Verification and Validation of MapReduce Program Model for Parallel Support Vector Machine Algorithm on Hadoop Cluster

Kiran M.[1], Amresh Kumar[2], Saikat Mukherjee[3] and Ravi Prakash G.[4]

**[1] Department of Computer Science and Engineering, Christ University Faculty of Engineering
Bangalore, Karnataka, India**

**[2] Department of Computer Science and Engineering, Christ University Faculty of Engineering
Bangalore, Karnataka, India**

**[3] Department of Computer Science and Engineering, Christ University Faculty of Engineering
Bangalore, Karnataka, India**

**[4] Department of Computer Science and Engineering, Christ University Faculty of Engineering
Bangalore, Karnataka, India**

Abstract
We currently live in the data age. It's not easy to measure the total volume of structured and unstructured data that require machine-based systems and technologies in order to be fully analyzed. Efficient implementation techniques are the key to meeting the scalability and performance requirements entailed in such scientific data analysis. So for the same in this paper the Sequential Support Vector Machine in WEKA and various MapReduce Programs including Parallel Support Vector Machine on Hadoop cluster is analyzed and thus, in this way Algorithms are Verified and Validated on Hadoop Cluster using the Concept of MapReduce. In this paper, the performance of above applications has been shown with respect to execution time/training time and number of nodes. Experimental Results shows that as the number of nodes increases the execution time decreases. This paper is basically a research study of above MapReduce applications.

Keywords: *Machine Learning, SVM, LIBSVM, WEKA Tool, MultiFileWordCount, PiEstimator, Parallel SVM, Hadoop, MapReduce.*

1. Introduction

The Machine Learning [1] field evolved from the broad field of *Artificial Intelligence*, which aims to mimic intelligent abilities of humans by machines. It is a scientific discipline concerned with the design and development of algorithms that take as input empirical data, such as that from sensors or databases, and yield patterns or predictions thought to be features of the underlying mechanism that generated the data. Machine learning is the body of research related to automated large-scale data analysis. Broadly speaking the main two subfields of machine learning are supervised learning and unsupervised learning. In supervised learning the focus is on accurate prediction (support vector machines, kernels, neural networks), whereas in unsupervised learning the aim is to find compact descriptions of the data (clustering, dimensionality reduction, deep learning).

Apache Hadoop [12] is a software framework that supports data-intensive distributed applications. It enables applications to work with thousands of computational independent computers and petabytes of data.

The Hadoop Distributed File System (HDFS) [8] is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. HDFS is the file system component of Hadoop.

MapReduce [6] is a distributed data processing or programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. MapReduce programs are written in a particular style influenced by *functional programming* constructs, specifically idioms for processing lists of data. This module explains the nature of this programming model and how it can be used to write programs which run in the Hadoop environment.

Data Classification, also referred to as pattern recognition, where one attempts to build algorithms capable of automatically constructing methods for distinguishing between different examples, based on their differentiating patterns. A Support Vector Machine (SVM) performs *classification* by constructing an *N*-dimensional hyperplane that optimally separates the data into two categories. This paper covers about Research Clarification: This includes Machine Learning, Supervised Learning

Classification, SVM, LIBSVM, Weka Tool, Hadoop, HDFS, MapReduce and Cloudera - Cloudera's Distribution Including Apache Hadoop version 4 (CDH4). Descriptive Study I: This includes List of Problems. Prescriptive Study: This includes Hadoop Architecture, MapReduce Programming Model and PSVM-MapReduce Algorithm. Descriptive Study II: Experimental Setup and Experimental Result & its Analysis.

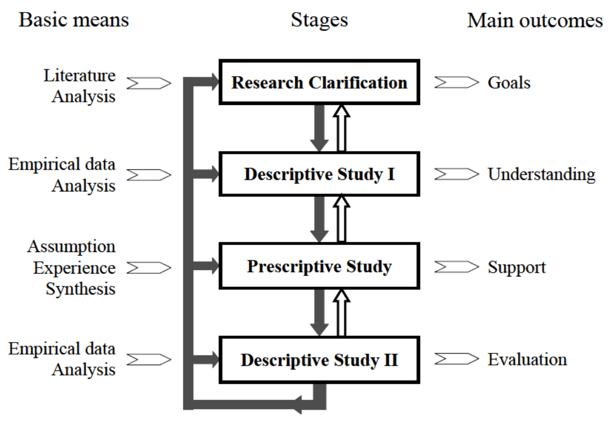


Fig. 1 Research Plan: Basic means, Stages and Main Outcomes.

2. Research Clarification

This section describes about Machine Learning, Types of Machine Learning, Supervised Learning Classification in detail, SVM, LIBSVM, Weka Tool, Hadoop, HDFS, MapReduce and Cloudera – CDH4.

2.1 Machine Learning

A major focus of machine learning research is the design of algorithms that recognize complex patterns and make predictions/intelligent decisions based on input data. A learner can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. Data can be seen as instances of the possible relations between observed variables.

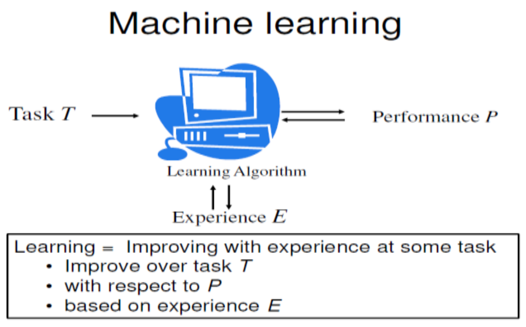


Fig. 2 Machine Learning.

Machine learning focuses on constructing algorithms for making predictions from data. A machine learning task aims to identify (to learn) a function $f : X \rightarrow Y$ that maps input domain X (of data) onto output domain Y (of possible predictions). The function f is selected from a certain function class, which is different for each family of learning algorithms. Elements of X and Y are application-specific representations of data objects and predictions respectively.

2.2 Supervised Learning - Classification

An important task in Machine Learning is *classification*, also referred to as pattern recognition, where one attempts to build algorithms capable of automatically constructing methods for distinguishing between different examples, based on their differentiating patterns. Supervised learning algorithms utilize training data to construct a prediction function *f*, which is subsequently applied to test instances. Typically, training data is provided in the form of labeled examples $(x,y) \varepsilon X \times Y$, where x is a data instance and y is the corresponding ground truth prediction for x.

2.3 Support Vector Machine

A Support Vector Machine (SVM) performs *classification* by constructing an *N*-dimensional hyperplane that optimally separates the data into two categories. In the reference of SVM literature, a predictor variable is called an *attribute*, and a transformed attribute that is used to define the hyperplane is called a *feature*. The task of choosing the most suitable representation is known as *feature selection*. A set of features that describes one case (i.e., a row of predictor values) is called a *vector*.

So the goal of SVM modeling is to find the optimal hyperplane that separates clusters of vector in such a way that cases with one category of the target variable are on one side of the plane and cases with the other category are on the other size of the plane. The vectors near the hyperplane are the *support vectors*.

An SVM analysis finds the line (or, in general, hyperplane) that is oriented so that the margin between the support vectors is maximized. In the figure above, the line in the right panel is superior to the line in the left panel.

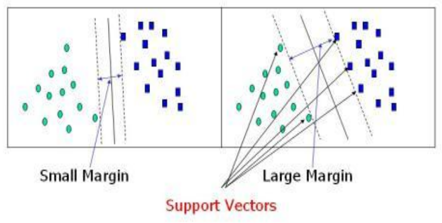


Fig. 3 Margin and Support Vectors.

2.4 LIBSVM

LIBSVM [4] is a library for Support Vector Machines (SVMs). The goal is to help users to easily apply SVM to their applications. LIBSVM has gained wide popularity in machine learning and many other areas. A typical use of LIBSVM involves two steps: first, training a data set to obtain a model and second, using the model to predict information of a testing data set. Many extensions of LIBSVM are available at libsvmtools. SVM formulations supported in LIBSVM are: C-support vector classification (C-SVC), *ν*-support vector classification (*ν*-SVC), distribution estimation (one-class SVM), ε-support vector regression (ε-SVR), and *ν*-support vector regression (*ν*-SVR). LIBSVM implements "one-against-one" multi-class method, because it results in less training time when compared to "one-against-all" multi-class method, so there are *k(k-1)/2* binary models, where *k* is the number of classes.

2.5 WEKA Tool

WEKA (Waikato Environment for Knowledge Analysis) [14]: Open-Source Software Tool is a collection of machine learning algorithms implemented in Java developed at the University of Waikato, New Zealand. WEKA consists of a large number of learning schemes for classification and regression numeric prediction - like decision trees, support vector machines, instance-based classifiers, Bayes decision schemes, neural networks etc. and clustering.

2.6 Hadoop

Apache Hadoop [12] is an open-source software framework that supports data intensive distributed applications, licensed under the Apache v2 license. It enables applications to work with thousands of computational independent computers and petabytes of data. Hadoop was derived from Google's MapReduce and Google File System (GFS) papers. Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project. An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts, and executing application computations in parallel close to their data. A Hadoop cluster scales computation capacity, storage capacity and IO bandwidth by simply adding commodity servers. Hadoop is a top-level Apache project being built and used by a global community of contributors, written in the Java programming language. The Apache Hadoop project and its related sub-projects (Core, Avro, MapReduce, HDFS, Pig, HBase, Zookeeper, Hive and Chukwa) have many contributors from across the ecosystem.

2.7 The Hadoop Distributed File System (HDFS)

HDFS [8] is a distributed, scalable, and portable file system written in Java for the Hadoop framework. It is the file system component of Hadoop. It stores file system metadata and application data separately. As in other distributed file systems, like PVFS, Lustre and GFS, HDFS stores metadata on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes. All servers are fully connected and communicate with each other using TCP-based protocols.

By default, HDFS stores three separate copies of each data block to ensure reliability, availability, and performance. In large clusters, the three replicas are spread across different physical racks, so HDFS is resilient towards two common failure scenarios: individual datanode crashes and failures in networking equipment that bring an entire rack offline. Replicating blocks across physical machines also increases opportunities to co-locate data and processing in the scheduling of MapReduce jobs, since multiple copies yield more opportunities to exploit locality.

2.8 MapReduce

In MapReduce, records are processed in isolation by tasks called *Mappers*. The output from the Mappers is then brought together into a second set of tasks called *Reducers*, where results from different mappers can be merged together.

Problems suitable for processing with MapReduce must usually be easily split into independent subtasks that can be processed in parallel. The map and reduce functions are both specified in terms of data is structured in key-value pairs. The power of MapReduce is from the execution of many map tasks which run in parallel on a data set and these output the processed data in intermediate key-value pairs. Each reduce task only receives and processes data for one particular key at a time and outputs the data it processes as key-value pairs.

The Hadoop MapReduce engine consists of a JobTracker and one or many TaskTrackers. A MapReduce job must be submitted to a job tracker which then splits the job into tasks handled by the task trackers. JobTracker dispatches jobs and assigns splits (splits) to mappers or reducers as each stage completes. TaskTracker executes tasks sent by the JobTracker and reports status to JobTracker.

"Map" step: The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node. "Reduce" step: The master node then collects the answers to all the sub-problems and combines them in some way to form the output – the answer to the problem it was originally trying to solve.

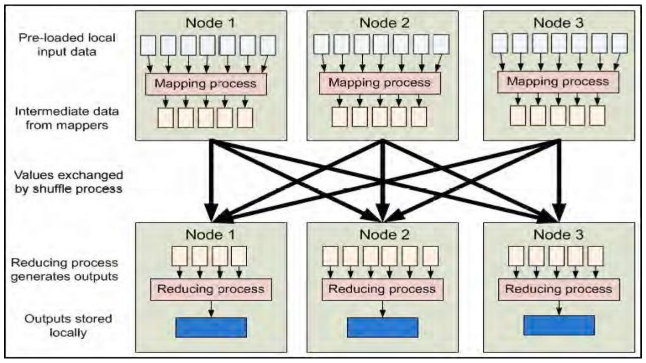

Pre-loaded local input data
Intermediate data from mappers
Values exchanged by shuffle process
Reducing process generates outputs
Outputs stored locally
Node 1
Node 2
Node 3
Mapping process
Reducing process

Fig. 4 MapReduce Design Illustration.

2.9 Cloudera – CDH4

Cloudera Inc. [18] is a software company that provides Apache Hadoop-based software, support and services called CDH. CDH has version of Apache Hadoop patches and updates. It provides how to install and configure version 4 of Cloudera's Distribution Including Apache Hadoop (CDH4) as a Yum, Apt, or zypper/YaST repository. It also describes how to deploy in standalone mode, pseudo-distributed mode, and on a cluster.

CDH4 introduces a new version of MapReduce: MapReduce 2.0 (MRv2) built on the YARN framework. Here, refer to this new version as YARN. CDH4 also provides an implementation of the previous version of MapReduce, now referred to as MRv1.

3. Descriptive Study – I

SVMs suffer from a widely recognized scalability problem in both memory use and computational time.

To improve scalability, a parallel SVM Algorithm is developed, which reduces memory use through parallel computation.

PSVM cannot achieve linear speedup when the number of machines continues to increase beyond a data-size-dependent threshold. This is expected because of Communication & Synchronization Overheads.

Communication Time is incurred when message passing takes place between machines. Synchronization Overhead is incurred when the Master node waits for the task completion on the slowest machine.

By using Hadoop Cluster with the same versions of Software (CentOS 6.2) and same hardware configurations, linear speedup can be achieved.

4. Prescriptive Study

This section includes Hadoop Architecture, MapReduce Programming Model & Structure and flow of PSVM algorithm using MapReduce.

4.1 Hadoop Architecture

Hadoop [16] consists of the *Hadoop Common* which provides access to the file systems supported by Hadoop. The Hadoop Common package contains the necessary JAR files and scripts needed to start Hadoop. The package also provides source code, documentation, and a contribution section which includes projects from the Hadoop Community. Putting everything together, the Architecture of a complete Hadoop cluster is shown in Figure below:

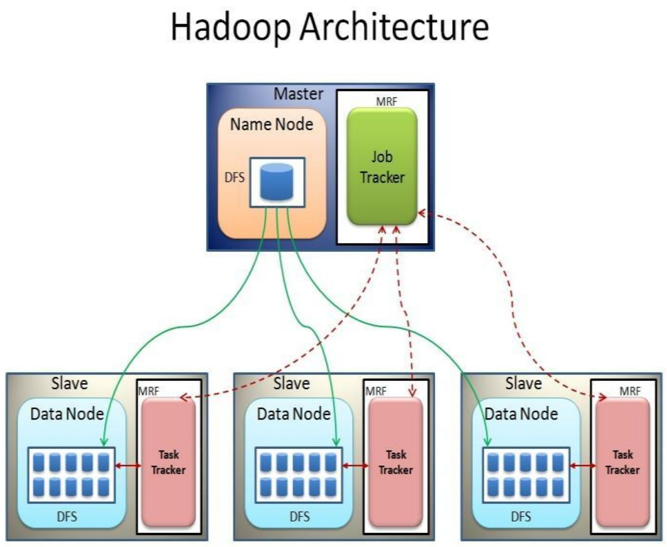


Fig. 5 Hadoop Cluster Architecture.

The HDFS namenode runs the NameNode daemon. The job submission node runs the JobTracker, which is the single point of contact for a client wishing to execute a MapReduce job. The *JobTracker* monitors the progress of running MapReduce jobs and is responsible for coordinating the execution of the mappers and reducers. Typically, these services run on two separate machines, although in smaller clusters they are often co-located. The bulk of a Hadoop cluster consists of slave nodes (only three of which are shown in the figure) that run both a TaskTracker, which is responsible for actually running user code, and a DataNode daemon, for serving HDFS data.

Hadoop MapReduce jobs are divided up into a number of map tasks and reduce tasks. *TaskTrackers* periodically send heartbeat messages to the JobTracker that also doubles as a vehicle for task allocation. If a tasktracker is available to run tasks (in Hadoop parlance, has empty task slots), the return acknowledgment of the tasktracker heartbeat contains task allocation information. The number of reduce tasks is equal to the number of reducers specified by the programmer. The number of map tasks, on the other hand, depends on many factors: the number of mappers specified by the programmer serves as a hint to the execution framework, but the actual number of tasks depends on both the number of input files and the number of HDFS data blocks occupied by those files. Hadoop requires JRE 1.6 or higher. The standard start-up and shutdown scripts require ssh to be set up between nodes in the cluster. In a larger cluster, the HDFS is managed through a dedicated NameNode server to host the file system index, and a secondary NameNode that can generate snapshots of the namenode's memory structures, thus preventing filesystem corruption and reducing loss of data. Similarly, a standalone JobTracker server can manage job scheduling. In clusters where the Hadoop MapReduce engine is deployed against an alternate filesystem, the NameNode, secondary NameNode and DataNode architecture of HDFS is replaced by the file system-specific equivalent.

4.2 MapReduce Programming Model

MapReduce computing model consists of two functions, Map and Reduce. The Map and Reduce functions are both defined with data structure of (key1; value1) pairs. Map function is applied to each item in the input dataset according to the format of the (key1; value1) pairs; each call produces a list (key2; value2). All the pairs which have the same key in the output lists are put to reduce function which generates one (value3) or an empty return. The results of all calls from a list, list (value3).

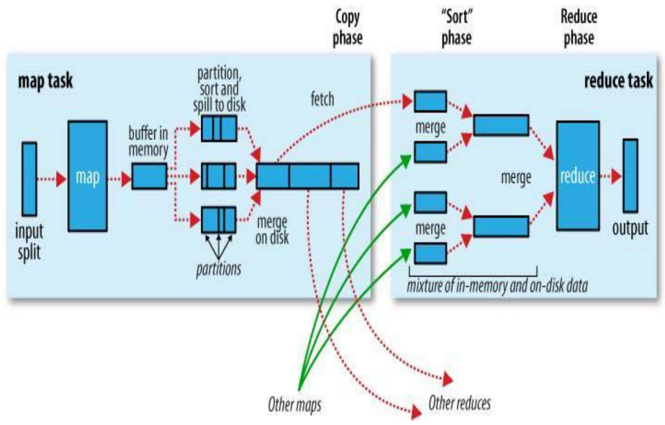


Fig. 6 Process of MAP and REDUCE is illustrated.

4.3 Structure and Flow of PSVM Algorithm using MapReduce

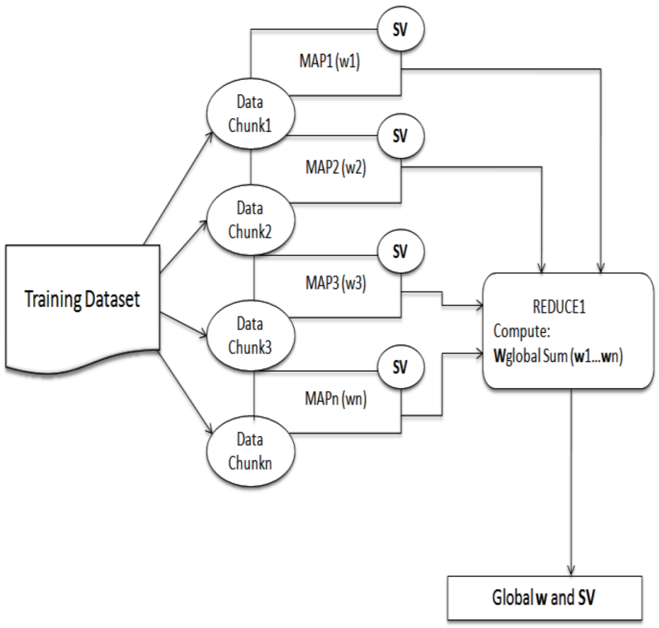


Fig. 7 Flow Diagram of PSVM Algorithm using MapReduce.

Algorithm:

1. Training Dataset: Having Instances, Attributes and Class-Labels are provided by user.

2. Map: In map step, map tasks processes an associated data chunk in its space. The output of each map process is the localized SVM weight vector (wj)

3. Reduce: To compute the global weight vector (Wglobal) by summing the individual maps' weight vectors.

4. Output: Results with a Model having Global W and SV (support vectors).

Description of PSVM Algorithm using MapReduce:

In general, A Map-Reduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*.

Mapper: Each MapReduce *map processes an associated data* chunk in its space. The output of each *map process is the* localized (per data chunk) SVM weight vector (*wj).*

Reducer: Again, the primary role of the associated *reduce phase is to compute the* global weight vector (*wglobal) by summing the* individual *maps' weight vectors.*

5. Descriptive Study – II

This section includes Experimental Setup and Experimental Result & its Analysis.

5.1 Experimental Setup

The experiments were carried out in WEKA and on the Hadoop cluster. The WEKA version used is v3.6, Operating System: 64-bit Windows 7 Ultimate with Intel Core i3-2310M CPU @ 2.10 GHz and 4GB of RAM. The Hadoop infrastructure consists of one cluster having four nodes distributed in one single lab. For the series of experiments, the nodes in the Hadoop cluster, with Intel Core 2 Duo CPU@ 2.53 GHz, 2 CPUs and 2GB of RAM for each node has been used. With a measured bandwidth for end-to-end TCP sockets of 100 MB/s, Operating System: CentOS 6.2 (Final) and SUN JAVA jdk 1.6.0_33.

5.2 Experimental Result and its Analysis

Experiment 01: Sequential SVM using LIBSVM in WEKA

Table 1: Sequential SVM using LIBSVM in WEKA

No. of Instances	Training Time (in sec)
150	57
846	265.75
2310	3024.35
4601	5256.29
10992	12556.35
20000	22846.34

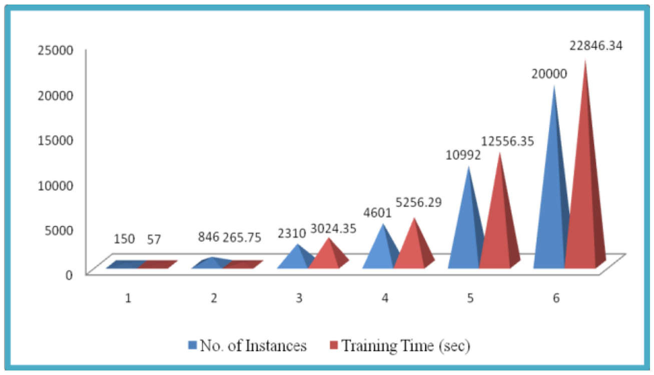


Fig. 8 Results of Sequential SVM using LIBSVM in WEKA.

Experiment 02: MultiFileWordCount

Table 2: MultiFileWordCount – No. of Files Increasing & Nodes Constant

No. of Files	Execution Time (in min)	No. of Nodes
2	3.13	2
3	4.45	2
4	6.9	2

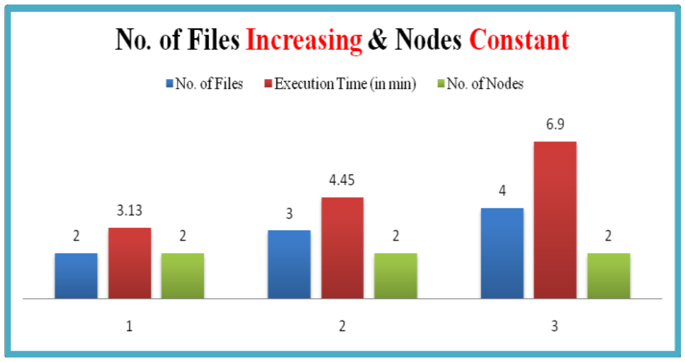


Fig. 9 Results of MultiFileWordCount - No. of Files Increasing & Nodes Constant.

In this experiment, size of 2 Files is 512MB (256MB+256MB), size of 3 Files is 768MB (256MB+256MB+256MB) and size of 4 Files is 1GB (256MB+256MB+256MB+256MB).

Table 3: MultiFileWordCount – No. of Files Constant & Nodes Increasing

No. of Files	Execution Time (in min)	No. of Nodes
4	6.09	2
4	5.18	3
4	4.14	4

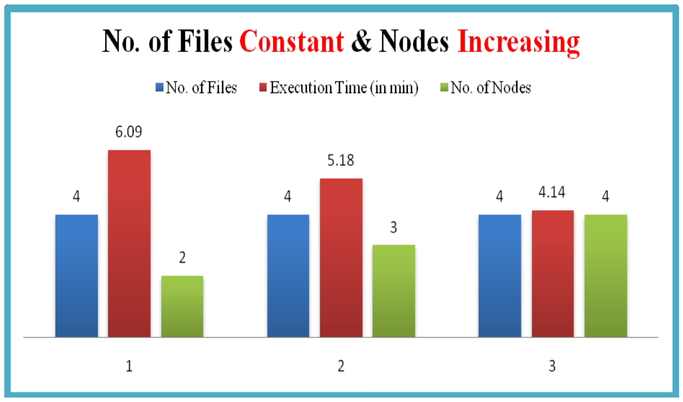


Fig. 10 Results of MultiFileWordCount - No. of Files Constant & Nodes Increasing.

Table 4: MultiFileWordCount – No. of Files Increasing & Nodes Increasing

No. of Files	Execution Time (in min)	No. of Nodes
2	3.13	2
3	4.05	3
4	4.14	4

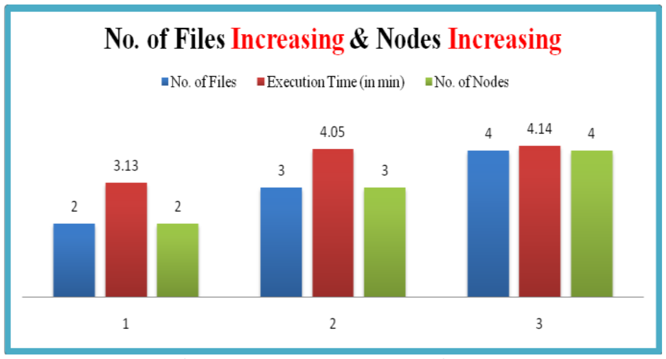
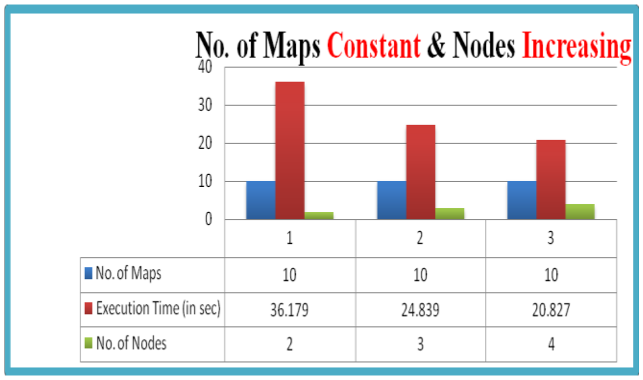


Fig. 11 Results of MultiFileWordCount - No. of Files Increasing & Nodes Increasing.

Experiment 03: PiEstimator

Table 5: PiEstimator – No. of Maps Constant & Nodes Increasing

No. of Maps	Execution Time (in sec)	No. of Nodes
10	36.179	2
10	24.839	3
10	20.827	4

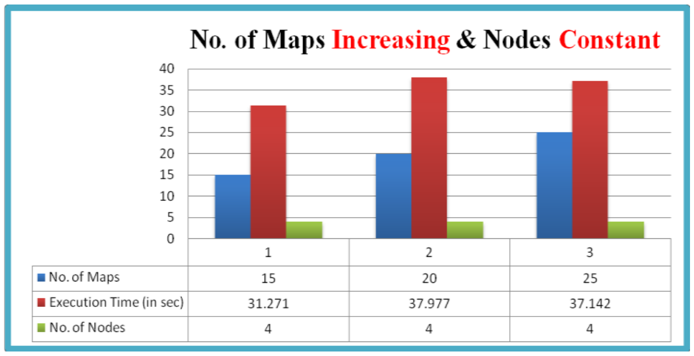


	1	2	3
🟦 No. of Maps	10	10	10
🟥 Execution Time (in sec)	36.179	24.839	20.827
🟩 No. of Nodes	2	3	4

Fig. 12 Results of PiEstimator - No. of Maps Constant & Nodes Increasing.

Table 6: PiEstimator – No. of Maps Increasing & Nodes Constant

No. of Maps	Execution Time (in sec)	No. of Nodes
15	31.271	4
20	37.977	4
25	37.142	4

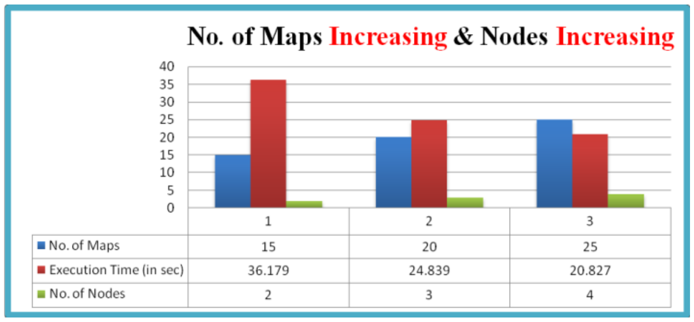


	1	2	3
🟦 No. of Maps	15	20	25
🟥 Execution Time (in sec)	31.271	37.977	37.142
🟩 No. of Nodes	4	4	4

Fig. 13 Results of PiEstimator - No. of Maps Increasing & Nodes Constant.

Table 7: PiEstimator – No. of Maps Increasing & Nodes Increasing

No. of Maps	Execution Time (in sec)	No. of Nodes
15	36.179	2
20	24.839	3
25	20.827	4



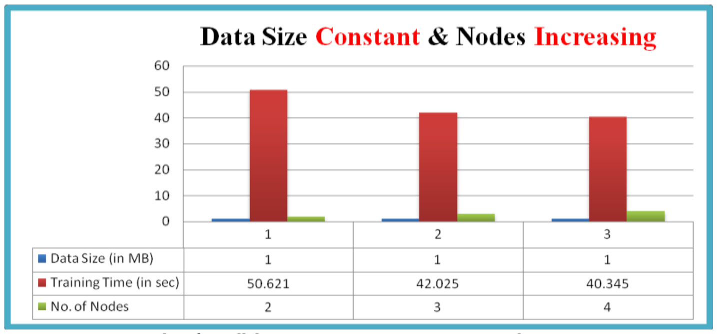
	1	2	3
🟦 No. of Maps	15	20	25
🟥 Execution Time (in sec)	36.179	24.839	20.827
🟩 No. of Nodes	2	3	4

Fig. 14 Results of PiEstimator - No. of Maps Increasing & Nodes Increasing.

Experiment 04: Parallel SVM

Table 8: Parallel SVM – Data Size Constant & Nodes Increasing

Data Size (in MB)	Training Time (in sec)	No. of Nodes
1	50.621	2
1	42.025	3
1	40.345	4

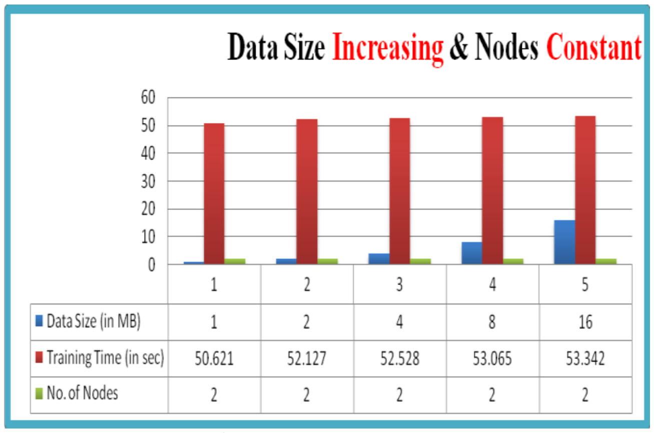


	1	2	3
🟦 Data Size (in MB)	1	1	1
🟥 Training Time (in sec)	50.621	42.025	40.345
🟩 No. of Nodes	2	3	4

Fig. 15 Results of Parallel SVM – Data Size Constant & Nodes Increasing.

Table 9: Parallel SVM – Data Size Increasing & Nodes Constant

Data Size (in MB)	Training Time (in sec)	No. of Nodes
1	50.621	2
2	52.127	2
4	52.528	2
8	53.065	2
16	53.342	2

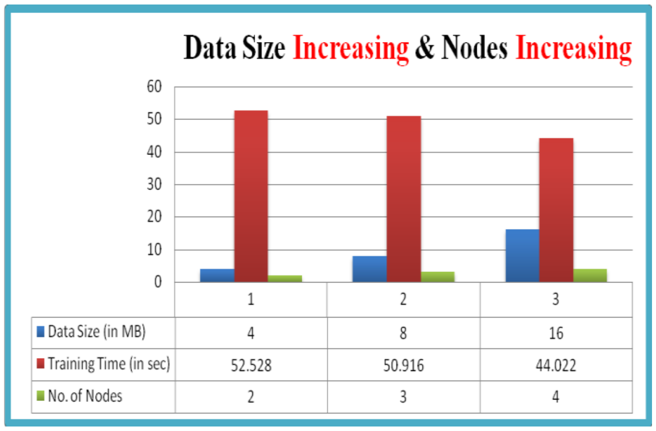


	1	2	3	4	5
🟦 Data Size (in MB)	1	2	4	8	16
🟥 Training Time (in sec)	50.621	52.127	52.528	53.065	53.342
🟩 No. of Nodes	2	2	2	2	2

Fig. 16 Results of Parallel SVM – Data Size Increasing & Nodes Constant.

Table 10: Parallel SVM – Data Size Increasing & Nodes Increasing

Data Size (in MB)	Training Time (in sec)	No. of Nodes
4	52.528	2
8	50.916	3
16	44.022	4



	1	2	3
🟦 Data Size (in MB)	4	8	16
🟥 Training Time (in sec)	52.528	50.916	44.022
🟩 No. of Nodes	2	3	4

Fig. 17 Results of Parallel SVM – Data Size Increasing & Nodes Increasing.

6. Conclusions

SVM classifier depends on the number of support vectors required. In SVM classification, the required memory to store the support vectors is directly proportional to the number of support vectors. Observations and Result analysis show that in Sequential SVM - as the number of instances increases, training time also increases. Also, in the Hadoop Cluster – it has been verified and validated that as the number of nodes increases, with respect to large size of Input Data, execution time decreases. From this, it is shown that Parallel SVM using MapReduce Model performs efficiently. An advantage of using HDFS & MapReduce is the data awareness between the NameNode & DataNode and also between JobTracker & TaskTracker. Till now, in this paper, Hadoop MapReduce has being observed in all i.e. Standalone, Pseudo-distributed and Fully Pseudo-distributed mode. This Hadoop cluster contains four nodes i.e. one Master (NameNode) and three Slaves (DataNode).

In the future work, Scaling up the Hadoop Cluster-having Client and Secondary NameNode [8]. Further study and research of various concepts related with hadoop. Ex: - hadoop – streaming [12]. Performance Evaluation of Parallel SVM Algorithm by introducing different Kernel Methods. Ex: - vector space kernel [21].

References

[1] Gunnar Ratsch, "A Brief Introduction into Machine Learning", Friedrich Miescher Laboratory of the Max Planck Society, 2004.
[2] Cortes and V. Vapnik. "Support-vector network", Machine Learning, 20:273-297, 1995.
[3] Chih-Wei Hsu and Chih-Jen Lin. A Comparison of Methods for Multi-class Support Vector Machines, 13 (2): 415-425, 2002.
[4] C. C. Chang and C. J. Lin, "LIBSVM: A Library for Support Vector Machines", National Taiwan University, Taipei, Taiwan, 2001.
[5] Chang, E. Y., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z. and Cui, H. Parallelizing "Support Vector Machines on Distributed Computers", 2007.
[6] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.
[7] Mahesh Maurya and Sunita Mahajan. "Performance analysis of MapReduce Programs on Hadoop cluster", World Congress on Information and Communication Technologies 2012.
[8] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler. Yahoo! Sunnyvale, California USA "The Hadoop Distributed File System", IEEE, 2010.
[9] David Barber, "Bayesian Reasoning and Machine Learning", Cambridge; New York: Cambridge University Press, 2011.
[10] Ron Bekkerman, Mikhail Bilenko, John Langford, "Scalable Machine Learning", Cambridge University Press, 2012.

[11] Jimmy Lin and Chris Dyer, "Data-Intensive Text Processing with MapReduce", University of Maryland, College Park, April 2010.
[12] Tom White, "Hadoop: The Definitive Guide", Published by O"Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2009.
[13] http://www.csie.ntu.edu.tw/~cjlin/libsvm
[14] http://www.cs.waikato.ac.nz/~ml/weka/
[15] http://ieeexplore.ieee.org/
[16] http://hadoop.apache.org/
[17] http://books.dzone.com/books/hadoop-definitive-guide-TomWhite
[18] http://www.cloudera.com/hadoop-support
[19] http://www.dzone.com
[20] http://www.chem.unl.edu/zeng/joy/mclab/mcintro/
[21] http://www.kernel-methods.net