ISSN No. 0976-5697

Volume 8, No. 1, Jan-Feb 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Algorithm to Solve Dirichlet Problem for Laplace's Equation

Dr. Mohammad Miyan
Associate Professor, Shia P. G. College, University of Lucknow
Sitapur Road, Lucknow (UP) India

Abstract:- The various researches and developments in the parallel computing Apache Spark framework allows to process petabyte-scale data and possesses properties such as scalability, fault tolerance, load balancing and mechanisms of in the memory computations across the nodes of the cluster. So, the features are much attractive for high performance of scientific computations. As the Hadoop platform is not much suitable for the iterative computing due to some typicality then Apache Spark with new distributed data structure (RDD) is much suitable. Here we are using the method and algorithm described by researchers from time to time for Hadoop-based algorithm to solve the Dirichlet problem for Laplace's equation. The comparative figures are drawn with respect to time to verify the performance. By seeing graphs and other details, we can say that the Spark based implementation is much suitable for solving the Dirichlet problem for their improved performance as compared to Hadoop-based implementation.

Keywords:- Apache Spark; Dirichlets problem; Hadoop; Laplace's equation; RDD.

I. INTRODUCTION

Most of the industries are using Hadoop broadly to analyze their data sets. The Hadoop framework, is generally based on the simple programming model normally i.e., MapReduce model and it empowers a computing solution that is flexible, scalable, cost effective and fault-tolerant. The important concern is to maintain the speed in the processing of the large datasets with respect to waiting time between queries and waiting time to run the used program. The Spark was initiated by Apache Software Foundation for speed up the Hadoop computing software process. The Apache Spark is the lightning-fast cluster based computing technology, designed for the fast computation. It is based on Hadoop MapReduce and it expands the MapReduce model to use it efficiently for the various computations that include suitable queries and the better processing. The main feature of the Spark is it's in memory cluster computation technique that increases the processing speed of the application. The Spark is designed for covering a big range of workloads like as batch applications, iterative algorithms, interactive streaming and queries. Including all these workload in the respective system, it also reduces the management problem of maintaining the separate tools.

The Spark is one of the Hadoop's sub-project established in UC Berkeley's (2009) AMP Lab by Matei Zaharia [1]. It was open under the BSD license (2010). After that it was given to a company Apache Software Foundation (2013) and then is known as Apache Spark and gets a top level (2014) [2].

The some important features of the Apache Spark are as follows [3]:

- Spark provides built in APIs in various programming languages like Scala, Java or Python. So, we can write the applications in various languages. The Spark comes up with eighty high level operators for the interactive and suitable querying.
- Spark helps to run a program in the Hadoop cluster up to hundred times faster in the memory and ten times faster when running on the disk. This is possible by reducing the number of read or write instructions to the disk only. It stores the intermediate processing data in the memory.
- The Spark also supports 'Map' and 'reduce', Streaming data, SQL queries, Machine learning and Graphical algorithms.

It is the better framework for performing the suitable data analysis on distributed computing cluster, like Hadoop. It provides in memory the computations for increasing speed and data processing over the MapReduce. It runs on top of the existing Hadoop cluster and access Hadoop data store i.e., HDFS, can also process the structured data in Hive and Streaming data from HDFS, Kafka, Flume, Twitter etc. Hadoop is the parallel data processing framework that has generally been used to run MapReduce jobs. These are normally long running jobs that take much time to complete. The Spark has designed to run on top of the Hadoop and it is the alternative to traditional batch MapReduce model which can be used for real time data processing and fast useful queries that complete within few moments. Therefore, Hadoop supports both traditional MapReduce and also Spark. The Spark stores data in memory i.e., better for speed in comparison to Hadoop, which stores the data on the disk. The Hadoop uses reproduction to get fault patience whereas Spark uses the different data storage model, flexible distributed datasets i.e., RDD, uses a suitable method of guaranteeing fault tolerance that minimizes the network I/O. The Hadoop is one of the ways to implement the Spark. The Spark uses Hadoop in two ways i.e., one is storage and second is processing. Since the Spark has its own cluster management computation so, it uses the Hadoop for storage purpose only [4]. The Hadoop ecosystem is shown in the figure-1 [5]. The process of executing a job by Spark is

demonstrated by figure-2 [6]. The Spark RDD is shown by the figure-3 [7] and the RDD-transformations and actions are shown by the figure-4 [8].

The Spark's major use [9] cases over the Hadoop are as follows:

- The suitable, interactive Data Mining and the Data Processing.
- The iterative Algorithms in the Machine Learning.
- The Stream processing i.e., Log processing and Fraud detection in the live streams for alerts, aggregates and analysis.
- The Spark has totally Apache Hive-compatible data warehousing structure that can run about hundred times faster than Hive.
- The sensor data processing i.e., where the data is fetched and joined from the different sources, in memory dataset really helpful as they are much easy and fast for processing.

The ways of Spark deployment are as follows:

- *Spark Standalone* deployment means Spark takes the position on the top of Hadoop Distributed File System and space is allocated for the HDFS. In this case, the MapReduce and Spark will run parallel to do the complete jobs.
- *Hadoop Yarn* deployment means, spark runs on Yarn without any pre installation or root access. It also helps to integrate Spark into Hadoop ecosystem. It is shown by figure-1. It also instructs the other sections to run on the top of the stack.
- *Spark in MapReduce* is normally used for starting the spark job with the standalone equipment. With SIMR, user can begin the Spark and uses its shell without any administrative approach.

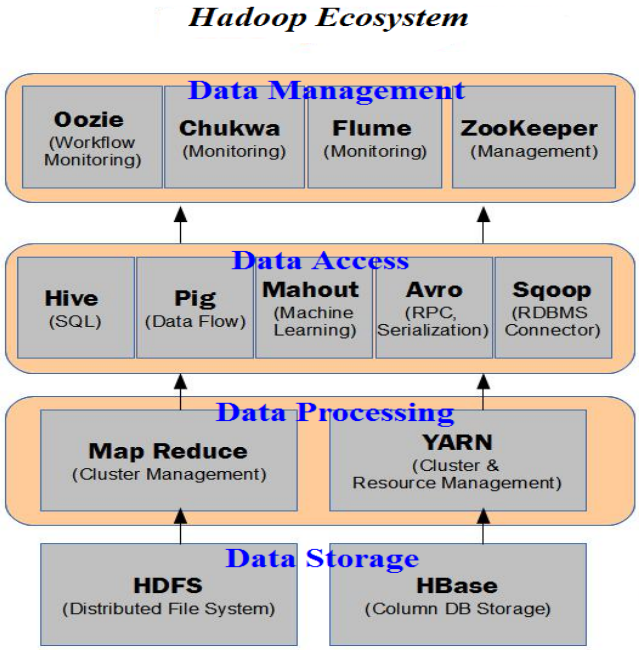

Figure 1 (Hadoop Echosystem)

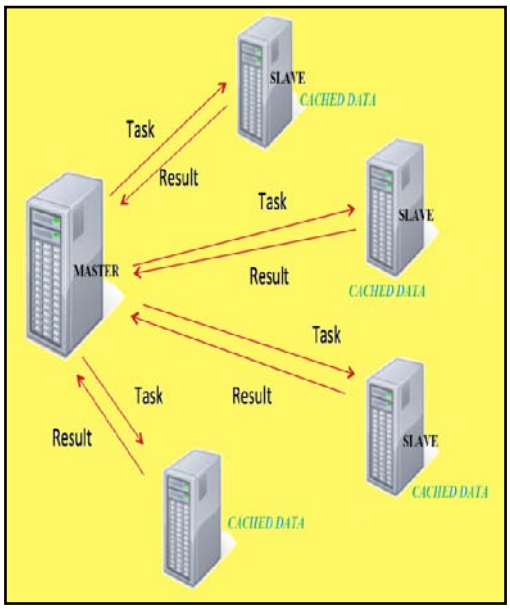

Figure 2 (Process of Executing a job by Spark)

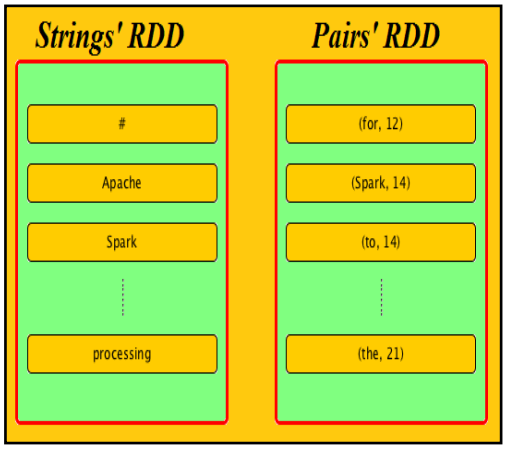

Figure 3 (Spark RDDs)

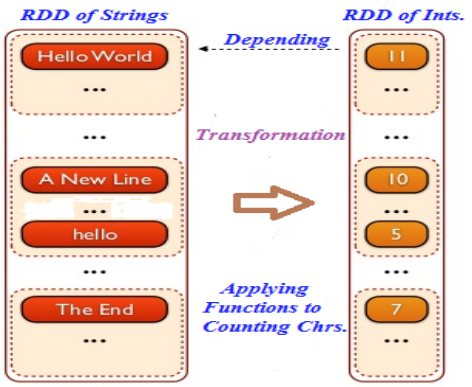

Figure 4 (RDD Transformations and Actions)

II. RELATED RESEARCHES

Freeman, (2014) [11]; Horlacher et al., (2014) [12]; Zhao et al., (2015) [13] have discussed about the large scope of problems in different areas of science that have successfully solved by using Apache Spark. The Apache Spark structure was initially comes and used due to the low performance of the machine learning tools for the large scale data

processing within time. The Spark gives a wide range of methods to handle the various types of problems. There are not much alternatives to choose from to replace or augment MPI computational paradigm for the big scale scientific problems with iterative schemes and research is ongoing with varied success in this useful field. The vast advantage of MPI over the Apache Spark is that MPI potentially can be broadened to the wide range of applications in HPC and still be much fast. So, the most important issue with MPI is due to its lack of built-in failure resistance. The failures can be problematic for long running works in setting of the huge number of computing nodes. There are various techniques and methods to avoid the failures in MPI structure but they are unvaried and vulnerable to implement.

Hans Johansen et al., (1998) [14] have presented a numerical based process to solve the Laplace's equation with the variable coefficients with the Dirichlet boundary conditions, on the two dimensional cases. His work suggested a new way of approaching discretizations for free or fixed boundary problems in that the boundary can be shown by using a volume of the fluid description.

The treatment of these methods is discussed by Gropp et al., (2004) [15]. The Apache Spark design can be treated as a generalization of MapReduce programming paradigm in the reference of the distributed programming models. MapReduce can be seen as the series of parallel map tasks followed by the series of parallel reduce tasks. The map is to derive key pairs from raw input according to some criteria functionally. The reduction takes the list of values with the specified key as an input and outputs different set of key pairs generated by the input list. The Spark offers aside from map and reduces the several earlier discussed operations and in general abstracts away these operations into the transformation concept. Large works are devoted to improve the speed of running MapReduce based programs.

Lu et al., (2011) [16] have describe the hybrid framework of using MPI as the pipeline to exchange an intermediate data between concurrently running reduce and map processes. The resulting solution outperforms some of the MPI-Mapreduce or Hadoop implementations on the applications i.e., Distributed Inverted Indexing, Word Count and Distributed Approximate Similarity Search.

Matei Zaharia et al., (2012) [2], have described the RDD internal design and properties. They also demonstrated its ability to do in-memory computations on the large clusters in the fault-tolerant way. They have discussed the large speed-up on iterative graph and machine learning algorithms with the help of Apache Spark over PGAS and Hadoop. On considering the conceptual differences of global-memory access languages like as PGAS and various parallel programming languages with different memory abstractions there is the trade-off between maintaining granularity of the elements in the memory and also doing large number of operations on these elements. The main advantage over the PGAS model is that RDD operations are coarse-grained so that reducing overhead of storing the states of every element in a distributed atmosphere. They have presented the resilient distributed datasets (RDDs), general-purpose, efficient and fault-tolerant abstraction for sharing the data in the cluster applications. The RDDs can instruct a huge range of parallel applications and techniques that also includes many specialized programming structures that have been proposed for the iterative computation and other new applications that these structures do not take up. B. Kumalakov et al., (2012) [18] studied of adapting the scientific computing problems to the cloud environment, like the Map Reduce. Presented research introduces novel iterative processing framework for Hadoop.

Lu et al., (2014) and Lu and Liang, (2016) [19], [20] have given a comparative better performance and communication library based on the MPI communication structures called Data MPI. As the result of which, that showed the using Data MPI communication primitive's, everyone can achieve performance gain of around 32% as compared to the Hadoop communication primitives. Authors also generalize communication patterns into 4D bipartite communication model and key value communication model, which fits into the requirements of Hadoop-like system specifications and could potentially lead to better design of communication sub-systems in Big Data frameworks.

Reyes-Ortiz et al., (2015) [21] have compared Apache Spark performance with Open MP /MPI based on KNN and Pegasos SVM machine learning algorithms. The results showed that open MP / MPI method is comparatively more than ten times faster with respect to running time; however, we can note that the Spark has also a great advantage of caching.

Li et al., (2016) [22] have described several avenues to improve in Hadoop MapReduce framework as follows:

- Developing of more efficient job scheduling mechanism which takes into account non-homogeneous distribution of the resources in the distributed system
- Improving of programming model to developing advanced iterative processing routines which would allow more efficient job execution
- Extending of the capabilities of system by allowing the parallel execution of map and reduce tasks
- Developing of more convenient real time processing by improving streaming functionality

Apache Spark is believed to show much better performance according to first and fourth items given above. Apache Spark is based on RDD distributed data structure storage.

Shomanov Aday et al., (2016) [9] have described an algorithm to solve Dirichlet problem for Poisson's equation is described, analyzed and compared to optimized Hadoop based implementations. Apache Spark uses a distributed data structure called RDD. The algorithm given by them consists of operations on RDD such as grouping, mapping and partitioning. The various drawbacks and benefits of the mathematical algorithm and also of applicability for tiny type computations are analyzed and discussed.

III. PARALLEL ALGORITHM

The exact analytical solutions for Dirichlet problem is only limited by the specific cases in appropriate domains therefore in major situations, the numerical approaches to find solution to the problem is applied. Let us consider the 3D model of Dirichlet problem for the Laplace's equation [9], [10] in a domain D of hypercube, where D is as defined:

$D= \{(x_i): 0 \leq x_i \leq d_i\}; \forall\ i$

$$\sum_{i=1}^{3} \frac{\partial^2 v}{\partial {x_i}^2} = F(x_i) \qquad (3.1)$$

In the domain, the number of points is P_1, P_2 and P_3 with the co-ordinates x_i. Then the result will be computational mesh with

$$\Delta x_i = \frac{D_i}{P_i}, \forall\, i \qquad (3.2)$$

The second order derivatives with approximation gives:

$$\frac{\partial^2 v}{\partial {x_1}^2} \approx \frac{v_{\alpha+1,\beta,\gamma} + v_{\alpha-1,\beta,\gamma} - 2v_{\alpha,\beta,\gamma}}{\Delta {x_1}^2} \qquad (3.3)$$

$$\frac{\partial^2 v}{\partial {x_2}^2} \approx \frac{v_{\alpha,\beta+1,\gamma} + v_{\alpha,\beta-1,\gamma} - 2v_{\alpha,\beta,\gamma}}{\Delta {x_2}^2} \qquad (3.4)$$

$$\frac{\partial^2 v}{\partial {x_3}^2} \approx \frac{v_{\alpha,\beta,\gamma+1} + v_{\alpha,\beta,\gamma-1} - 2v_{\alpha,\beta,\gamma}}{\Delta {x_3}^2} \qquad (3.5)$$

From the above equations, we have

$$v_{\alpha,\beta,\gamma}^{n+1} = \frac{\left[\begin{array}{c} \frac{v^n_{\alpha+1,\beta,\gamma} + v^n_{\alpha-1,\beta,\gamma}}{\Delta {x_1}^2} + \frac{v^n_{\alpha,\beta+1,\gamma} + v^n_{\alpha,\beta-1,\gamma}}{\Delta {x_2}^2} + \\ \frac{v^n_{\alpha,\beta,\gamma+1} + v^n_{\alpha,\beta,\gamma-1}}{\Delta {x_3}^2} - F_{\alpha,\beta,\gamma} \end{array}\right]}{\sum_i \frac{2}{\Delta {x_i}^2}} \qquad (3.6)$$

Here $v_{\alpha,\beta,\gamma}, F_{\alpha,\beta,\gamma}$ are the values of *v* and *F* in *(α, β, γ)* node of computational mesh. Now we are using the method and algorithm described by researchers (B. Kumalakov, 2013; Mansurova *et al*., 2014; Shomanov Aday et al., 2016) to Hadoop-based algorithm for solving Dirichlet problem for Laplace's equation.

The Hadoop based implementation to solve the Dirichlet problem has the following steps:

Step1. The Map phase algorithm maps each point in the computational domain to certain reducer that will perform the computations in its own sub domain.

Step2. In the Reducer phase every point is mapped inside the 3-dimensional array to perform stencil computations according to Laplace's equation difference scheme. The new internal points after computation in the current iteration are stored to a suitable place to avoid redistribution of them across the nodes of the cluster in the process of the next iteration and the new boundary points are reduced and written to the suitable output HDFS directory.

Step3. If the iteration count is reached the limit program will terminate, otherwise the algorithm will continue with step 1.

Apache Spark algorithm for solving Dirichlet problem for Laplace's equation uses object based representation of points in the computational domain. Point class is used for storing single point in computational mesh. Every point consists of the following properties: Integer partition_id, integer number *a* for x_1 coordinate, integer number *b* for x_2 coordinate, integer number *c* for x_3 coordinate, floating-point number value for value in specific point (x_1, x_2, x_3) of the discrete mesh. Partition property is responsible for storing partition number of the point that is sub-domain identifier. Algorithm consists of several steps each step performs certain operations on RDDs.

The parallel algorithm to solve the Dirichlet problem for Laplace's equation with the use of MapReduce consists of the three parts i.e., initializing, iterating and controlling parts.

The initialization part is executed before the iteration part. The main function of it is to initialize all points with respect to the problem conditions.

The Iteration section of the algorithm [10] is as follows:

Map function

Input*: in_key subcube number, in_value is a string like (a_b_c Value);*

Output: *(out_key=in_key, out_value=in_value)*

I_1: collect (out_key, out_value)

I_2: End;

Reduce function

Input: (in_key, in_value), in_key is a string that represents the number of part of the cube; in_value is a string like (a_b_c Value).

Output: (out_key, out_value), out_key is a string represents the neighbor-subcubes" of current one, out_value is the string represents new values of boundary plains' points.

I_1: *Define 2-multidimensional matrices for the data*
I_2: *Initialize one of them with data received from mapper*
I_3: for *a = 0, . . . , number of Plains* ***do***
I_4: ***for*** *b = 0, . . . , number of Rows* ***do***
I_5: ***for*** *c = 0, . . . , number of PointsInRow* ***do***
I_6: ***if*** *conditions excluding "ghost plains"* ***then***
I_7: ***Fa*** *= (data[a + 1][b][c] + data[c- 1][b][c]) / owa;*
I_8: ***Fb*** *= (data[a][b + 1][c] + data[a][b - 1][c]) / owb;*
I_9: ***Fc*** *= (data[a][b][c + 1] + data[x][b][c - 1]) / owc;*
I_{10}: *newData[a][b][c]=*
*(Fa + Fb + Fc - Ro((a+dta)*ha, b* hb, c*hc))/c;*
I_{11}: *Writing inner updated values to hdfsfiles[]*
I_{12}: *Out of boundary values[]*
I_{13}: *Output (out_key, out_value)*
I_{14}: *End;*

So to perform the exchange operation we want to apply mapping, grouping and partitioning transformations in all the iteration of the algorithm. The mapping transformation will map point elements into tuple of the point and partition number. The partition number is identified by the point coordinates. After that the points are grouped together by applying group. By key transformation and finally every partition is separated by applying partition into locations. Such operations slightly degrade the performance of the program leading to moderate speed-up. The major distinction between Spark implementation and Hadoop based implementation is that the Spark solution uses in-memory computations and thus is suited better for iteration based tasks like as Dirichlet problem [10].

IV. EXPERIMENTAL ANALYSIS

The cluster setup consisted of the following hardware and software settings: quadcore i_3 Intel processor PC each equipped with 8 GB memory cards, Intel Xeon Processors and 4 Gigabit Ethernet switch for the network connection and equipped with Hadoop and Spark software. After testing the program on cluster environment Spark implementation

has been compared to Hadoop combined with MPI and Hadoop implementations as described by (B. Kumalakov, 2013; Mansurova *et al*., 2014; Shomanov Aday et al., 2016) [9], [10], [23] in terms of running time for different sizes of computational domain and different number of cores and cluster nodes. The computational domain represents a cube with fixed and same number of discrete points along every dimension. The results of computation were verified for correctness compared to the sequential code for the same problem with the same settings of the domain size and initial conditions.

Now increasing the number of iterations results in the performance improvement of the Spark implementation as compared to Hadoop-based implementation. The reason for that performance difference is that the Spark performs more operations it perform most of the operations in the memory to leads the higher performance in case of big number of iterations. Whereas the Hadoop performs better on single iteration cycle, but at the end of the every iteration it writes data back to HDFS or to local file system which leads to poor performance due to the accumulation of input output latencies over the course of computation. The test runs of the software measured system performance with the hypercubes of input sizes: 128 x 128 x 128, 256 x 256 x 256 and 512 x 512 x 512. The graphs of the various ratios are as follows:

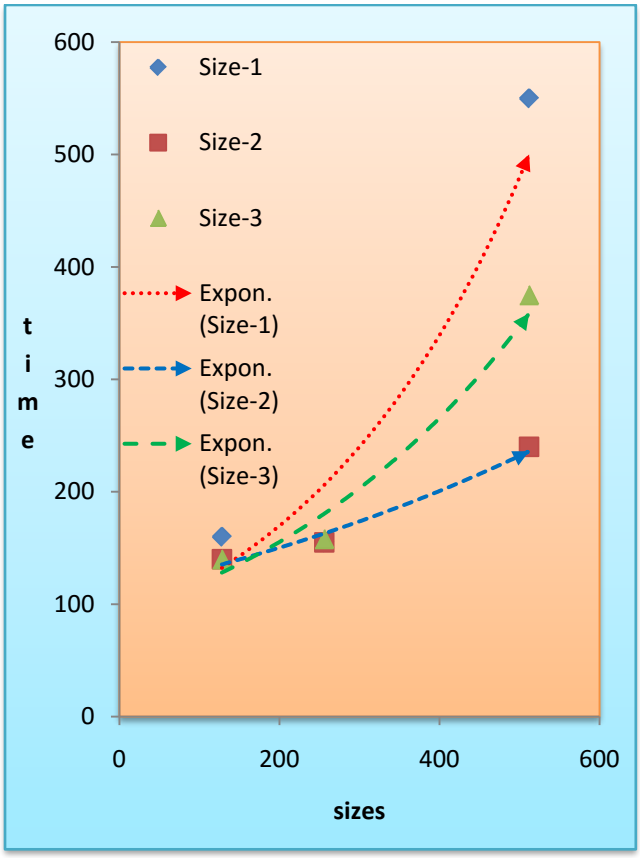

Figure 5. (The execution time in the three cube sizes and the number of nodes)

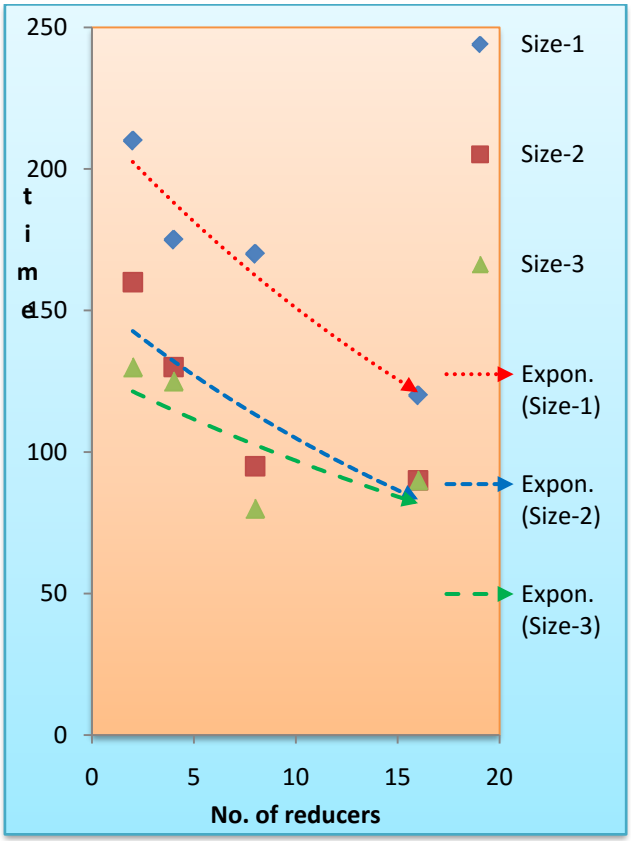

Figure 6. (The execution time in different reducers and number of nodes)

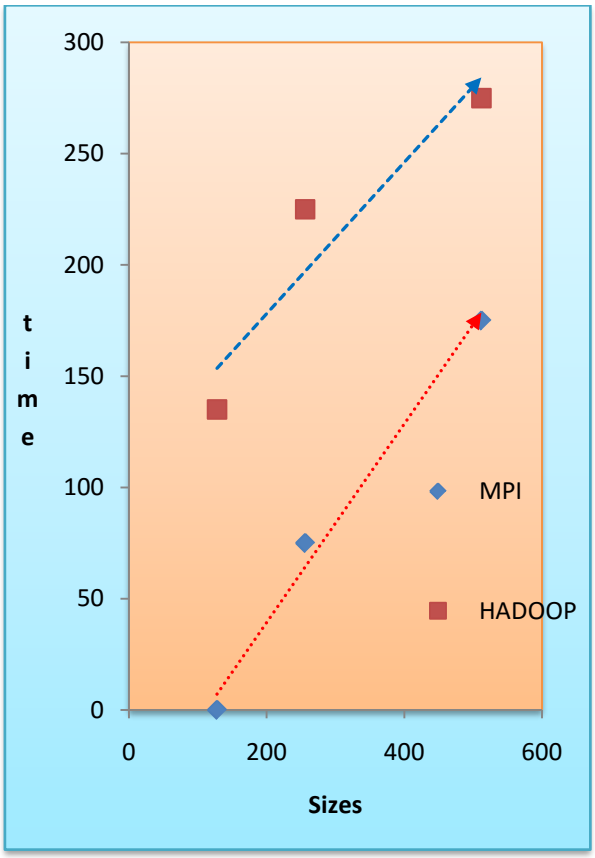

Figure 7. (The comparison of the execution times in MPI and Hadoop platform)

The graphical representation by figure-5 represents ratios between the data numbers and used time to complete an

execution. By increasing the number of nodes the speed of processing decreases but for the small data there is no benefit. The figure 6 represents ratios between the number of reducers and time taken to complete the test execution with cube (256 x 256 x 256). By increasing the reducer's number, it decreases the time of execution. The figure 7 represents the comparison between the calculations on MPI and Hadoop respectively.

V. CONCLUSIONS

By using the algorithm and the computational analysis for solving the Dirichlet problem for Laplace's equation and by using Apache Spark framework and compared it with Hadoop-based implementations. By analyzing the graphical analysis and other details, we can say that the Spark based implementation is much suitable for solving the Dirichlet problem for their improved performance as compared to Hadoop-based implementation.

VI. REFERENCES

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, "Spark: Cluster computing with working sets", Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, Jun 2010, 22-25, ACM, USA, pp: 10-10.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave and J. Ma *et al*., "Resilient distributed datasets: A faulttolerant abstraction for in-memory cluster computing", Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, Apr. 2012, 25-27, ACM, USA., pp: 2-2.

[3] Spark SQL Tutorial Notes by Tutorials Point (I) Pvt. Ltd., 2015, pp. 1-29. https://www.tutorialspoint.com/spark_sql/spark_sql_tutorial.pdf.

[4] Apache Spark Tutorial Point, 2017. https://www.tutorialspoint.com/apache_spark/apache_spark_introduction.htm

[5] Kornkorneliusz, "Hadoop Ecosystem and Big Data", May 2014. https://blog.udemy.com/hadoop-ecosystem/

[6] A. Kuntamukkala (Software Architect), "Apache Spark: An Engine for Large-Scale Data Processing", SciSpike. https://dzone.com/refcardz/apache-spark

[7] RDD-Resilient Distributed Dataset Analysis by GitBooks. https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-rdd.html

[8] F. R. Olivera, Apache Spark Java Conference, Buenos Aires, Argentina, Nov. 2014. http://www.slideshare.net/frodriguezolivera/apache-spark-41601032

[9] S. Aday and M. Madina, "Novel Apache Spark based Algorithm to Solve Dirichlet Problem for Poisson Equation in 3D Computational Domain", Journal of Computer Sciences, 2016, 12 (10): 502-509. DOI: 10.3844/jcssp.2016.502.509

[10] B. Kumalakov, A. Shomanov, Ye. Dadykina, S. Ikhsanov and B. Tulepbergenov, "Solving Dirichlet Problem for Poisson's Equation Using MapReduce Hadoop", Poceed. Of Conf., Kazakhistan, 2013, pp. 136-142. http://taac.org.ua/files/a2013/proceedings/KZ-2-Bolatzhan%20Kumalakov-327.pdf

[11] J. Freeman, "Mapping brain activity at scale with cluster computing", Nat. Meth., 11: 941-950, 2014. DOI: 10.1038/nmeth.3041.

[12] O. Horlacher, , F. Lisacek and M. Müller,. Mining large scale tandem mass spectrometry data for protein modifications using spectral libraries. J. Proteome Res., 2014, 15: pp. 721-731. DOI: 10.1021/acs.jproteome.5b00877

[13] G. Zhao, C. Ling and D. Sun, "Spark SW: Scalable distributed computing system for large-scale biological sequence alignment", Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 2015, 4-7, IEEE Xplore Press, pp: 845-852. DOI: 10.1109/CCGrid.2015.55.

[14] H. Johansen, "Applied Numerical Algorithms Group", Computational Research , Barkley Lab, US Deptt. of Energy, 1998. http://crd.lbl.gov/departments/applied-mathematics/ANAG/about/staff-and-postdocs/hans-johansen/

[15] W. Gropp and E. Lusk, "Fault tolerance in message passing interface programs", Int. J. High Performance Comput. Applic., 2004, 18: 363-372. DOI: 10.1177/1094342004046045.

[16] X. Lu, and F. Liang, "Accelerating iterative big data computing through MPI", Int. J. Comput. Sci. Technol., 2016, 30: 283-294. DOI: 10.1007/s11390-015-1522-5.

[17] Solving Critical Simulation Problems Under Emergency ConditionsUsing Volunteer Computing

[18] B. Kumalakov, D. A. Zaki and G. Dobrowolski, 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, SIMULTECH 2014 pp. 170-178.

[19] X. Lu, B. Wang, L. Zha and Z. Xu, "Can MPI benefit Hadoop and MapReduce applications?", Proceedings of 40th International Conference on Parallel Processing Workshops, Sept. 2011, 13-16, IEEE Xplore Press, pp: 371-379. DOI: 10.1109/ICPPW.2011.56.

[20] X. Lu, , F. Liang, B. Wang, L. Zha and Z. Xu, "DataMPI: Extending MPI to hadoop-like big data computing", Proceedings of the IEEE 28th International Parallel Distributed Processing Symposium, May 2014, 19-23, IEEE Xplore Press, pp: 829-838. DOI: 10.1109/IPDPS.2014.90.

[21] J. L. Reyes-Ortiz, L. Oneto and D. Anguita, "Big data analytics in the cloud: Spark on hadoop Vs MPI/OpenMP on Beowulf", Proc. Comput. Sci., 2015, 53: 121-130. DOI: 10.1016/j.procs.2015.07.286.

[22] R. Li, H. Hu, H. Li, Y. Wu and J. Yang, "MapReduce parallel programming model: A state of-the-art survey", Int. J. Parallel Programm., 2016, 44: 832-866. DOI: 10.1007/s10766-015-0395-0.

[23] M. Mansurova, D. Ahmed-Zaki, A. Shomanov, Y. Dadykina and S. Ikhsanov *et al*., "Solving dirichlet problem for poissons equation using mapreduce hadoop and MPI", Proceedings of International Conference New Trends in Information and Communication Technologies, (ICT' 14), 2014, pp: 226-234.