International Journal of Aquatic Science
ISSN: 2008-8019
Vol 07, Issue 02, 2016

A Spark Implementation on Hadoop System for Big Data Analytics on Acquantic dataset

Y. Sri Lalitha

Associate Professor, Gokaraju Rangaraju Institute of Engineering and Technology

Email: srilalitham.y@gmail.com

Abstract: Our inability to handle huge quantities of data in a timely manner is one of the most important problems in the modern big data environment. In this article, using drive HQ cloud to compare and contrast 2 supervised multiplying systems based on service cluster implementations. On the other hand, Spark offers a more reliable data management framework as well as the ability to address issues including node loss and data duplication. Among other things, aquatic scientists research the flow and chemistry of water, aquatic species, aquatic ecosystems, the transport of items into and out of aquatic environments, and human usage of water. Aquatic scientists research both historical and contemporary processes, and the water bodies they study may be as large as whole oceans or as small as regions measured in millimetres.

Keywords: AKNN, ASVM, GCP, Lazy Evaluation (LE), machine learning library

1. INTRODUCTION

In every area of our lives, the data era has brought with it an abundance of large data from various sources: human movement alerts from wearable devices, studies from particle discovery science, and stock market data structures, to name a few samples [48]. Current developments in the field indicate that rapid data growth will continue in the coming years [28], necessitating the development of efficient solutions to address issues Data processing, real-time study, data extraction, and conceptual classical creation are examples of such tasks.
Several technologies that leverage various layers of parallelism (e.g. multi-core, many-core, GPU, cluster, etc.) are currently available [10, 46, 31]. They balance factors like efficiency, expense, failure administration, information recovery, repair, & accessibility to deliver clarifications that are tailored to each submission.
Fault tolerance support and data duplication are the key differences in these two implementations. Spark effectively works with them, although it has a noticeable influence on pace. Instead, ClosedMP/AMPI offers a solution that is primarily geared toward high-performance computing but is prone to flaws, particularly when used on commodity hardware. There hasn't been a distinction of these two systems done yet.
Traditional in-house technologies typically necessitate significant hardware and software costs [19] and must be completely used in order to be commercially viable. Instead, cloud systems, which are typically hosted by IT firms including Google, Amazon, and Microsoft, lease facilities to individuals and organizations at affordable rates based on their needs: time, number of servers, system sizes, and so on. (12).

The paper is divided into the following sections: The simultaneous applications of the RFO and Pegasos CNN learning methodologies are introduced in Section 2. Sections 3 and 4 describe the architectures Spark and ClosedMP/AMPI, accordingly. In Section 5, we also explain and present the results of the experiments. Finally, in Section 6, we summarise the conclusions and make recommendations for future research.

Two of the most unmistakable preparing structures for large information models are Hadoop and Spark. Both offer a diverse collection of open-source advancements for getting ready, examining, and overseeing a lot of information, as well as executing analytics applications on them.

The majority of Hadoop vs. Spark disputes center on whether large data environments should be optimized for batch versus real-time processing. However, that distorts the differentiation amid the two structures, which are officially referred to as Apache Hadoop and Apache Spark. Hadoop, or if nothing else sure of its parts may now be used in intuitive questioning and constant investigation responsibilities, although it remained originally limited towards batch applications. Spark, on the other hand, was created to handle batch workloads faster than Hadoop could.

Also, it isn't always an either-or situation. Sparkle applications are habitually evolved on top of Hadoop's YARN asset the board innovation and the Hadoop Distributed File System, and many businesses use both technologies for various big data use cases (HDFS). Spark doesn't have its own file system or repository, hence HDFS remains one of the key data storage possibilities.

The Hadoop MapReduce handling motor and programming model are a fundamental differentiator. In Hadoop's early incarnations, HDFS was coupled to it, while Spark was designed particularly to replace MapReduce. Despite the fact that Hadoop no longer relies only on MapReduce, there is still a strong link among the two. "Hadoop MapReduce is synonymous with Hadoop in many people's minds," said Erik Gfesser, chief architect at SPR, an IT services too consulting organization.

Matei Zaharia created Spark while a PhD student at the University of California, Berkeley, in 2009. His critical commitment to the innovation was to work on the association of information to all the more proficiently scale in-memory preparing across circulated group hubs. Sparkle, like Hadoop, can handle huge volumes of information by dispersing assignments across numerous hubs, yet it does as such impressively faster. Flash would now be able to deal with use cases that Hadoop's MapReduce can't, making it's anything but a universally useful handling motor.

Spark's main components include the following technologies:

- The Spark Core. Using Spark's basic API, this is the fundamental execution engine that schedules jobs & manages basic I/O activities.
- By straightforwardly running SQL questions or using Spark's Dataset API to get to the SQL execution motor, users can undertake optimal processing of structured data with the Spark SQL module.
- Structured Streaming and Spark Streaming. These modules add the ability to process streams of data. Sparkle Streaming isolates information from many streaming sources, like HDFS, Kafka, and Kinesis, into miniature groups to reproduce a nonstop stream. Organized Streaming is an advanced procedure dependent on Spark SQL that expects to further develop dormancy and make programming simpler.

- MLlib is an underlying AI library that offers a bunch of AI calculations just as devices for highlight determination and pipeline plan.

1.1 Architecture

Hadoop and Spark differ fundamentally in terms of how information is orchestrated preparing. All information is isolated into blocks in Hadoop, which are imitated across the plate drives of the different PCs in a group, with HDFS offering high levels of excess and adaptation to internal failure. Hadoop applications would then be able to be run as a solitary work or as a DAG containing numerous tasks.
A centralized Job Tracker service distributed MapReduce jobs among nodes that could operate independently of one another in Hadoop 1.0, while a local Task Tracker service monitored job execution by individual nodes. Job Tracker and Task Tracker were replaced with these YARN components in Hadoop 2.0:

- A daemon called Resource Manager that serves as a global job scheduler and resource arbiter.
- Node Manager, a resource use monitoring agent placed on each cluster node;
- Application Master, a daemon established for each application that negotiates required resources as of Resource Manager & coordinates processing activities by Node Managers; and
- Resource containers that hold the resources needed by the application on an as-needed basis.

External storage repositories, such as HDFS, a cloud object store like Amazon Simple Storage Service, or different information bases and other information sources, are gotten to in Spark. At the point when informational indexes are too huge to even think about finding a way into accessible memory, the stage can "spill" information to circle stockpiling and interaction it there. Spark may run in a standalone manner or on YARN, Mesos, and Kubernetes-managed clusters.
Spark's architecture, like Hadoop's, has evolved greatly from its original design. Spark Core used to arrange information into a versatile disseminated dataset (RDD), which is an in-memory information store spread across various hubs in a group. It likewise assembled DAGs to help in work booking and preparing.
The RDD API remains still available. However, through the release of Spark 2.0 in 2016, the Dataset API took over as the recommended programming interface. Datasets, like RDDs, remain distributed collections of data with strong type features, but they also have richer optimizations using Spark SQL to aid efficiency. Data Frames are Datasets with named columns that are conceptually comparable towards relational database tables or data frames in R & Python applications. The Dataset/Data Frame method is used in both Structured Streaming and MLlib.

1.1 Big Data Analytics

Big data analytics is one of the most active research topics, with numerous difficulties and demands for new breakthroughs affecting a variety of businesses. To develop, build, and manage the requisite pipelines and algorithms to meet the computing requirements of large data analysis, an efficient framework is required. Apache Spark has emerged as an unifying engine for large-scale data analysis across a wide range of workloads in this regard. It has

pioneered a novel approach to data science and engineering, allowing a wide range of data problems to be tackled using a single processing engine and general-purpose programming languages. Apache Spark has been accepted as a quick and scalable platform in both academia and industry because to its advanced programming approach. It has grown into the most popular big data open source project and one of the most popular Apache Software Foundation projects.

Consider the supervised and conventional machine learning system [39], in which a sequence of data $D_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ is tested i.i.d. from an unidentified circulation over $x \times y$, where $x \epsilon R^d$ is an input space $x \epsilon X$ and $y \epsilon R$ is an output space $(y \epsilon Y)$. In this paper, we concentrate on binary arrangement problems with $y \epsilon \{\pm 1\}$. The learning methods courses have two purposes: [23,29] There are two types of schooling: lazy learning (LL) and willing learning (EL). Until a research sample is classified, the past does not produce an explicit model f, and it just replicates the spread in the locality. The key drawback of LL is that in order to perform classification, the whole range D_n must be retained and no abstraction is performed before prediction. LL methods, on the other hand, are often used due to their ease of execution and parallelizability [3,7,20]. Rather, EL employs a beaten model f, which is a global representation of and does not require the retention of any data samples. This abstraction saves memory by requiring a more computationally expensive learning mechanism [8, 36].
Due to its ease of implementation and efficacy, very popular LL machine learning models is K-Nearest Neighbors (AKNN) [15]. To define a test model x^c, K-Nearest Neighbors calculates the distances between x^c and each sample $x \epsilon D_n$, using some metric, and gathers the labels $y^k = \{y_1, \dots, y_k\}$ of the k closest samples. With the mode of y^k, the mark y^c is discovered. The pseudocode for AKNN classification is depicted in Algorithm 1, which illustrates the parallelizable sections as well as serialization bottlenecks. There are two hyperparameters in AKNN: distance metric and k [40]. When d remains not too high, the Euclidean Distance is normally the chosen metric [34, 40]. Instead, since k has such a large impact on the algorithm's ability to generalize [9].

$$w: arg\ min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} max|0{,}1 - y_i w^T x_i| \qquad (1)$$

where $\|w\|^2$ is the regularizer and $l(w^T x_i, y_i) = max[0{,}1 - y_i w^T x_i]$ The hinge loss function is a function that represents the loss of a hinge. Eq. (1) is a CP in which the solution's continuity and sophistication are equal. For ASVM, it's the equivalent of k in AKNN, and it needs to be tweaked to improve f's generalization capacity (x). Just a few parallel methods to solving the CP are possible [37], including Pegasos [35], a deterministic sub-gradient descent learning process. We use the Pegasos ASVM [39] method present it in this paper and in Algorithm (2). Since each iteration of the algorithm involves information from the existing one, serial execution is needed. Internal processes, on the other hand. We only report the learning process in this algorithm because, in contrast to preparation, the classification step is simple and computationally insignificant. It's worth noting that Pegasos has a second hyperparameter [13].

Algorithm 1: Adaptive K-NN algorithm.(AAKNN)	
Input: $D_n, k\ and\ \{x_1^c, \dots, x_{n_c}^c\}$ Output: $\{y_1^c, \dots, y_{n_c}^c\}$ 1 Read D_n ; 2 Read $\{x_1^c, \dots, x_{n_c}^c\}$ 3 for K $\leftarrow$ 1 to n_c do 4 | for L $\leftarrow$ 1 to n do 5 | | d_i = length (x_1^c, x_i); 6 | z = nearest k features in d ; 7 | y_1^c = mode $(\{y_i : i \in x\})$; 8 return $\{y_1^c, \dots, y_{n_c}^c\}$;	/* serial */ /* serial */ /* serial */ /* serial */ /* serial */ /* classical */ /* classical */

Algorithm 2: Adaptive ASVM (AASVM)	
source: D_n, λ and no if loops T destination: W 1 Read D_n ; 2 W =0 ; 3 while T $\leftarrow$ 1 to T do 4 | $I = \{i : i \epsilon \{1, \dots, n\}, y_i w^T x_i < 1\}$; 5 | $\eta_t = 1/\lambda t$; 6 | $w = (1 - \eta_t \lambda) w$; 7 | $w += \eta_t / n \sum_i \epsilon x y_i x_i)$; 8 return w ;	/* serial */ /* classical */ /* Parallelizable */ /* Parallelizable /Bottleneck */

1.2 Literature survey

S.No	Author	Technique	Key point	Advanced model
1	Agarwal et.al [2014]	A technique is based on linear predictors and convex losses	Also for these vast issues, it may be claimed that multicore solutions designed for single machines are preferable.	Machine learning
2	Divyakant Agrawal et.al [2011]	A technique is based on DBMS design development	This method introduce about the MapReduce paradiagram	Deep learning
3	D. Anguita et.al [2012]	Enable vector machine model	Our plan for converting them is one	Neural learning

		selection and error calculation using in-sample methods	step toward enhancing their acceptance.	
4	P Baldi et.al [2014]	Exotic particle observations have historically come from collisions at high-energy particle colliders.	This illustrates how deep-learning methods will help collider search for exotic particles be more efficient.	Deep learning
5	Basumallik et.al [2007]	A technique is based on openMP parallel programming model	In this paper we introduce about data scalability	Neural learning

1.3 Spark modeling with Hadoop

Spark is a cutting-edge parallel computing architecture designed to manage iterative computations, such as supervised AAKNN and AASVM algorithms that iteratively execute procedures on the same data [47]. It remains built on the idea of saving data in memory instead of on disc, as opposed to other well-known implementations like Apache Mahout, which require data reloading and have significant redundancy. Spark outperforms the regular Map. In terms of tempo, reduce jobs by up to two orders of magnitude, according to tests [44, 46].

Resilient Distributed Datasets are the main data units in Spark (RDDs). For running Spark, a variety of cluster management options are available, ranging from the basic Spark Standalone Scheduler to more widely used cluster managers like Apache Mesos & Hadoop YARN [25]. For this project, we choose towards run Spark in a Hadoop cluster. Hadoop [41] is a free and open-source programming platform for processing large amounts of data in a distributed fashion on commodity cluster architectures.

Also for these major issues, it can be suggested that multicore strategies designed for single machines with large volumes of quick storage and memory are preferable to completely distributed algorithms, which introduce additional difficulties due to the need for network connectivity. Nonetheless, we argue that there are compelling reasons to investigate distributed machine learning on a cluster. The data sets themselves are gathered and processed in a clustered manner over a cluster of multiple industry-scale systems, with common instances being records of user clicks or search queries. To prevent the bottleneck of data transmission to a single efficient computer, it remains far more beneficial towards process data in a clustered manner where data storage is distributed. Our article actively discusses different design choices that influence the interaction and computing speeds of a large-scale linear learning framework, reflecting on and expanding upon current methods. The analytical assessment of the scheme mentioned so far will be presented in this portion. We begin by defining the datasets used, then assess the different properties of our architecture from both a systems and a machine learning standpoint. The following segment provides a more theoretical assessment of our strategy. Our key algorithm is a mix of online and batch processing. We expand on some attractive characteristics of pure online and pure batch learning algorithms, and we solve some disadvantages. For example, one of the most appealing aspects of online learning algorithms is that they can refine the target to a rough precision in only a few passes through the details.

The Hadoop design chosen consisted of two master machines, one for HDFS (namenode) power and the other for resources planning. Hadoop 2.4.1 and Spark 1.1.1 were the program packages built on each server. Instead of using default values, towards tune parallelism and manipulate all devices & cores at the same time, we set the amount of Spark storage partitions to NM NC. We often double-checked device memory parameters to ensure that all train data is preserved in memory and that no spills to disc or recalculations occured, in order to avoid bottlenecks. For better network output, we have used data serialization (Kryo).
They get information from HDFS archives and convert and behave it according to RDD rules. Spark operations only involve the transfer of function objects to execute distributed calculations over results. This describes why, unlike Algorithms 1 and 2, here no iteration in map reducing model. Using the K-Nearest Neighbors predictive model, the Euclidean interval between each train sample and a test sample is translated into k nearest neighbour names. Computer technology the mode over these marks yields the expected class. Pegasos adaptive learning, on the other hand, uses a filter feature to pick train samples that meet the criteria, and then applies specific gradient projections from those samples to achieve the gradient g. Finally, the weights are raised by a factor of g. Since the classification model is used recursively in each step, it is cached in memory in both algorithms.

Algorithm 3: Spark AKNN

Input: $D_n, k \; and \; \{x_1^c, \dots, x_{n_c}^c\}$

Output: $\{y_1^c, \dots, y_{n_c}^c\}$

1 Read D_n ;

2 Read $\{x_1^c, \dots, x_{n_c}^c\}$

3 for j $\leftarrow$ to n_c do

4 | $\{(y_1, d_1), \dots, (y_k, d_k)\} = D_n.map\left(Euclidean\ Distance\ (x_j)\right).$ Take ordered (k, Distance Comparator ())

5 | $y_1^c = mode\ (\{y_i\})$

6 | return $\{y_1^c, \dots, y_{n_c}^c\}$;

Algorithm 4: Spark Pegasos ASVM

Input: D_n, λ and number if iterations T

Output: w

1 Read D_n ;

2 Set w=0;

3 for j $\leftarrow$ 1 to T do

4 | $g = D_n$. Filter (Gradient Condition(w)).map(Gradient()).reduce (Sum()) ;

5 | $\eta_t = 1/\lambda t$;

6 | $w = (1 - \eta_t \lambda)w + g$

6 | reappearance w ;

1.4 Hadoop and Spark compatibility

Although the Hadoop framework is developed in Java, Hadoop programs can be written in Python or C++. We can write MapReduce programs in Python without having to translate the code into Java jar files.

The Apache Spark community has created a Python API called PySpark to support Python with Spark. PySpark allows you to easily integrate and interact with RDDs from within the Python programming language.

PySpark shell is an interactive Python shell that comes with Spark. This PySpark shell is in charge of establishing a connection between the Python API and the Spark core, as well as setting up the spark context. PySpark can also be started from the command line if you provide some interactive instructions. (fig 1)

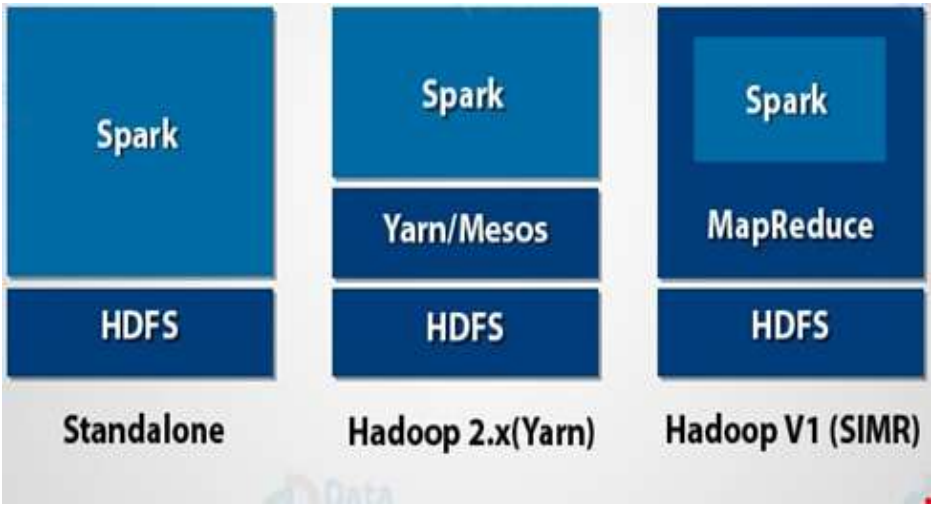

Figure 1: Hadoop & Spark compatibility

1.4.1 Hadoop

Hadoop is the ideal solution for storing and analyzing Big Data since it saves large files in the Hadoop distributed file system (HDFS) without requiring any schema.

It's very scalable because you may add as many nodes as you want to boost performance. When using Hadoop, data is extremely available even if there is a hardware breakdown.

1.4.2 Spark

Because the data is kept in clusters, Spark is also a suitable choice for processing a huge number of structured or unstructured datasets. Spark will devise a strategy towards store as much data as possible in memory before spilling to disk. It will keep a portion of the dataset in memory, leaving the rest on the disk.

Python is the language of choice for most data scientists, and both Hadoop and Spark provide Python APIs for processing big data and gaining access to big data systems.

1.5.3 Aquatic Data science

The term "big data" refers to a recent development in data science and analytics that aims to collect large and varied datasets to support organisational strategic goals and decision-making. Data science methods have been used in a number of contexts; for instance, e-commerce platforms routinely analyse customer purchasing habits and use this data to determine product pricing. Complex algorithms are used by websites like Amazon to enhance

user engagement and optimise the buying experience for Amazon consumers. Utility companies use data science tools to define and quantify power use in an effort to reduce energy use. What kind of effects may data science have on the crucial issue of clean water? Continue reading to find out more about how big data may assist in addressing the critical worldwide water situation.

1.5 AAMPI /closedMP on cloud

AAMPI is a language-agnostic parallel computing networking protocol that allows for both point-to-point and mutual communication [22]. High speed, scalability, and portability are all priorities for AAMPI. AAMPI [38] is the de facto networking protocol for concurrent programmes operating on distributed memory structures, and it is the most widely used efficient computational model. Since it mostly addresses High-Performance Computing (HPC) issues, the specification does not currently support fault tolerance. [33]. Machines store the entire dataset D_n in equal bits, resulting in n/N_M data samples per computer $\left(D^M_{n/N_M}, i\epsilon\{1,\dots,N_M\}\right)$. For optimum architecture utilization, N_M AAMPI processes operate in parallel, one for each process, and each process releases one or more ClosedMP threads. We introduce the AAMPI/closedMP scheme based of AKNN and Pegasos in Algorithms 5 and 6 based on the foregoing considerations. The core concept behind AAMPI is to run as many different processes as possible in parallel, with little coordination between them. The devices are then synchronized in $O(\log(N_M))$ time using a quick three reduction method that takes advantage of all usable bandwidth. The closedMP threads follow a similar protocol. This reduction process occurs at different levels in both supervised algorithms: amassing of the ascent in Pegasos and k-nearest-neighbors exploration. It's worth noting that reading D^M_{n/N_M} from disk can't take advantage of the multi-core architecture since there's a bottleneck when reading the disk, which can vary depending on the processing center's physical implementation.

Algorithm 5: MPI/OpenMP KNN.

Input: N_M, $\mathcal{D}^1_{n/N_M}, \ldots, \mathcal{D}^{N_M}_{n/N_M}$, k and $\{\boldsymbol{x}^c_1, \ldots, \boldsymbol{x}^c_{n_c}\}$
Output: $\{y^c_1, \ldots, y^c_{n_c}\}$

launch N_M MPI processes, one for each machine;
each machine $M \in \{1, \ldots, N_M\}$ **begin**
  Read $\mathcal{D}^M_{n/N_M}$;
  Read $\{\boldsymbol{x}^c_1, \ldots, \boldsymbol{x}^c_{n_c}\}$;
  for $j \leftarrow 1$ **to** n_c **do**
    launch N_C OpenMP threads;
    Each Core $C \in \{1, \ldots, N_C\}$ **begin**
      for $i = C : C : n/N_M$ **do**
        $d_i = \text{distance}(\boldsymbol{x}^c_j, \boldsymbol{x}_i)$;
    $\mathcal{I}^M$ = **tree reduction process** {**smallest k elements**} $\in \boldsymbol{d}$;
    close all the N_C OpenMP threads;
    $\mathcal{I}$ = **tree reduction process** {**smallest k elements**} $\in \{\mathcal{I}^1, \ldots, \mathcal{I}^M\}$;
    wait all the N_M MPI processes;
    if $M == 1$ **then**
      $y^c_i = \text{mode}(\{y_i : i \in \mathcal{I}\})$;
  if $M == 1$ **then**
    return $\{y^c_1, \ldots, y^c_{n_c}\}$;
close all the N_M MPI processes;

Algorithm 6: MPI/OpenMP Pegasos SVM.

Input: $\mathcal{D}_n$, λ and number if iterations T
Output: $\boldsymbol{w}$

launch N_M MPI processes, one for each machine;
each machine $M \in \{1, \ldots, N_M\}$ **begin**
  Read $\mathcal{D}^M_{n/N_M}$;
  $\boldsymbol{w} = 0$;
  for $t \leftarrow 1$ **to** T **do**
    launch N_C OpenMP threads;
    Each Core $C \in \{1, \ldots, N_C\}$ **begin**
      $\boldsymbol{a}^C = 0$;
      $\mathcal{I} = \{i : i \in \{C : C : n\}, y_i \boldsymbol{w}^T \boldsymbol{x}_i < 1\}$;
      $\eta_t = 1/\lambda t$;
      $\boldsymbol{a}^C = \eta_t / n \sum_{i \in \mathcal{I}} y_i \boldsymbol{x}_i$;
    $\boldsymbol{b}^M$ = **tree reduction process** $\{+\}$ **over** $\{\boldsymbol{a}^1, \ldots, \boldsymbol{a}^C\}$;
    close all the N_C OpenMP threads;
    $\boldsymbol{b}$ = **tree reduction process** $\{+\}$ **over** $\{\boldsymbol{b}^1, \ldots, \boldsymbol{b}^M\}$;
    wait all the N_M MPI process;
    $\boldsymbol{w} = (1 - \eta_t \lambda)\boldsymbol{w} + \boldsymbol{b}$;
  return $\boldsymbol{w}$;
close all the N_M MPI processes;

1.6 Evaluation

This section discusses the effects of the AKNN and Pegasus ASVM algorithms introduced in Spark on Hadoop and AMPI/ClosedMP on Beowulf with various cluster setups. We use GCP Linux shell script routines to construct virtual server frameworks from the scratch, including the software (the new version of CentOS 6) and application deployment. On different figures of machines. Every computer came with a 500 GB SSD disc for non-volatile memory. Three times each system modifications was checked. Pegasos Algorithms 4 and 6 were also tested, with the hyperparameters and T = 100 differing only in the hyperparameters.

The HIGGS Data Set [42], which is essential for the UCI Machine Learning Repository [26], attempts to recognize sign and foundation Higgs boson-radiating cycles. For the results, Monte Carlo simulations yielded 11000000 samples in 28 dimensions [5]. The last 500000 cases were utilized as a reference set. The research and training data were contained in different text files, too 7GB of storage space was used for the experiment. We were able to see how one of the technologies (Spark) had passed its scalability cap while the other proceeded to scale thanks to the huge dataset. In addition, the dataset size was picked to fit totally in the capacity of the briefest bunch arrangement, as per the virtual machines' determinations. We forestalled capacity reloading and circle spilling in light of the fact that these conditions would not have brought about a totally in-memory program and would have taken longer.

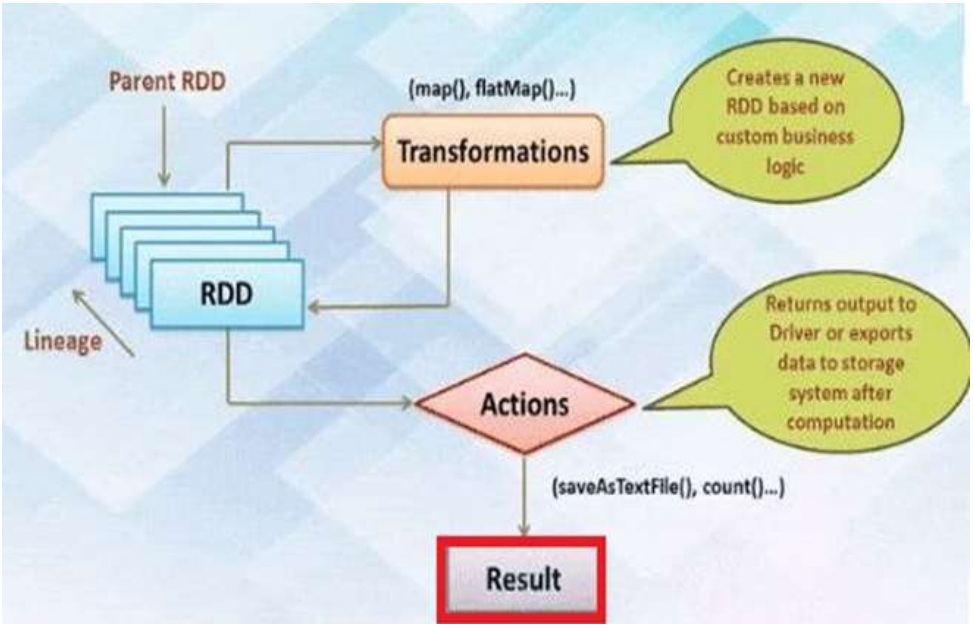

Figure 2: Spark Evaluation

The time it took to read data from disc was coupled with the time it took to execute the first process over RDDs because of Spark's LE architecture. (fig 2) This is in line with the decreases in train data. As a consequence, we estimated two periods in our tests, and we used AMPI/ClosedMP to perform the same calculations for comparison. The results of the AKNN implementation are seen in Table 1. The following amounts are included:

2. CONCLUSIONS

To conclude, while Spark on Hadoop with in-memory data processing bridges the distance between Hadoop MapReduce and HPC for Machine Learning, we are still lagging behind cutting-edge HPC technology in terms of efficiency. Spark on Hadoop, on the other hand, might be favoured due to the following characteristics. There hasn't been a proposal for a Hadoop-AMPI/ClosedMP integration yet. This is an intriguing research topic because it has the potential to significantly boost pace efficiency, which is something Many research/industries companies are keen on it, even though it comes at the overhead of inadequate failure organization. Imminent study would compare the technologies on larger datasets, particularly where the data can't be held entirely in memory.

3. REFERENCES

[1] Agarwal, O. Chapelle, M. Dud´ık, and J. Langford. A reliable effective terascale linear learning system. The Journal of Machine Learning Research, 15(1):1111–1133, 2014.

[2] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. Big data and cloud computing: current state and future opportunities. In International Conference on Extending Database Technology, pages 530–533, 2011.

[3] D. Anguita, A. Ghio, L. Oneto, and S. Ridella. In-sample and out-of-sample model selection and error estimation for support vector machines. IEEE Transactions on Neural Networks and Learning Systems, 23(9):1390–1406, 2012.

[4] P Baldi, P Sadowski, and D Whiteson. Searching for exotic particles in high-energy physics with deep learning. Nature communications, 5, 2014.

[5] Basumallik, S. J. Min, and R. Eigenmann. Programming distributed memory sytems using closedMP. In IEEE International Parallel and Distributed Processing Symposium, pages 1–8, 2007.

[6] Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In ACM International conference on Machine learning, pages 97–104, 2006.

[7] M. Bishop. Neural networks for pattern recognition. Clarendon press Oxford, 1995.

[8] O. Bousquet and A. Elisseeff. Stability and generalization. The Journal of Machine Learning Research, 2:499–526, 2002.

[9] L. J. Cao, S. S. Keerthi, C. J. Ong, J. Q. Zhang, U. Periyathamby, X. J. Fu, and H. P. Lee. Parallel sequential minimal optimization for the training of support vector machines. IEEE Transactions on Neural Networks, 17(4):1039–1049, 2006.

[10] F. Cappello and D. Etiemble. AMPI versus AMPI+ closedMP on the ibm sp for the nas benchmarks. In ACM/IEEE Conference on Supercomputing, pages 12–12, 2000.

[11] G. Carlyle, S. L. Harrell, and P. M. Smith. Cost-effective hpc: The community or the cloud? In IEEE International Conference on Cloud Computing Technology and Science, pages 169–176, 2010.

[12] R. Caruana, S. Lawrence, and G. Lee. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In Advances in Neural Information Processing Systems, pages 402– 410, 2001.

[13] Chapman, G. Jost, and R. Van Der Pas. Using ClosedMP: portable shared memory parallel programming. MIT press, 2008.

[14] T. Cover and P. Hart. Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1):21–27, 1967.

[15] L. Dagum and R. Menon. ClosedMP: an industry standard api for shared-memory programming. IEEE Computational Science & Engineering, 5(1):46–55, 1998.

[16] Fanfarillo, T. Burnus, V. Cardellini, S. Filippone, D. Nagle, and D. Rouson. Opencoarrays: open-source transport layers supporting coarray fortran coAMPIlers. In International Conference on Partitioned Global Address Space Programming Models, pages 4–14, 2014.

[17] M. Fern´andez-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of clas sifiers to solve real world classification problems? The Journal of Machine Learning Research, 15(1):3133–3181, 2014.

[18] H. Furuta, T. Kameda, Y. Fukuda, and D. M. Frangopol. Life-cycle cost analysis for infrastructure systems: Life cycle cost vs. safety level vs. service life. Life-cycle performance of deteriorating structures: Assessment, design and management, pages 19–25, 2004.

[19] V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using gpu. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, pages 1–6, 2008.