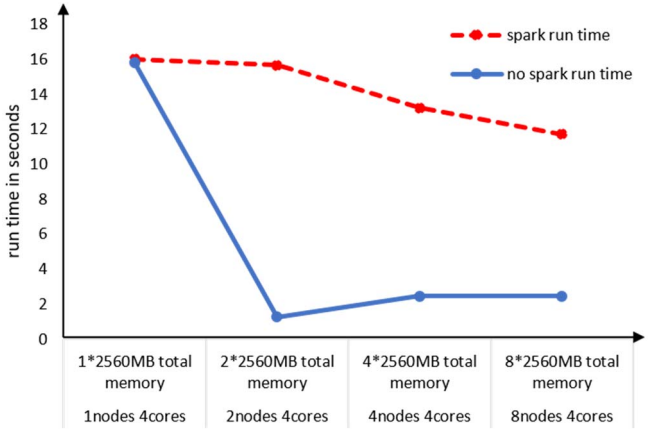

Fig. 5. Execution time for Spark and no Spark for a data size of 9000 rows with a fixed number of cores and varying the number of nodes.

However, the continuous increase of the number of nodes does not have a tremendous impact on the run time initially. However, with further increase in number of nodes, the

execution time definitely decreases for distributed computing using spark.

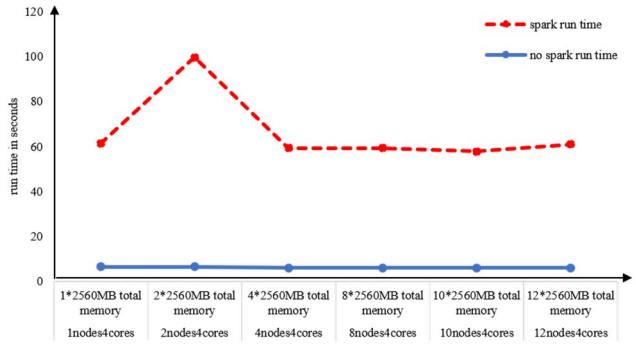

Fig. 6. Execution time for spark and no spark for a data size of 2250000 rows with a fixed number of cores and varying the number of nodes.

3) Fixed number of nodes:

The number of nodes has been fixed to be 8, and the number of cores per node have been varied. Fig. 7 and Fig. 8 depict that increasing the number of cores does not necessarily increase the performance in no spark standalone case.

The execution time goes significantly down by providing 4 or more cores in spark distributed processing, but the performance deteriorates a little after 8 cores. The observation for more than 8 cores clearly shows that the performance doesn't pick up with the increase in the number of cores.

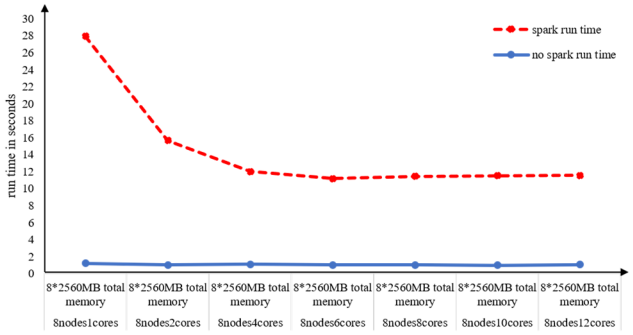

Fig. 7. Execution time for spark and no spark for a data size of 9000 rows with fixed number of nodes and varying the number of nodes.

Even though the run time has reduced but the memory and CPU efficiency are significantly disturbed. It can be clearly stated that an optimal number of cores is between 4 cores per node to 8 cores per node for spark distributed computing based on the data size.

IV. CONCLUSION

In this paper, the performance evaluation has been successfully done for apache spark and no spark platforms for varying number of cores, nodes and different memory sizes. It is noted that the no-spark standalone machine learning performs better than the distributed computing spark platform only when the data fits into the memory of one node.

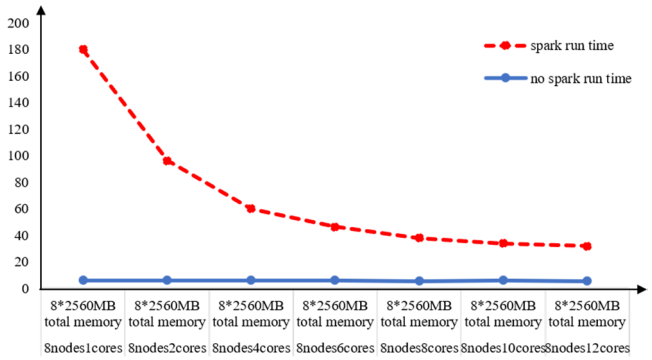

Fig. 8. Execution time for spark and no spark for a data size of 2250000 rows with fixed number of nodes and varying the number of nodes

But, when the data becomes sufficiently huge, the no spark standalone machine learning runs out of memory and spark performs considerately. The reason is that for small data sizes, the overhead of distribution of resources is very high than the overhead of computations itself. When the data size becomes sufficiently large, the overhead of distribution of resources becomes negligible and the performance of spark is definitely notable. Sufficient memory is definitely required for the execution of a successful job in a non-distributed condition.

A need for optimal number of nodes and cores has been studied and verified under multiple test cases. It has been noted that the performance increases with the increase in the number of nodes in a spark cluster. Also, the increase in the number of cores per node improves performance only when the number of cores is between 4 and 8. Beyond 8 cores per node, the performance deteriorates. However, having a very high amount of memory does not necessarily decrease the execution time.

Future work could focus on improving the computational time using parallel computing compared to the non-distributed fashion. Also, the experiments could be extended to deal with a very large datasets to evaluate the performance of Apache Spark. It will be interesting to evaluate Spark's performance when the number of nodes is very high.

ACKNOWLEDGEMENT

This publication was made possible by NPRP grant [NPRP10-0101-170082] from the Qatar National Research Fund (a member of Qatar Foundation) and the co-funding by IBERDROLA QSTP LLC. The statements made herein are solely the responsibility of the author[s].

REFERENCES

[1] M. Aiello and G. A. Pagani, "The smart grid's data generating potentials.," in *2014 Federated Conference on Computer Science and Information Systems*, 2014.

[2] Y. Arfat, S. Suma, R. Mehmood and A. Albeshri, "Parallel Shortest Path Big Data Graph Computations of US Road Network Using Apache Spark: Survey, Architecture, and Evaluation," *Smart Infrastructure and Applications,* 2020.

[3] K. Kambatla, G. Kollias, V. Kumar and G. Ananth, "Trends in big data analytics.," *Journal of Parallel and Distributed Computing,* vol. 74, no. 7, pp. 2561-2573, 2014.

[4] M. e. a. Zaharia, "Apache spark: a unified engine for big data processing," 2016.

[5] B. H. e. al., "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center".

[6] M. A. J. a. M. G. A. B. Yoo, SLURM: Simple Linux Utility for Resource Management, Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 2862, pp. 44–60, doi: 10.1007/10968987_3, 2003.

[7] B. J. Mathiya and D. Vinodkumar L., "Apache hadoop yarn parameter configuration challenges and optimization," in *International Conference on Soft-Computing and Networks Security (ICSNS)*, 2015.

[8] W. W. L. O. T. G. X. L. H. a. J. J. X. Wei, Integrating local job scheduler - LSFTM with GfarmTM, in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 3758 LNCS, pp. 196–204, doi: 10.1007/11576235_25, 2005.

[9] J. Zhang, W. Jian-Syuan, L. Tianrui and P. Yi, "A comparison of parallel large-scale knowledge acquisition using rough set theory on different MapReduce runtime systems," *International Journal of Approximate Reasoning,* vol. 55, no. 3, pp. 896-907, 2014.

[10] J. G. Shanahan and D. Laing, "Large scale distributed data science using apache spark," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015.

[11] X. Lu, I. Nusrat S., W.-U.-R. Md, J. Jithin, S. Hari, W. Hao and P. Dhabaleswar K., "High-performance design of Hadoop RPC with RDMA over InfiniBand," in *In 2013 42nd International Conference on Parallel Processing*, 2013.

[12] J. G. Son, K. Ji-Woo, A. Jae-Hoon, A. Hyung-Joo, C. Hyo-Jung and K. Jung-Guk, "Parallel Job Processing Technique for Real-time Big-Data Processing Framework," in *International Conference on Research in Adaptive and Convergent Systems*, 2016.

[13] K. M. M. H. K. Wang, N. Nhan and G. Swapna, "A Model Driven Approach Towards Improving the Performance of Apache Spark Applications.," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019.

[14] N. Chaimov, M. Allen, C. Shane, I. Costin, Z. I. Khaled and S. Jay, "Scaling spark on hpc systems," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, 2016.

[15] A. J. Awan, B. Mats, V. Vladimir and A. Eduard, "Performance characterization of in-memory data analytics on a modern cloud server," in *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, 2015.

[16] S. Michael, A. Thota and R. Henschel, "Hpchadoop: A framework to run hadoop on cray x-series supercomputers," in *Cray USer Group (CUG)*, 2014.

[17] N. J. Gunther, P. Puglia and K. Tomasette, "Hadoop superlinear scalability," *Commun. ACM,* vol. 58, no. 4, pp. 46-55, 2015.

[18] R. C. Green, L. Wang and M. & Alam, "Applications and trends of high performance computing for electric power systems: Focusing on smart grid," *IEEE Transactions on Smart Grid,* vol. 4, no. 2, pp. 922-931, 2013.

[19] O. Marcu, A. Costan, G. Antoniu and M. Pérez-Hernández, "Spark versus flink: Understanding performance in big data analytics frameworks," in *IEEE International Conference on Cluster Computing (CLUSTER)*, 2016.

[20] "Classification and regression," Apache Spark 2.4.4, [Online]. Available: https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-regression.

[21] B. Mathiya, "Apache hadoop yarn parameter configuration challenges and optimization," in *V. D.-2015 I. C. on Soft, and undefined 2015, ieeexplore.ieee.org.*.

[22] B. a. K. A. a. Z. M. a. G. A. a. J. A. D. a. K. R. H. a. S. S. a. S. I. Hindman, "Mesos: A platform for fine-grained resource sharing in the data center.," in *NSDI*, 2011, pp. 22--22, Volume 11.