MapReduce and Data Intensive Applications

Tak-Lon (Stephen) Wu
Computer Science, School of Informatics and Computing
Indiana University, Bloomington, IN
taklwu@indiana.edu

Abstract
Distributed and parallel computing have emerged as a well developed field in computer science. Cloud computing offers new approaches for commercial and scientific applications because it incorporates different perspectives for the use of both hardware and software. MapReduce-distributed data processing architecture, instantiating the new paradigm of "bring the computation to data" has become a popular solution for large scale scientific applications, such as data/text-mining, bioinformatics sequence alignment, Matrix Multiplication, etc. To understand whether the MapReduce computing model applies to these data-intensive analytic problems, we have explored several problems by analyzing their usage for different MapReduce platforms in HPC-Clouds environments. This paper mainly review the state-of-the-art MapReduce systems for scientific applications. It also summarizes research issues found in prior studies.

I. Introduction

Every day, an enormous amount of terabyte data is generated from scientific experiments, commercial online data collecting, and physical digital device transaction. These collected data are generally analyzed according to different computational algorithms, which yield meaningful results. Therefore, the emergence of scientific computing, especially large-scale data-intensive computing for science discovery, is a growing field of research for helping people analyze how to predict or to explore more possibilities for data analysis. Since tremendous amounts of data are collected, parallel computing options, such as the cloud pleasingly model, have recently been adopted due to its easy-to-use and easy-to-scale features. However, this model is not yet universally accepted. Software developers are always worried about cost, computational models, and comparisons to previous computational models. Typically, the migration from original sequential model to parallel model is the main challenge.

Therefore, in this study, we give a summary of Service Level Agreement (SLA) and cost issues.. The remainder of the paper is organized as follows. Section 2 provides an overview of MapReduce. Section 3 demonstrates several applications that have implemented MapReduce, as well as the experimental results that have been deployed in different environments. Section 4 introduces literature on MapReduce. Concluding remarks are given in Section 5.

II. MapReduce Background

MapReduce is the data centric computing model of cloud computing that provides the computational power to implement and deploy applications within a distributed environment. The MapReduce computation model was initially introduced by Google [1] and aims to solve daily text-based and semantic web data analysis for the data collected by Google services. Hadoop MapReduce [2] was later presented by the open source community and was directed by Yahoo and IBM, with the main differences consisting of the changes from C++ to Java. Thereafter, Hadoop has become MapReduce's main track for academic research, such as [3-6], as well as for "big data" analytic solutions for companies besides Google (e.g., Facebook).

II.1 Architecture

As can be seen in Figure 1, the architecture of MapReduce follows a traditional centralized server-client (master-slaves) model. The master node normally runs the Namenode and

Jobtracker services and takes on most of the administrative work, such as high-level metadata information handling, uploaded data splitting and distribution, jobs and tasks scheduling, and

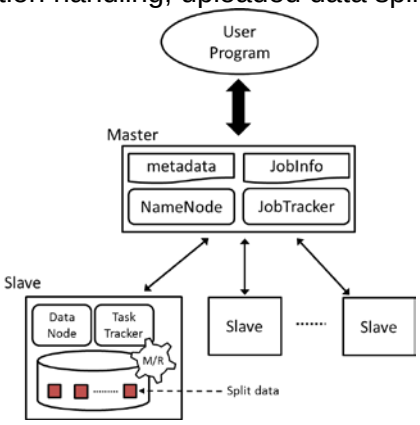


Figure 1. MapReduce Architecture

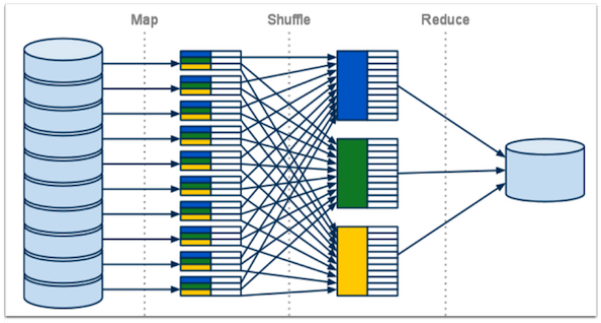


Figure 2. Overview of MapReduce data processing [7]

command-line query handling. On the other hand, other nodes normally run with datanodes and tasktracker services, which store the real split data block and execute the assigned tasks from master node. A general MapReduce job is split into four different stages: Data split, Map, Shuffle, and Reduce.

Input data (files) uploaded to GFS/HDFS are split into sequential blocks with a specified size [google paper and hadoop book], for example, 64 MB is the default block size of Hadoop MapReduce. Then, each block is stored across datanodes according to placement assignment constructed by the master node. Generally, the placement strategy is random and is built based on user-defined replica's factor. For example, if the replica factor is 3, Hadoop places the first split data block on a local datanode within the same rack. Then, it duplicates the same block to another random datanode in another rack. Finally, it forwards the second copy to another random datanode in the same or other rack.

Based on the selected input data format for map stage, the split data blocks are constructed into <key, value> pairs, in which key is normally represented as a unique name and is used to perform in-memory local sorting. Map functions compute the intermediate result according to assigned <key, value> pairs. The Shuffle stage sorts intermediate KeyValue pairs by their keys and sends each record to an assigned reducer. Finally, reducers combine and compute with the collected KeyValue, and yield meaningful results to the disk. Figure 2 demonstrates an example of the overall data processing.

```
map(split input) {
       for each key in input
              emit(key, value)
}

reduce(key, values) {
       sum = 0
       for each value in values
              sum += value
       emit(key, sum)
}
```

Figure 3. MapReduce example program

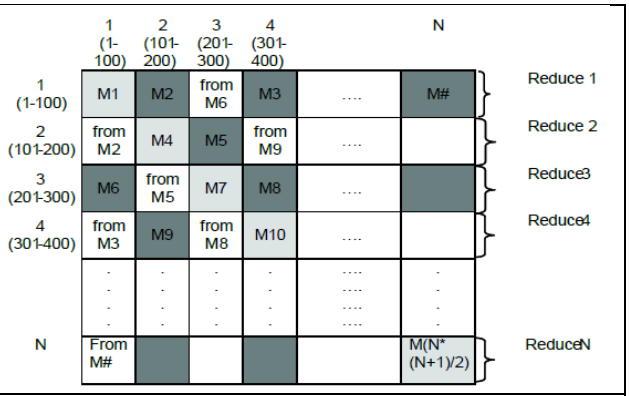


Figure 4. SWG algorithm with MapReduce [8]

II.2 Job scheduling

Data locality is one of MapReduce's key features. For each split data block, the locality information is stored as metadata in Namenode. Jobtracker utilizes this location information to

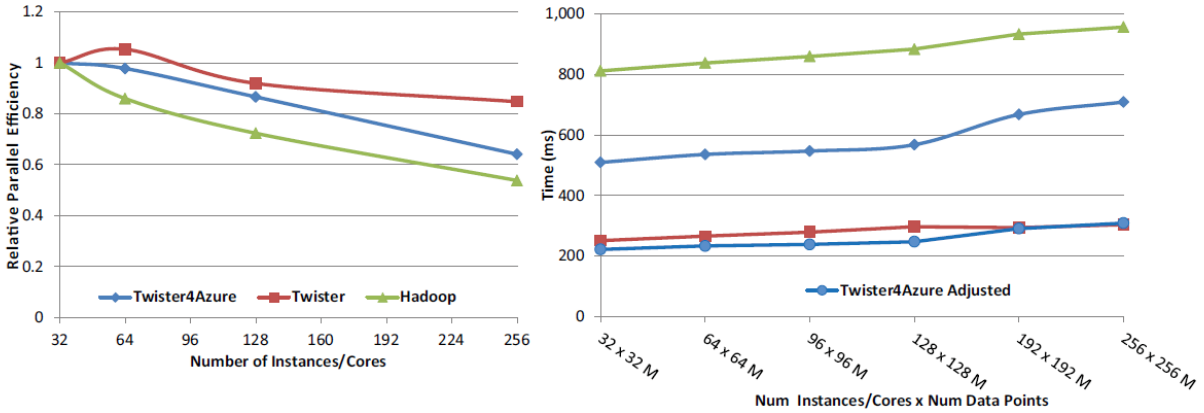


Figure 5. Kmeans clustering relative parallel efficiency using 128 million data points (Left) and weak scaling from 32 to 256 million data points (Right) [9]

assign close to data computation, which reduces the cost from transferring data over the network. Several studies [10] have proved that with this key feature, MapReduce is definitely running faster than random or fairly random task scheduling.

II.3 Classes of MapReduce applications

The idea of MapReduce is that of a simple program on multiple data models (SPMD) in which programmers can easily convert a sequential program using MapReduce API and deploy hundreds or thousands of nodes. Figure 3 offers an example of MapReduce programming style. Here, map function basically takes the split input and generates the intermediate results as the form of <key, value> pairs, then the reduce function aggregates these intermediate <key, values> into results. However, domain scientists may wonder how this function could be used and how the MapReduce style could be integrated into their data problems [11] has categorized the types of applications that MapReduce generally supports, which are: 1) Map-Only, 2) Classic MapReduce, 3) Iterative MapReduce.

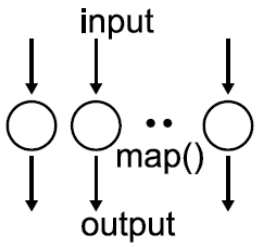
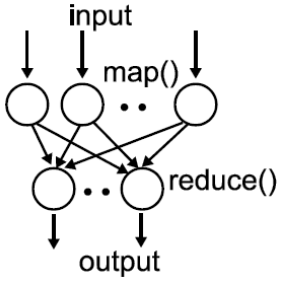
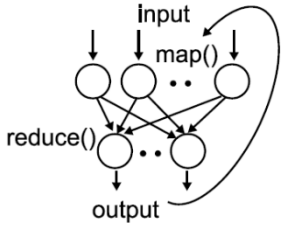
Map-Only	Classic MapReduce	Iterative MapReduce
		

Table 1. Classification of MapReduce Applications

II.3.1 Map-Only

Map-Only, also known as a zero reduce task, is the simplest MapReduce application because it basically converts a collection of data from its original format into other formats. Each map task of a MapReduce program ideally has minimum dependencies and is executed similarly with different split data. MapReduce Cap3 [4] has been implemented in this style.

II.3.2 Classic MapReduce

As mentioned in Section II, a general map stage converts the data input into KeyValue pairs and generates intermediate results. Here, the intermediate data size is decided based on the weight of computation in the map stage. The Reduce stage collects and computes the sorted intermediate results, then, the program aggregates meaningful output according to the user-defined algorithm. WordCount is the classic text-mining problem that finds occurrences of words from a set of text documents and is represented as the standard example of the MapReduce programming model. Generally, a given input document is constructed as <keys, values>, in which keys are the position in the file, and values are the line of text in a file. Map tasks take these initial KeyValue pairs and emit intermediate data in the form of <word, 1>, while reduce tasks aggregate the intermediate KeyValue into <word, occurrences> based on their unique "word" terms. Due to its embarrassingly parallel characteristics, MapReduce has been widely used by commercial and academic communities; for instance, MapReduce SWG [4] has been successfully implemented and deployed on a nationwide scale cluster that runs hundreds of cores.

II.3.3 Iterative MapReduce

Generally, a MapReduce job only contains one round of the map-to-reduce phrase, so the output is then emitted to a distributed file system, such as HDFS. Nevertheless, each job needs a comparably long configuration time to assign map and reduce tasks. For many applications or algorithms, especially data intensive applications, which continuously run within a conditional loop before termination, the output of each round of map-reduce phrase may need to be reused for the next iteration in order to obtain a completed result. Classic MapReduce does not fit with this computing model; therefore, iterative MapReduce is considered a solution for these applications. There are many form of iterative runtimes, such as Twister [12] and HaLoop [13], Twister4Azure [9] (initially AzureMapReduce [4]), and Spark [14], each of which is supported with different features for different optimized applications.

Mainly, these works are similar and extend the classic MapReduce model to support in-job multiple iterations with providing caching for loop-invariant data. For instance, Twister allows a long-running MapReduce job while keeping static/loop-invariant data in memory between iterations; HaLoop extends Hadoop and supports loop-invariant caching and also keeps the reduced outputs in memory in order to provide a faster fix point scheduling; Twister4Azure is an alternative implementation of Twister but deployed to the Microsoft Azure Platform [15], it provides similar loop-invariant data caching either in-memory or on-disk. Moreover, it introduces merge tasks to collect reducer outputs and cache-aware scheduling in order to quickly assign computation regarding cached data; Spark, on the other hand, is built with a high-level language, Scala [16], which is based on resilient distributed datasets [17] (RDDs, read-only data objects across a set of machines). RDDs are used for loop-invariant data caching and reconstruct these data in case of node failure. Spark's programming model and style are similar to query language, but these support MapReduce jobs by exposing the map and reducing tasks to a functional programming interface that is supported by Mesos API [18].

III. Data Intensive applications with MapReduce and results

Scientific computing is a very important task: this type of intensive analytical results are useful for human society's evolution. But the scale of data collection size is rapidly increasing, so traditional sequential computation mechanisms take longer than scientists can bear. The scientific domain has therefore adopted parallel computing such as MPI and MapReduce. Although MPI is also capable of solving the big data problem, it requires a great deal of background knowledge and programming skills [19]. Compared to MPI, MapReduce model provides a user-friendly programming style and has rapidly gained the acceptance of many commercial companies and scientists. In addition, most of the scientific computing algorithms

are data intensive applications which normally run for tens or even hundreds of hours, meaning that failure tasks are common in this regard; this matches the heterogeneous/commodity computing environment, in which software-level or hardware-level fails are common. The MapReduce model does handle fault-tolerance well, as the ability to do this is one of the initial design goals. Finally, either native or optimized data locality scheduling diminishes the data transmission cost of worker nodes.

III.1 Examples for different MapReduce classes

Based on the above benefits of using MapReduce, it has become a growing trend for scientists to utilize MapReduce for daily analysis. In our previous works [4, 9, 20, 21], we have implemented several important bioinformatics and scientific applications, such as SWG [22], Cap3 [23], Kmeans [24], MDS [25] and PageRank [26]. We describe three of them as examples of the above classification.

MapReduce Cap3 is a typical CPU-intensive application for map-only programming mode. Cap3 originally is a DNA sequence assembling and aligning binary written in C. It could be possibly converted into Hadoop Java version. But since the nature of this binary could handle gene sequence independently, [21] implements it as a pleasingly parallel application. The challenge of this type of application is that Cap3 binary must take a file path of a continuous file as program argument, where Hadoop does not have such data input format. Therefore, this study [21] purpose a data input classes `DataFileInputFormat` and `FileRecordReader` for constructing the map's KeyValue pair, where the key the logical filename and the value is the HDFS path of the file. The map task of MapReduce Cap3 takes the sequence assembly binary given with a FASTA-formatted file stored on HDFS, then it generates a partial consensus sequences according to the given file. Those partial output files can be collected by a single reduce task or by a few shell script lines.

Smith Waterman Gotoh (SWG) algorithm [27] is a pairwise dissimilarity comparison of two given DNA or protein sequences. Figure 4 shows the SWG decomposition with Classic MapRedure model. Either Hadoop SWG or Twister SWG calculates the dissimilarity distance by decomposing the given dataset into NxN matrix, when N is amount of DNA or protein sequences. Each map task takes a block matrix of size M, e.g. 100, and computes the SWG distances within that block. Reduce tasks collect the results in the same row and write them into files. In practices, Hadoop SWG splits a large FASTA file into file blocks stored on HDFS, and it constructs <key, values> pairs as <block index, file content> with the support of native SequenceFileInputFormat. Where Twister sends this splits data blocks directly to each map tasks from master node. Also, as the triangular feature of these blocks, only half of the blocks are meaningful and computed. This example shows that any general algorithms could be customized with MapReduce.

Kmeans clustering [24] is a famous data mining algorithm which partitions given data set into disjoint clusters. It is generally implemented in an iterative fashion in which the algorithm returns the result only certain break conditions met, e.g. error rate. Hadoop Kmeans is a workable implementation but slow. The native Hadoop does not support multiple iterations within a MapReduce job, therefore, the only way is to simulate it with submitting multiple MapReduce jobs and sources the output of each iteration as input for next job. In addition, updated centroids of each round are flushed into HDFS and reloaded on all map tasks every time. There are few drawbacks of this implementation – job configuration overhead, lack of caching, and dynamic computation assignment. Twister or HaLoop are considered to be solution for this type of iterative applications. In practice, Twister caches the read-only data points in-memory for each map task which improves I/O performance. Also, with this caching feature, map tasks'

assignment is fixed after the initial iteration. Different from Hadoop Kmeans, updated centroids are broadcasted to all map tasks at the end of each iteration via the messaging broker. The result [9] in Figure 5 shows that Twister Kmeans is significantly better than Hadoop implementation.

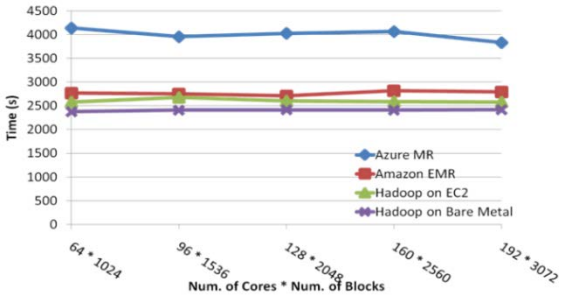


Figure 6. a) SWG performance [4]

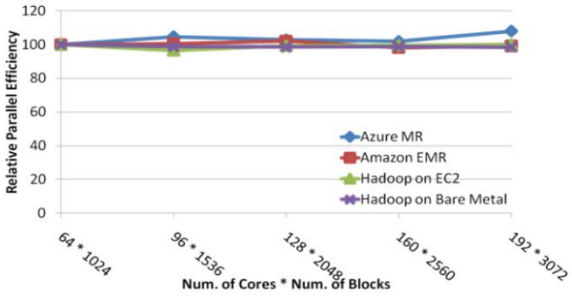


Figure 6. b) SWG relative parallel efficiency [4]

III.2 Experiments and deployments on local cluster and clouds

Above applications have been tested on two different environments: Bare Metal cluster, Amazon EC2 [28] cloud and Azure [15] cloud: 1) Bare Metal tests were performed on an FutureGrid [29] India iDataplex cluster, in which each computer node had two Intel Xeon E5410 2.33GHz CPUs (a total of 8 cores) and 16 GB memory attached to a Gigabit Ethernet network. 2) Amazon EC2 cloud tests were performed using both High CPU extra-large instances and extra-large instances, in which the former has 8 virtual cores (considered 20 EC2 compute units) with 7 GB memory and the latter has 4 virtual cores (considered 8 EC2 compute units) with 15 GB memory. The reason that the cloud tests used two sets of compute instances as SWG applications reached the memory limitation of High CPU extra-large instances. 3) Azure tests were performed using azure small instances, which has one core per computer node.

As seen in Figures 6a and 6b, the SWG application has similar performance results on a different platform, except that the use of the NAligner [30] library in Windows increases total execution time (meanwhile, all other platforms use the JAligner [31] library). We could also prove this issue when running Cap3, as shown in Figures 7a and 7b, in which the overall performance of clouds is nearly same as the bare metal cluster.

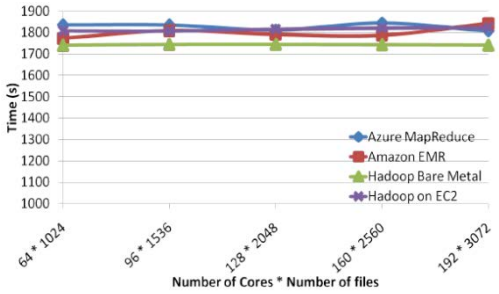


Figure 7. a) Cap3 performance [4]

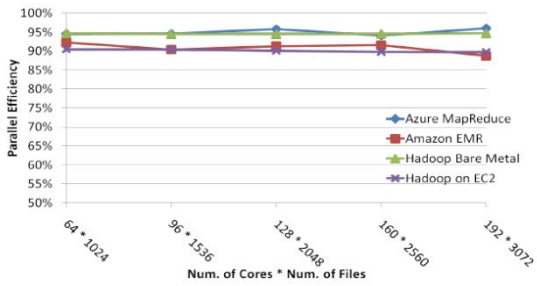


Figure 7. b) Cap3 relative parallel efficiency [4]

III.3 Commercial Cloud Cost

Domain scientists or small companies may not own dedicated clusters; therefore, it is possible to run their application on the commercial clouds. Table 2 shows the instance details of Amazon Cloud and Azure cloud with the basic pricing plan. Currently, there are increasing numbers of new features such as bid-your-own-instances and cluster reservations, but the price range and capacity are uncertain. In addition, Figures 8a and 8b present the total costs of running experiments on clouds. It is acceptable if an individual has a closed deadline without being

assigned to a nationwide cluster. We also captured the daily performance behavior on clouds as being generally normalized without any performance fluctuation issues. The results are shown in Figure 9.

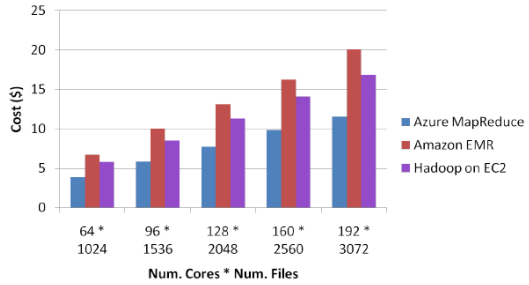


Figure 8. a) Cap3 Cost on Clouds [4]

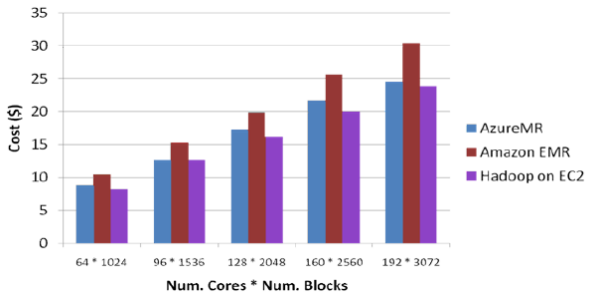


Figure 8. b) SWG costs on clouds [4]

Instance Type (as of 04/20/2013)	Mem. (GB)	Compute units / Virtual cores	Storage (GB)	$ per hours (Linux/Unix)	$ per hours (Windows)
EC2 Small	1.7	1/1	160	0.06	0.091
EC2 Medium	3.75	1/2	410	0.12	0.182
EC2 Large	7.5	4/2	850	0.24	0.364
EC2 Extra Large	15	8/4	1690	0.48	0.728
EC2 High-CPU Extra Large	7	20/2.5	1690	0.58	0.9
EC2 High-Memory Extra Large	68.4	26/3.25	1690	1.64	2.04
Azure Small	1.75	X/1	224+70	0.06	0.09
Azure Medium	3.5	X/2	489+135	0.12	0.18
Azure Large	7	X/4	999+285	0.24	0.36
Azure Extra Large	14	X/8	2039+605	0.48	0.72

Table 2. Amazon EC2 and Azure clouds pricing snapshot (X represents unknown)

IV. Research Issues

As computer scientists, we will always investigate the possibilities of finding effective solutions for various algorithms such that we optimize these applications with different systems, implementations and network communications. Numerous scientists have investigated MapReduce in-depth and they have gotten their own algorithms to run on specified models. Mostly, these related studies can be categorized as MapReduce framework and its extensions. In this study, we introduce these systems in the perspective of data locality.

IV.1 MapReduce framework and its extensions research

MapReduce provides users a simple application interface and executes their single program across a distributed environment. Normally, a user will first put their program inputs into HDFS. A job is then submitted to the master node jobtracker, which then creates tasks based on the data location. As a research angle, tasks scheduling thus becomes a very interesting topic. Zhuhua *et al*. [10] investigated default Hadoop data locality scheduling and observed that the original design does not guarantee optimal data local tasks assignment as it assigned tasks in sequential orders to idle compute slots. It therefore suggests a new LSAP-scheduling, which presents a new cost-evaluated model to generate an overview of unscheduled tasks to instant idle compute slots, which in turn improves the quality of data locality. This model performs better dl-task assignment when the system has more instant idle compute slots.

In addition to scheduling, another interesting study found that the shuffling stage is one of the most significant barriers to the MapReduce job [32]. Originally, in order to avoid interference to reduce tasks, reducers only begin when all results for the map tasks are locally sorted and passed. If a job has a long final map task, the reserved computing resource does not perform efficiently. In order to improve the use of idle resources, Verma *et al*. [32] proposes a modification to bypass the sorting and invokes the reduce function as soon as there has been a partial record emitted by map tasks. In addition to the shift reducer stage in advance, for applications that generate large intermediate data in the shuffling stage, Twister [12] introduces local reduction across tasks on the same compute nodes before sending it to the reducer.

For very large data sets, such as TB-level data, a single commodity cluster may not be able to serve computation efficiently even if it is using MapReduce technology. Yuo *et al*. [33] presents a hierarchical MapReduce, a "Map-Reduce-Global-Reduce" model, and solves this lack of computation resource issue. User-trusted clusters are gathered across the Internet and then conduct a global MapReduce framework, the program inputs are split or pre-allocated to each cluster and map-reduce tasks are assigned by the global controller. At the end of each job, a global reducer collects all output/result from reducers distributed on each cluster and generates the final output. The drawback of this study does not consider the data locality of input, which always moves data to computation. Figure 10 shows the architecture and data flow of this work.

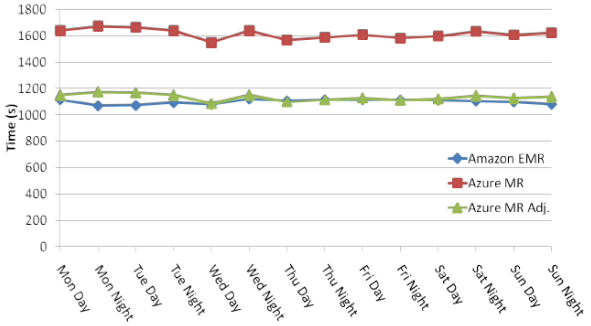


Figure 9. Sustained performance of clouds [4]

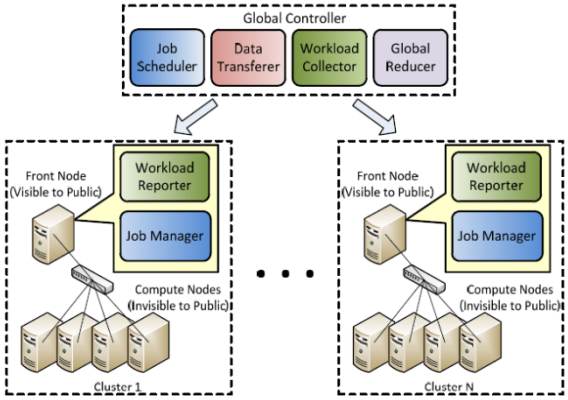


Figure 10. Map-Reduce-Global-Reduce

V. Conclusion

The new era of big data has boosted the growth of the parallel computing community and from this has emerged the need for Big data solutions. The MapReduce framework, originally inspired by Google developers, provides a parallel programming model that solves many large-scale data problems. With its SPMD model, scientists save their energy and focus on their data analysis problems, rather than struggling with parallelism and computing scalability issues. In addition, many companies and research communities have deployed their own Hadoop MapReduce clusters in order to serve increasing needs for big data analysis. To meet different requirements for various perspectives, many MapReduce studies have been released and these are optimized for different targets. However, due to the variety and complexity of data-intensive applications, the MapReduce model generally fit some of applications, but it may not be suitable for multi-tier data mining algorithms with heavy internal data exchange [34].

Reference

[1] J Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters.* Sixth Symposium on Operating Systems Design and Implementation, 2004: p. 137-150.

[2] Apache. *Hadoop MapReduce*, 2010 [accessed 2010 November 6]; Available from: http://hadoop.apache.org/mapreduce/docs/current/index.html.

[3] M. C. Schatz, *CloudBurst: highly sensitive read mapping with MapReduce.* Bioinformatics, 2009. **25**(11): p. 1363-1369. DOI:10.1093/bioinformatics/btp236

[4] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox, *MapReduce in the Clouds for Science*, in *CloudCom 2010*. 2010: IUPUI Conference Center Indianapolis.

[5] Jaliya Ekanayake, Shrideep Pallickara, and Geoffrey Fox. *MapReduce for Data Intensive Scientific Analyses*. in *Fourth IEEE International Conference on eScience*. 2008: IEEE Press.

[6] Qiu X., Ekanayake J., Gunarathne T., Bae S.H., Choi J.Y., Beason S., and Fox G. *Using MapReduce Technologies in Bioinformatics and Medical Informatics*. in *Using Clouds for Parallel Computations in Systems Biology workshop at SC09*. 2009. Portland, Oregon.

[7] *Mike Aizatsky's 2011 Google IO conference presentation*, 2011; Available from: https://developers.google.com/appengine/docs/python/dataprocessing/overview.

[8] Thilina Gunarathne, Tak-Lon Wu, Jong Youl Choi, Seung-Hee Bae, and Judy Qiu, *Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications.* Concurr. Comput. : Pract. Exper. Special Issue, 2010.http://grids.ucs.indiana.edu/ptliupages/publications/ecmls_jour_15.pdf

[9] T. Gunarathne, Zhang Bingjing, Wu Tak-Lon, and J. Qiu. *Portable Parallel Programming on Cloud and HPC: Scientific Applications of Twister4Azure*. in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. 2011.

[10] Zhenhua Guo, Geoffrey Fox, and Mo Zhou, *Investigation of Data Locality in MapReduce*, in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. 2012, IEEE Computer Society. p. 419-426.

[11] Jaliya Ekanayake, *ARCHITECTURE AND PERFORMANCE OF RUNTIME ENVIRONMENTS FOR DATA INTENSIVE SCALABLE COMPUTING*, in *Computer Science*. 2012, Indiana University: Bloomington, USA.

[12] J.Ekanayake, H.Li, B.Zhang, T.Gunarathne, S.Bae, J.Qiu, and G.Fox, *Twister: A Runtime for iterative MapReduce*, in *Proceedings of the First International Workshop on MapReduce and its Applications of ACM HPDC 2010 conference June 20-25, 2010*. 2010, ACM: Chicago, Illinois.

[13] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst, *HaLoop: Efficient Iterative Data Processing on Large Clusters*, in *The 36th International Conference on Very Large Data Bases*. 2010, VLDB Endowment: Singapore.

[14] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. *Spark: cluster computing with working sets*. in *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. 2010. Berkeley, CA, USA: ACM.

[15] *Windows Azure Platform*, Retrieved April 20, 2010, from Microsoft: http://www.microsoft.com/windowsazure/.http://www.microsoft.com/windowsazure/.

[16] *Scala programming language*; Available from: http://www.scala-lang.org.

[17] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica, *Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing*, in *Proceedings of*

the 9th USENIX conference on Networked Systems Design and Implementation. 2012, USENIX Association: San Jose, CA. p. 2-2.

[18] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica, *Mesos: a platform for fine-grained resource sharing in the data center*, in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*. 2011, USENIX Association: Boston, MA. p. 22-22.

[19] Geoffrey Fox, *MPI and MapReduce*, in *Clusters, Clouds, and Grids for Scientific Computing CCGSC*. 2010: Flat Rock NC.

[20] Bingjing Zhang, Yang Ruan, Tak-Lon Wu, Judy Qiu, Adam Hughes, and Geoffrey Fox, *Applying Twister to Scientific Applications*, in *CloudCom 2010*. 2010: IUPUI Conference Center Indianapolis.

[21] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox, *Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications*, in *Proceedings of the Emerging Computational Methods for the Life Sciences Workshop of ACM HPDC 2010 conference*. 2010: Chicago, Illinois.

[22] T. F. Smith and M. S. Waterman, *Identification of common molecular subsequences.* Journal of Molecular Biology, 1981. ***147***(*1*): p. 195-197.

[23] X. Huang and A. Madan, *CAP3: A DNA sequence assembly program.* Genome Res, 1999. **9**(9): p. 868-77.

[24] S. Lloyd, *Least squares quantization in PCM.* Information Theory, IEEE Transactions on, 1982. **28**(2): p. 129-137. DOI:10.1109/tit.1982.1056489

[25] J. B. Kruskal and M. Wish, *Multidimensional Scaling*. 1978: Sage Publications Inc.

[26] Sergey Brin and Lawrence Page, *The anatomy of a large-scale hypertextual Web search engine*, in *Proceedings of the seventh international conference on World Wide Web 7*. 1998, Elsevier Science Publishers B. V.: Brisbane, Australia. p. 107-117.

[27] O. Gotoh, *An improved algorithm for matching biological sequences.* Journal of Molecular Biology, 1982. **162**: p. 705-708.

[28] Amazon. *Amazon Elastic Compute Cloud (Amazon EC2)*, [accessed 2010 November 7]; Available from: http://aws.amazon.com/ec2.

[29] *FutureGrid*; Available from: https://portal.futuregrid.org/.

[30] *NAligner: a C# port of JAligner*; Available from: http://jaligner.sourceforge.net/naligner/.

[31] "JAligner," *Smith Waterman Software*, http://jaligner.sourceforge.net, 2009.

[32] A. Verma, N. Zea, B. Cho, I. Gupta, and R. H. Campbell. *Breaking the MapReduce Stage Barrier*. in *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*. 2010.

[33] Yuan Luo, Zhenhua Guo, Yiming Sun, Beth Plale, Judy Qiu, and Wilfred W. Li, *A hierarchical framework for cross-domain MapReduce execution*, in *Proceedings of the second international workshop on Emerging computational methods for the life sciences*. 2011, ACM: San Jose, California, USA. p. 15-22.

[34] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. *Large Scale Distributed Deep Networks*. in *NIPS 2012: Neural Information Processing Systems,*. 2012. Lake Tahoe, Nevada, United States.