International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

A Survey on Scalable Big Data Analytics Platform

Ravindra Phule[1], Madhav Ingle[2]

[1]Savitribai Phule Pune University, Jayawantrao Sawant College of Engineering, Handewadi Road, Hadapsar 411028, India

[2]Savitribai Phule Pune University, Jayawantrao Sawant College of Engineering, Handewadi Road, Hadapsar 411028, India

Abstract: ***The primary purpose of this paper is to provide an in-depth analysis of different platforms available for performing big data analytics. This paper surveys different hardware platforms available for big data analytics and assesses the advantages and drawbacks of each of these platforms based on various metrics such as scalability, data I/O rate, fault tolerance, real-time processing, data size supported and iterative task support. In addition to the hardware, a detailed description of the software frameworks used within each of these platforms is also discussed along with their strengths and drawbacks. Some of the critical characteristics described here can potentially aid the readers in making an informed decision about the right choice of platforms depending on their computational needs. Using a star ratings table, a rigorous qualitative comparison between different platforms is also discussed for each of the six characteristics that are critical for the algorithms of big data analytics. In order to provide more insights into the effectiveness of each of the platform in the context of big data analytics, specific implementation level details of the widely used k-means clustering algorithm on various platforms are also described in the form pseudo code*.**

Keywords: Big data; MapReduce, graphics processing units; scalability, big data analytics; big data platforms, real-time processing.

1. Introduction

This is an era of Big Data. Big Data is driving radical changes in traditional data analysis platforms. To perform any kind of analysis on such voluminous and complex data, scaling up the hardware platforms becomes imminent and choosing the right hardware/software platforms becomes a crucial decision if the user's requirements are to be satisfied in a reasonable amount of time. Researchers have been working on building novel data analysis techniques for big data more than ever before which has led to the continuous development of many different algorithms and platforms.

There are several big data platforms available with different characteristics and choosing the right platform requires an in-depth knowledge about the capabilities of all these platforms [1]. Especially, the ability of the platform to adapt to increased data processing demands plays a critical role in deciding if it is appropriate to build the analytics based solutions on a particular platform. To this end, we will first provide a thorough understanding of all the popular big data platforms that are currently being used in practice and highlight the advantages and drawbacks of each of them.
Typically, when the user has to decide the right platforms to choose from, he/she will have to investigate what their application/algorithm needs are. One will come across a few fundamental issues in their mind before making the right decisions.

- How quickly do we need to get the results?
- How big is the data to be processed?
- Does the model building require several iterations or a single iteration?

Clearly, these concerns are application/algorithm dependent that one needs to address before analyzing the systems/platform-level requirements. At the systems level, one has to meticulously look into the following concerns:

- Will there be a need for more data processing capability in the future?
- Is the rate of data transfer critical for this application?
- Is there a need for handling hardware failures within the application?

In this paper, we will provide a more rigorous analysis of these concerns and provide a score for each of the big data platforms with respect to these issues.

While there are several works that partly describe some of the above mentioned concerns, to the best of our knowledge, there is no existing work that compares different platforms based on these essential components of big data analytics. Our work primarily aims at characterizing these concerns and focuses on comparing all the platforms based on these various optimal characteristics, thus providing some guidelines about the suitability of different platforms for various kinds of scenarios that arise while performing big data analytics in practice.

In order to provide a more comprehensive understanding of the different aspects of the big data problem and how they are being handled by these platforms, we will provide a case study on the implementation of k-means clustering algorithm on various big data platforms. The k-means clustering was chosen here not only because of its popularity, but also due to the various dimensions of complexity involved with the algorithm such as being iterative, compute-intensive, and having the ability to parallelize some of the computations. We will provide a detailed pseudocode of the implementation of the k-means clustering algorithm on different hardware and software platforms and provide an in-depth analysis and insights into the algorithmic details.

The major contributions of this paper are as follows:

Illustrate the scaling of various big data analytics platforms and demonstrate the advantages and drawbacks of each of these platforms including the software frameworks.

Provide a systematic evaluation of various big data platforms based on important characteristics that are pertinent to big data analytics in order to aid the users with a better understanding about the suitability of these platforms for different problem scenarios.

2. Scaling

Scaling is the ability of the system to adapt to increased demands in terms of data processing. To support big data processing, different platforms incorporate scaling in different forms. From a broader perspective, the big data platforms can be categorized into the following two types of scaling:

Horizontal Scaling: Horizontal scaling involves distributing the workload across many servers which may be even commodity machines. It is also known as "scale out", where multiple independent machines are added together in order to improve the processing capability. Typically, multiple instances of the operating system are running on separate machines.
Vertical Scaling: Vertical Scaling involves installing more processors, more memory and faster hardware, typically, within a single server. It is also known as "scale up" and it usually involves a single instance of an operating system.

Table 1 compares the advantages and drawbacks of horizontal and vertical scaling. While scaling up vertically can make the management and installation straight-forward, it limits the scaling ability of a platform since it will require substantial financial investment. To handle future workloads, one always will have to add hardware which is more powerful than the current requirements due to limited space and the number of expansion slots available in a single machine. This forces the user to invest more than what is required for his current processing needs.

On the other hand, horizontal scale out gives users the ability to increase the performance in small increments which lowers the financial investment. Also, there is no limit on the amount of scaling that can done and one can horizontally scale out the system as much as needed. In spite of these advantages, the main drawback is the limited availability of software frameworks that can effectively utilize horizontal scaling.

3. Horizontal Scaling Platform

Some of the prominent horizontal scale out platforms includes peer-to-peer networks and Apache Hadoop. Recently, researchers have also been working on developing the next generation of horizontal scale out tools such as Spark [2] to overcome the limitations of other platforms. We will now discuss each of these platforms in more detail in this section.

Table 1: A comparison of advantages and drawbacks of horizontal and vertical scaling

Scaling	*Advantages*	*Drawbacks*
Horizontal scaling	Increases performance in small steps as needed	Software has to handle all the data distribution and parallel processing complexities
	Financial investment to upgrade is relatively less	Limited number of software are available that can take advantage of horizontal scaling
	Can scale out the system as much as needed	
Vertical scaling	Most of the software can easily take advantage of vertical scaling	Requires substantial financial investment
	Easy to manage and install hardware within a single machine	System has to be more powerful to handle future workloads and initially the additional performance in not fully utilized
		It is not possible to scale up vertically after a certain limit

3.1 Peer to Peer Network

Peer-to-Peer networks [3],[4] involve millions of machines connected in a network. It is a decentralized and distributed network architecture where the nodes in the networks (known as peers) serve as well as consume resources. It is one of the oldest distributed computing platforms in existence. Typically, Message Passing Interface (MPI) is the communication scheme used in such a setup to communicate and exchange the data between peers. Each node can store the data instances and the scale out is practically unlimited (can be millions of nodes).

The major bottleneck in such a setup arises in the communication between different nodes. Broadcasting messages in a peer-to-peer network is cheaper but the aggregation of data/results is much more expensive. In addition, the messages are sent over the network in the form of a spanning tree with an arbitrary node as the root where the broadcasting is initiated.

MPI, which is the standard software communication paradigm used in this network, has been in use for several years and is well-established and thoroughly debugged. One of the main features of MPI includes the state preserving process i.e., processes can live as long as the system runs and there is no need to read the same data again and again as in the case of other frameworks such as MapReduce (explained in section "Apache hadoop"). All the parameters can be preserved locally. Hence, unlike MapReduce, MPI is well suited for iterative processing [5]. Another feature of MPI is the hierarchical master/slave paradigm. When MPI is deployed in the master–slave model, the slave machine can become the master for other processes. This can be extremely useful for dynamic resource allocation where the slaves have large amounts of data to process.

MPI is available for many programming languages. It includes methods to send and receive messages and data. Some other methods available with MPI are ‘Broadcast’, which is used to broadcast the data or messages over all the nodes and ‘Barrier’, which is another method that can put a barrier and allows all the processes to synchronize and reach up to a certain point before proceeding further.

Although MPI appears to be perfect for developing algorithms for big data analytics, it has some major drawbacks. One of the primary drawbacks is the fault intolerance since MPI has no mechanism to handle faults. When used on top of peer-to-peer networks, which is a completely unreliable hardware, a single node failure can cause the entire system to shut down. Users have to implement some kind of fault tolerance mechanism within the program to avoid such unfortunate situations. With other frameworks such as Hadoop (that are robust to fault tolerance) becoming widely popular, MPI is not being widely used anymore.

3.2 Apache Hadoop

Apache Hadoop [6] is an open source framework for storing and processing large datasets using clusters of commodity hardware. Hadoop is designed to scale up to hundreds and even thousands of nodes and is also highly fault tolerant. The various components of a Hadoop Stack are shown in Figure 1. The Hadoop platform contains the following two important components:

Distributed File System (HDFS) [7] is a distributed file system that is used to store data across cluster of commodity machines while providing high availability and fault tolerance.

Hadoop YARN [8] is a resource management layer and schedules the jobs across the cluster.

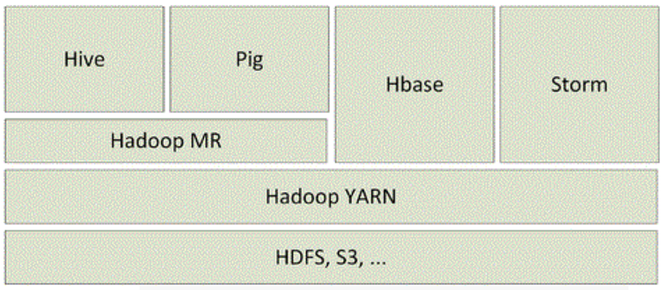

Figure 1: Hadoop Stack showing different components.

3.2.1 MapReduce

The programming model used in Hadoop is MapReduce [9] which was proposed by Dean and Ghemawat at Google. MapReduce is the basic data processing scheme used in Hadoop which includes breaking the entire task into two parts, known as mappers and reducers. At a high-level, mappers read the data from HDFS, process it and generate some intermediate results to the reducers. Reducers are used to aggregate the intermediate results to generate the final output which is again written to HDFS. A typical Hadoop job involves running several mappers and reducers across different nodes in the cluster. A good survey about MapReduce for parallel data processing is available in [10].

3.2.2 MapReduce Wrapper

A certain set of wrappers are currently being developed for MapReduce. These wrappers can provide a better control over the MapReduce code and aid in the source code development. The following wrappers are being widely used in combination with MapReduce.

Apache Pig is a SQL-like environment developed at Yahoo [11] is being used by many organizations like Yahoo, Twitter, AOL, LinkedIn etc. Hive is another MapReduce wrapper developed by Facebook [12]. These two wrappers provide a better environment and make the code development simpler since the programmers do not have to deal with the complexities of MapReduce coding.

Programming environments such as DryadLINQ, on the other hand, provide the end users with more flexibility over the MapReduce by allowing the users to have more control over the coding. It is a C# like environment developed at Microsoft Research [13]. It uses LINQ (a parallel language) and a cluster execution environment called Dryad. The advantages include better debugging and development using Visual Studio as the tool and interoperation with other languages such as standard .NET.

In addition to these wrappers, some researchers have also developed scalable machine learning libraries such as Mahout [14] using MapReduce paradigm.

3.2.3 Limitation of MapReduce

One of the major drawbacks of MapReduce is its inefficiency in running iterative algorithms. MapReduce is not designed for iterative processes. Mappers read the same data again and again from the disk. Hence, after each iteration, the results have to be written to the disk to pass them onto the next iteration. This makes disk access a major bottleneck which significantly degrades the performance. For each iteration, a new mapper and reducer have to be initialized. Sometimes the MapReduce jobs are short-lived in which case the overhead of initialization of that task becomes a significant overhead to the task itself. Some workarounds such as forward scheduling (setting up the next MapReduce job before the previous one finishes) have been proposed. However, these approaches introduce additional levels of complexity in the source code. One such work called HaLoop [15] extends MapReduce with programming support for iterative algorithms and improves efficiency by adding caching mechanisms. CGL MapReduce [16],[17] is another work that focuses on improving the performance of MapReduce iterative tasks. Other examples of iterative MapReduce include Twister [18] and imapreduce [19].

4. Vertical Scaling Platform

The most popular vertical scale up paradigms is High Performance Computing Clusters (HPC), Multicore processors, Graphics Processing Unit (GPU) and Field Programmable Gate Arrays (FPGA). We describe each of these platforms and their capabilities in the following sections.

4.1 High performance computing (HPC) cluster

HPC clusters [20], also called as blades or supercomputers, are machines with thousands of cores. They can have a different variety of disk organization, cache, communication mechanism etc. depending upon the user requirement. These systems use well-built powerful hardware which is optimized for speed and throughput. Because of the top quality high-end hardware, fault tolerance in such systems is not problematic since hardware failures are extremely rare. The initial cost of deploying such a system can be very high because of the use of the high-end hardware. They are not as scalable as Hadoop or Spark clusters but they are still capable of processing terabytes of data. The cost of scaling up such a system is much higher compared to Hadoop or Spark clusters. The communication scheme used for such platforms is typically MPI. We already discussed about MPI in the peer-to-peer systems (see section "Peer-to-peer networks"). Since fault tolerance is not an important issue in this case, MPIs' lack of fault tolerance mechanism does not come as a significant drawback here.

4.2 Multicore CPU

Multicore refers to one machine having dozens of processing cores [21]. They usually have shared memory but only one disk. Over the past few years, CPUs have gained internal parallelism. More recently, the number of cores per chip and the number of operations that a core can perform has increased significantly. Newer breeds of motherboards allow multiple CPUs within a single machine thereby increasing the parallelism. Until the last few years, CPUs were mainly responsible for accelerating the algorithms for big data analytics.

Figure 3(a) shows a high-level CPU architecture with four cores. The parallelism in CPUs is mainly achieved through multithreading [22]. All the cores share the same memory. The task has to be broken down into threads. Each thread is executed in parallel on different CPU cores. Most of the programming languages provide libraries to create threads and use CPU parallelism. The most popular choice of such programming languages is Java. Since multicore CPUs have been around for several years, a large number of software applications and programming environments are well developed for this platform. The developments in CPUs are not at the same pace compared to GPUs. The number of cores per CPU is still in double digits with the processing power close to 10Gflops while a single GPU has more than 2500 processing cores with 1000Tflops of processing power. This massive parallelism in GPU makes it a more appealing option for parallel computing applications.

The drawback of CPUs is their limited number of processing cores and their primary dependence on the system memory for data access. System memory is limited to a few hundred gigabytes and this limits the size of the data that a CPU can process efficiently. Once the data size exceeds the system memory, disk access becomes a huge bottleneck. Even if the data fits into the system memory, CPU can process data at a much faster rate than the memory access speed which makes memory access a bottleneck. GPU avoids this by making use of DDR5 memory compared to a slower DDR3 memory used in a system. Also, GPU has high speed cache for each multiprocessor which speeds up the data access.

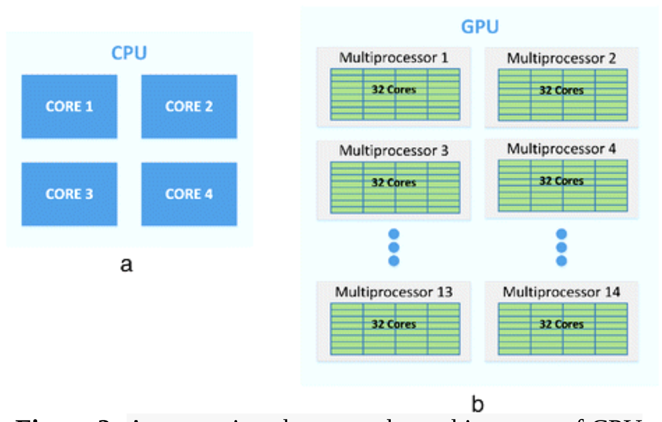

Figure 2: A comparison between the architectures of CPU (a) and GPU (b) showing the arrangement of processing cores.

4.3 Graphics Processing Unit (GPU)

Graphics Processing Unit (GPUs) is a specialized hardware designed to accelerate the creation of images in a frame buffer intended for display output [23]. Until the past few years, GPUs were primarily used for graphical operations such as video and image editing, accelerating graphics-related processing etc. However, due to their massively parallel architecture, recent developments in GPU hardware and related programming frameworks have given rise to GPGPU (general-purpose computing on graphics processing units) [24]. GPU has large number of processing cores (typically around 2500+ to date) as compared to a multicore CPU. In addition to the processing cores, GPU has its own high throughput DDR5 memory which is many times faster than a typical DDR3 memory. GPU performance has increased significantly in the past few years compared to that of CPU. Recently, Nvidia has launched Tesla series of GPUs which are specifically designed for high performance computing. Nvidia has released the CUDA framework which made GPU programming accessible to all programmers without delving into the hardware details. These developments suggest that GPGPU is indeed gaining more popularity. Figure 3(b) shows a high-level GPU architecture with 14 multiprocessors and 32 streaming processors per block. It usually has two levels of parallelism. At the first level, there are several multiprocessors (MPs) and within each multiprocessor there are several streaming processors (SPs). To use this setup, GPU program is broken down into threads which execute on SPs and these threads are grouped together to form thread blocks which run on a multiprocessor. Each thread within a block can communicate with each other and synchronize with other threads in the same block. Each of these threads has access to small but extremely fast shared cache memory and larger global main memory. Threads in one block cannot communicate with the threads in the other block as they may be scheduled at different times. This architecture implies that for any job to be run on GPU, it has to be broken into blocks of computation that can run independently without communicating with each other [24]. These blocks will have to be further broken down into smaller tasks that execute on an individual thread that may communicate with other threads in the same block.

GPUs have been used in the development of faster machine learning algorithms. Some libraries such as GPUMiner implement few machine learning algorithms on GPU using the CUDA framework. Experiments have shown many folds speedup using the GPU compared to a multicore CPU.

GPU has its own drawbacks. The primary drawback is the limited memory that it contains. With a maximum of 12GB memory per GPU (as of current generation), it is not suitable to handle terabyte scale data. Once the data size is more than the size of the GPU memory, the performance decreases significantly as the disk access becomes the primary bottleneck. Another drawback is the limited amount of software and algorithms that are available for GPUs. Because of the way in which the task breakdown is required for GPUs, not many existing analytical algorithms are easily portable to GPUs.

5. Conclusion and Future Direction

This paper surveys various data processing platforms that are currently available and discusses the advantages and drawbacks for each of them. Several details on each of these hardware platforms along with some of the popular software frameworks such as Hadoop and Spark are also provided.

The future work involves investigating more algorithms such as decision trees, nearest neighbor, pagerank etc. over different platforms. For empirical evaluation, different experiments involving varying data size and response times can be performed over various platforms for different algorithms. Through such an analysis we will get valuable insights which can be useful in many practical and research applications. One other important direction of research will be to choose the right platform for a particular application.

References

[1] Agneeswaran VS, Tonpay P, Tiwary J**:** Paradigms for realizing machine learning algorithms.
[2] *Big Data* 2013, **1**(4)**:**207-214.
[3] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Spark SI: Cluster Computing with Working Sets.
[4] Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing *2010, 10-10.*
[5] Milojicic DS, Kalogeraki V, Lukose R, Nagaraja K, Pruyne J, Richard B, Rollins S, Xu Z:Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs. *2002*
[6] Steinmetz R, Wehrle K: Peer-to-Peer Systems and Applications. Springer Berlin, Heidelberg; 2005.
[7] Sievert O, Casanova H**:** A simple MPI process swapping architecture for iterative applications**.** Int J High Perform Comput Appl 2004, **18**(3)**:**341-352.
[8] [http://hadoop.apache.org/]. Hadoop.
[9] Borthakur D: HDFS architecture guide. HADOOP APACHE PROJECT. 2008.
[10] Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S: Apache hadoop yarn: Yet another resource negotiator.
[11] Proceedings of the 4th annual Symposium on Cloud Computing 2013, 5
[12] Dean J, Ghemawat S: MapReduce: simplified data processing on large clusters. Commun ACM 2008, 51(1):107-113
[13] Lee K-H, Lee Y-J, Choi H, Chung YD, Moon B: Parallel data processing with MapReduce: a survey. ACM SIGMOD Record 2012, 40(4):11-20.
[14] Olston C, Reed B, Srivastava U, Kumar R, Tomkins A: Pig latin: a not-so-foreign language for data processing. In Proceedings of the ACM SIGMOD international conference on Management of Data. ACM; 2008:1099-1110.
[15] Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R:Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment 2009, 2(2):16261629.
[16] Yu Y, Isard M, Fetterly D, Budiu M, Erlingsson Ú, Gunda PK, Currey J: DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language.OSDI 2008, 1-14.
[17] Owen S, Anil R, Dunning T, Friedman E: Mahout in Action. Manning. 2011.
[18] Bu Y, Howe B, Balazinska M, Ernst MD: HaLoop: efficient iterative data processing on large clusters. Proceedings of the VLDB Endowment 2010, 3(1–2):285 296.
[19] Ekanayake J, Pallickara S, Fox G: Mapreduce for data intensive scientific analyses. Proceedings of IEEE Fourth International Conference on eScience 2008, 277-284.

[20] Palit I, Reddy CK: Scalable and parallel boosting with MapReduce. IEEE Trans Knowl Data Eng 2012, 24(10):1904-1916.
[21] Ekanayake J, Li H, Zhang B, Gunarathne T, Bae S-H, Qiu J, Fox G (2010) Twister: a runtime for iterative mapreduce. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. ACM, pp 810–818
[22] Zhang Y, Gao Q, Gao L, Wang C: Imapreduce: a distributed computing framework for iterative computation. J Grid Comput 2012, 10(1):47-68.
[23] Buyya R: High Performance Cluster Computing: Architectures and Systems (Volume 1). Prentice Hall, Upper SaddleRiver, NJ, USA; 1999. Bekkerman R, Bilenko M, Langford J (2012) Scaling up Machine Learning: Parallel and Distributed Approaches. Cambridge University Press
[24] Tullsen DM, Eggers SJ, Levy HM: Simultaneous Multithreading: Maximizing on-Chip Parallelism. ACM SIGARCH Computer Architecture News 1995, 392-403.
[25] Owens JD, Houston M, Luebke D, Green S, Stone JE, Phillips JC: GPU computing. Proc IEEE 2008, 96(5):879-899.
[26] Nickolls J, Dally WJ: The GPU computing era. IEEE Micro 2010, 30(2):56-69.
[27] Hong S, Kim H: An analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness. ACM SIGARCH Computer Architecture News 2009, 152-163.