International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878, Impact Factor (2014): 3.05

Implementation of the Map Reduce Paradigm Techniques in Big Data

Sagar J. Pol[1], Rakesh Suryawanshi[2]

[1]PG Scholar, Department of MCA, A.C Patil College of Engineering, Navi Mumbai, India

[2]HOD, Department of MCA, A.C Patil College of Engineering, Navi Mumbai, India

Abstract: ***This MapReduce methodology has been popularized by use at Google, was recently proprietary by Google to be used on clusters and authorized to Apache, and is currently being developed by an intensive community of researchers. MapReduce is a programming model initiated by Google's Team for processing huge datasets in distributed systems; it helps programmers to write programs that process big data. MapReduce is a programming model for processing and generating large data sets. Users specify a map operate that processes a key/value combine to get a group of intermediate key/value pairs and a scale back operate that merges all intermediate values related to constant intermediate key. Using MapReduce programming paradigm the big data is processed. Big data sizes are a constantly moving target currently ranging from a few dozen terabytes to many petabytes of data in a single data set. MapReduce has been seen as one of the key enabling approaches for meeting continuously increasing demands on computing resources imposed by massive data sets. The reason for this is often the high scalability of the MapReduce paradigm that permits for massively parallel and distributed execution over an large number of computing nodes. The volume of data with the speed it is generated makes it difficult for the current computing infrastructure to handle big data. To overcome this drawback, big data processing can be performed through a programming paradigm known as MapReduce. Typical, implementation of the MapReduce paradigm requires networked attached storage and parallel processing. Hadoop and HDFS by apache is widely used for storing and managing big data.***

Keywords: MapReduce, Hadoop, BigData, Google.

1. Introduction

Nowadays, dealing with datasets in the order of terabytes or even petabytes is a reality [3]. MapReduce is a programming model for expressing distributed computations on massive amounts of data and an execution framework for large-scale data processing on clusters [14]. Therefore, processing such big datasets in an efficient way is a clear need for many users [3]. The main reasons of such acceptance are the scalability, failover and ease-of-use properties of Hadoop MapReduce. MapReduce is an abstraction to organize parallelizable tasks [15]. MapReduce Algorithm is been described into two ways:

1) Map: Processing of data.
2) Reduce: Collection and digestion of data.

The MapReduce framework will take care of nodes coordination, data transport etc. [15]. MapReduce is a programming model [14] and a software framework for processing huge data sets in a distributed fashion over a several machines. The core idea behind MapReduce is mapping your data set into a collection of <key, value> pairs, and then reducing overall pairs with the same key [13]. In the Big Data community, MapReduce has been continuously increasing demands on computing resources imposed by massive data sets. Also, MapReduce faces a number of problems when dealing with Big Data including the lack of a high-level language such as SQL, support for iterative ad-hoc data exploration, stream processing and challenges in implementing iterative algorithms [4].

The described MapReduce challenges grouped into four main categories corresponding to Big Data tasks types: data storage, analytics, online processing, security and privacy. An overview of the described challenges is presented in Table I while details of each category are discussed. Additionally, this paper presents current accomplishment aimed at improving and extending MapReduce to address the described challenges [4].

Table 1: An Overview of Mapreduce Challenges

	Main challenges	*Main solution approaches*
Data Storage	Schema-free, index free	In-database MapReduce
		NoSQL stores – MapReduce with various indexing approaches
	Lack of standardized SQL-like language	Apache Hive – SQL on top of Hadoop
		NoSQL stores: proprietary SQL-like languages (Cassandra, Mongo DB) or Hive (HBase)
Analytics	Scaling complex linear algebra	Use computationally less expensive, though less accurate, algebra
	Interactive analysis	Map interactive query processing techniques for handling small data, to MapReduce
	Iterative algorithms	Extensions of MapReduce implementation such as Twister and Hadoop
	Statistical challenges for learning	Data pre-processing using MapReduce
Online processing	Performance / Latency issues	Direct communication between phases and jobs
	Programming model	Alternative models, such as Map Update and Twitter's Storm
Privacy and security	Auditing	Trusted third party monitoring, security analytics
	Access control	Optimized access control approach with semantic understanding
	Privacy	Privacy policy enforcement with security to prevent information leakage

2. Background

Processing large volume of data requires distributing data into thousands of nodes in order to final processing task in short reasonable time. Google's team developed MapReduce to automatically parallelize computation, MapReduce manages data partitioning and distribute it among computation nodes. MapReduce also handles failures in node. MapReduce helps programmers to concentrate in writing a program that process large data, programmers concentrate on the matter in details and MapReduce manage distributed computation problems. Apache Hadoop is an open source implementation of MapReduce. The input of Map function must be represented as key/value pair, for example, the map function that processes a set of document to compute word counts takes a document as a key and the content of the document as a value. Map function produces a set of intermediate key/values. In case of word count the intermediate keys will be individual words and the value will be the number of occurrences of these words. The input of Reduce function is the output of Map function (intermediate key/value pairs). The data taken as a input is managed by Google File System (GFS), Google File System divides input data into a number of blocks; the section block size is specified by users. Google File System replicates each block; the default duplicate number is three. Google File System puts one replicate in the same rack and puts the other clone in other rack. MapReduce runs in cluster of nodes; one node acts as a master node and other nodes act as workers node. Workers nodes are responsible for running map and reduce tasks; the master is responsible for assigning tasks to the idle workers. Each map worker reads the content of its associated split and extracts key/value pairs and passes it to the user defined Map function. The output of Map function is been buffered in memory and partitioned into a set of partitions equals to number of reducers. Master notifies the reduce workers to read the data from local disks of map workers. The result of reduce function is appended to output files. Users may use these files as input to another MapReduce call or use them for another distributed application [5].

The origins of the map function in programming can be traced back to LISP programming language and reduce to APL, the precise specification being dependent on implementation. A detailed study of several different implementations is given in reference. Map-reduce operations are often used in standard imperative languages. Waters studied programs in the IBM Scientific Subroutine Package and found that that 90% of the code could be expressed as maps, filters and accumulations [7]. Map-reduce is used in Google's MapReduce library to utilize large scale clusters for parallelized data processing applications. Programmers simply describe the associated mapreduce computation and a map-reduce library deals with the issues of initialization, configuration, load balancing, networking and fault tolerance. This serves to provide programmers with a simple means to develop applications for a massively parallel machine without the usual associated complications. They showed that a variety of algorithms including locally weighted linear regression, logistic regression, K-means, naïve Bayes, independent component analysis, principal component analysis, support vector machine, Gaussian discriminative analysis, expectation maximization and back propagation can be described and efficiently implemented on multicore and multiprocessor machines. The Brook language for stream computing on GPUs directly supports the efficient compilation of the map and reduces operations [7].

3. Big Data Analytics

Big Data is typically characterized by the so called, 3 "V's" namely; volume, velocity and variety, and when it comes to the volume, the statistics bandied around by Big Data cognoscenti are truly breathtaking [11].

Volume: The increase in data volumes among enterprise systems is caused by dealing volumes and alternative traditional data types, similarly as by new sort of data. Excessive volume is a storage issue, but too much data is also a massive analysis issue [12].

Variety: IT leaders perpetually had a problem translating giant volumes of transactional information into choices — currently there are more types of information to analyze — mainly coming from social media and mobile (context-aware). Selection includes tabular data (databases), hierarchical information, documents, e-mail, metering data, video, still images, audio, stock ticker information, financial transactions and more [12].

Velocity: This involves streams of data, structured record creation, and convenience for access and delivery. Velocity means both how fast information is being produced and how fast the data must be fulfill to meet demand [12].

For example, 15 out of 17 industry sectors in the United States will have more data stored per company than the U.S. Library of Congress, which itself collected 235 terabytes of data in April 2011. Wal-Mart Stores Inc. handles more than 1 million customer transactions every hour, feeding databases estimated at more than 2.5 petabytes, or the equivalent of 167 times the books in the Library of Congress. 30 billion pieces of content are shared on Facebook, monthly. Finally, Intel estimates that there will be 15 billion devices connected to the internet by 2015. Ironically, in many parts of the world, more people have access to a mobile device than to a toilet or running water [11].

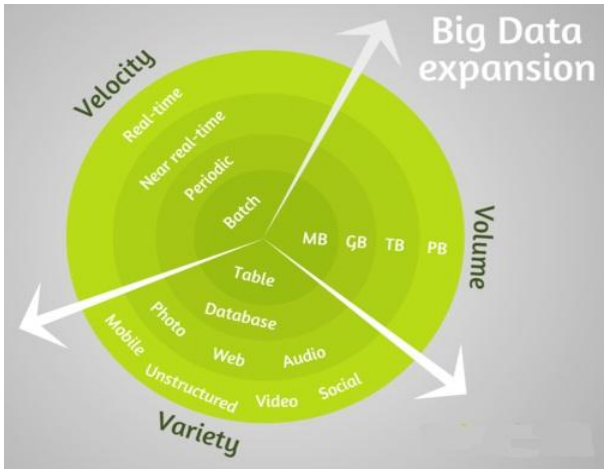


Figure 5: Big Data Expansion

4. Types of Mapreduce

The MapReduce algorithm consists of three steps:-

1. *MAP*: The master node that takes the input, divides the input node into smaller sub issues, and distributes them to the employee nodes. A worker node which may do this again in turn, noted to a multilevel tree structure. The employee node measures the smaller drawback, and passes the solution back to its master node [6].

map(inKey, inValue) -> list(intermediateKey, intermediateValue)
The purpose of the **map** phase is to construct the data in preparation for the alter done in the **reduction** phase. The input to the map function is in the form of key value pairs, even though the data input to a MapReduce program is a file or file(s). By default, the value is a data record and the key is generally the counterbalance of the data record from the beginning of the data file [9].

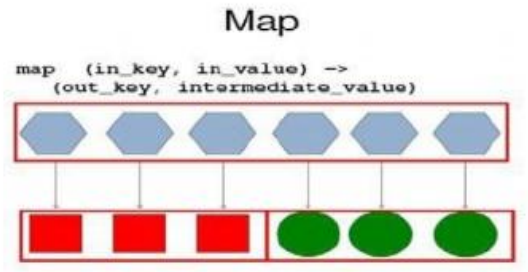


Figure 2: Map Process

2. *SHUFFLE*: Worker nodes readjust data based on the output, such that all data belonging to one key is located on the same worker node.
3. *REDUCE*: The master node then gather the answers to all the sub problems and couple them in some way to form the output – the answer to the problem it was originally trying to solve [6].

Reduce (intermediateKey, list(intermediateValue)) -> list(outKey, outValue)

Each **reduce** function processes the intermediate values for a particular key generated by the **map** function and generates the output. Essentially there exists a one to one mapping between keys and reducers. Multiple reducers can run in parallel, since they are independent of one another. The number of reducers is then decided by the user. By default, the number of reducers is 1 [9].

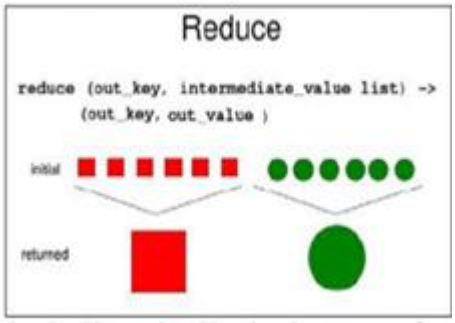
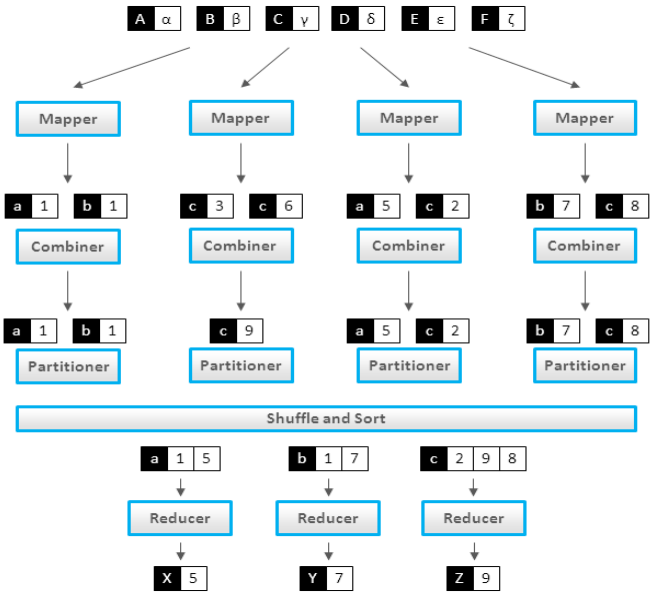


Figure 3: Reduce Process

As the multiple tasks run in parallel, it manages all communications and data transfers between the various parts of the system [6].



MapReduce Framework

5. Dataflow

The frozen part of the MapReduce framework is a large distributed sort. The hot spots, which the application defines, are [8]:

1) Input reader
The *input reader* divides the input into proper size 'splits' (in practice commonly 64 MB to 128 MB) and the framework grants one split to each *Map* function. The *input reader* interprets data from stable storage (typically a **distributed file system**) and generates key/value pairs. A common example will read a directory full of text files and then return each line as a record.

2) Map function
The *Map* function takes a sequence of key/value pairs, processes each, and develops zero or more output key/value pairs. The input and output types of the map can be (and often are) different from each other. If the application is doing a word count, the map function would break the line into words and output a key/value pair for each word. Each output

pair would contain the word as the key and the number of occurrence of that word in the line as the value.

3) ***Partition function***
Each *Map* function output is allocated to a particular *reducer* by the application's *partition* function for sharding purposes. The *partition* function is given the key and the number of reducers and returns the index of the desired *reducer*.
A typical default is to hash the key and use the hash value modulo the number of *reducers*. It is important to pick a partition function that gives an approximately uniform distribution of data per shared for load-balancing purposes, otherwise the MapReduce operation can be held up waiting for slow reducers (reducers assigned more than their share of data) to finish. Between the map and reduce stages, the data is *shuffled* (parallel-sorted / exchanged between nodes) in order to move the data from the map node that produced it to the shard in which it will be reduced. The shuffle can sometimes take longer than the computation time depending on network bandwidth, CPU speeds, data produced and time taken by map and reduce computations.

4) ***Comparison function***
The input for each *Reduce* is pulled from the machine where the *Map* ran and sorted using the application's *comparison* function.

5) ***Reduce function***
The framework calls the application's *Reduce* function once for each unique key in the sorted order. The *Reduce* can iterate through the values that are associated with that key and produce zero or more outputs. In the word count example, the *Reduce* function takes the input values, sums them and generates a single output of the word and the final sum.

6) ***Output writer***
The Output Writer writes the output of the Reduce to the stable storage [8].

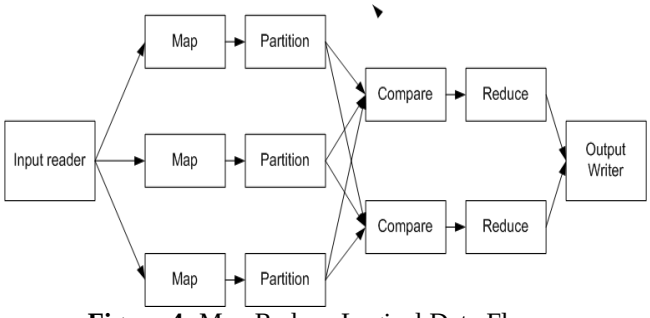


Figure 4: Map Reduce Logical Data Flow

6. Applications of Mapreduce

Several MapReduce applications have been implemented and executed on Google's clusters for example, processing crawled documents, web request logs in order to compute various kinds of derived data, such as inverted indices. In addition, MapReduce has been applied for machine learning task, scientific simulation and large scale image processing tasks. Cho, et al proposed a framework for opinion mining in MapReduce and word map. Opinion Mining is a technique for extracting estimation from the internet. It is known as sentiment classification. It reads a text, analyzes it and produces a result like, and, which is similar to MapReduce data structure. Therefore, it is possible to match it well within MapReduce. Corduroy, et al used MapReduce for clustering very large moderate-to-high dimensionality dataset. Further, Chandar used MapReduce for performing join task. Chandar classifies joins algorithms into two categories with three algorithms for each category as follow: • Two-way joins where joins involving only 2 datasets. The two way join is classified into Map-Side join, Reduce-Side join, and broadcast join. • Multi-way joins where joins involving more than 2 datasets, and it is classified into : Map-Side Join (for multi-way joins), Reduce-Side One-Shot Join, Reduce-Side Cascade Join Chandar found that for two-way join Map-Side joins performs well when the key distribution in the dataset is uniformly distributed. In case of one of the datasets is too small and it can be easily replicated across all machines the broadcast join is the best option [5].

7. Fault Tolerance

MapReduce is designed to deal with hundreds or thousands of assets machines. Therefore, it must tolerate machine failure. The failure may be occur in master node or worker nodes. In case of master failure all MapReduce task will be aborted, and it have to be remake after assigning new master node. On the other hand, to track worker failure, the master monitors all workers checking worker status by periodically. If a worker doesn't respond to master ping in a certain amount of time, the master marks the worker as failed. In case of failure of map task worker; any map tasks either in progress or completed by the worker are reset back to their initial idle state, and will be assigned to other worker. While in case of failure in reduce task worker, any task in progress on a failed worker is assigned to idle worker. The output of completed reduce tasks is stored in global file system, so completed reduce tasks do not need to be re-executed [5].

Since the MapReduce library is designed to help process very large amounts of data using hundreds or thousands of machines, the library must tolerate machine failures gracefully [1].

Worker Failure
The master pings every employee periodically. If no response is received from a employee in a bulk of time, the master marks the worker as failed. Any map tasks which is completed by the worker are been reset back to their initial idle state, and therefore become eligible for scheduling on other workers. Similarly, if any map task or reduce task in progress on a failed worker is also reset to idle and becomes eligible for rescheduling. Completed map tasks are re-executed on a failure because their output is stored on the local disk of the failed machine and is therefore distant. Completed reduce tasks do not need to be re-executed since their output is stored in a global file system. When a map task is executed first by worker A and then later executed by worker B (because A failed), all workers executing reduce tasks are notified of the re-execution. Any reduce task that has not already read the data from worker A will read the data from worker B [1].

Master Failure
It is easy to make the master write periodically checkpoints of the master data structures described above. If the master task dies, from the last check pointed state a new copy can be started. However, given that there is only a single master, its failure is absurd; therefore our current implementation aborts the MapReduce computation if the master fails. Clients can check for this condition and retry the MapReduce operation if they desire [1].

Semantics in the Presence of Failures
When the user-supplied map and the reduce operators are deterministic functions of their input values, our distributed implementation produces the same output as would have been produced by a non-faulting consecutive execution of the whole program. We deem atomic commits of map and reduce task outputs to realize this property. Every in-progress task writes its output to in personal temporary files. When a reduce task completes, the reduce employee atomically renames its temporary output file to the final output file. If the same lower task is executed on multiple machines, multiple rename calls will then be executed for the same final output file [1].

8. What Kind of Problem Does Map Reduce Solve

MapReduce is nice for scaling the process of huge datasets, but it is not designed to be responsive. MapReduce which is a universal term. You most likely mean to ask whether a fully featured MapReduce framework with job control, like Hadoop, is appropriate for you [10].

In the Hadoop implementation, as an example, the overhead of startup sometimes takes a group of minutes alone. The concept here is to take a processing job that would take days and bring it down to the form of hours, or hours to minutes, etc. But you would not start a new job in reply to a web request and expect it to finish in time to respond.

To touch on why this is the case, consider the way MapReduce works:
A bundle of nodes receive parts of input data called splits and do some process (the map step). The intermediate data (output from the last step) is repartitioned specified information with like keys finishes up together. This usually needs some data transfer between nodes. The reduce nodes (which are not necessarily definite from the mapper nodes - a single machine can do multiple jobs in succession) perform the reduce step. Result data is collected and combined to obtain the final output set.
MapReduce is extremely great for preprocessing data such that the web queries can be much faster BECAUSE they don't need to engage in processing [10].

9. Future of Mapreduce

The big data boom has modified a plenty on the Web. Most noticeably, it has stimulated the creation and development of many new technologies, and no one of them has made a bigger impression than the Apache Hadoop open source software framework.

As the Web continues to grow and we develop more ways for users to access it, big data will only get larger. Because of this, Hadoop is one of the most necessary projects out there. According to the "Hadoop-MapReduce Market Forecast 2013-2018," Hadoop MapReduce is expected to rate of grow at a compound annual rate of growth (CAGR) of 58 percent by 2018 that accounts for approximately $2.2 billion. It seems clear that Hadoop is well-positioned to become the industry standard technology for controlling big data and business intelligence solutions, and it presents opportunities for business environments that rely heavily on big data. As they begin to see a growing need to accommodate increasing amounts of data that they have to process, store, and analyze, they additionally see more cost-prohibitive valuation models being imposed by established IT vendors. These problems negatively impact the IT budgets of many corporations, but thanks to the open source nature of Hadoop, it becomes more of a viable and cost-effective solution every day.

10. Security and Privacy

In this section security and privacy concerns for MapReduce and Big Data are discussed. Also, current efforts to locate these issues for MapReduce are presented. Accountability and analyzing are security issues that present a problem for both Big Data and MapReduce. In MapReduce accountability is only provided when the mappers and reducers are held answerable for the tasks they have completed. One result to this issue that has been proposed is the establishment of an Accountable MapReduce.

An additional secured challenge presented to Big Data and MapReduce is that of providing access control, which shown through 3 of Big Data's defining V properties that are volume, variety and velocity. When dealing with a huge volume of information, work performed on that information is likely to require access to multiple storage devices and locations. Therefore, multiple access requirements will be required for any one task. When dealing with data that has a huge variety, semantic understanding of the data should play a role in the access control decision process. Finally, the velocity requirement of MapReduce and Big Data requires that whatever access control approach is used must be advance to determine access control rights in a reasonable amount of time. Privacy is a major topic of concern whenever huge amounts of information are been used. Processes such as data mining and predictive analytics can deduce or discover information linkages. Information linkages are advantageous to organizations, allowing them to get better understanding, target and provide for their clients or users. However, on an individual basis this discovery of information can cause the identities of data providers to be exposed [4].

11. Conclusion

As we have entered into an era of Big Data, processing huge volumes of data has never been bigger. Through better Big Data analysis tools like Map Reduce over

Hadoop guarantees quicker advances in several scientific disciplines and improving the profitability and success of many enterprises [2].

The MapReduce programming model has been successfully used at Google for several different functions. We attribute this success to many reasons. First, the model is easy to use, even for the programmers without experience with parallel and also distributed systems, since it hides the information of parallelization, locality optimization, load balancing and fault-tolerance. Second, a huge variety of issues are easily expressible as MapReduce computations [1].

Issues and challenges MapReduce faces when dealing with Big Data are identified and categorized according to four main Big Data task types- analytics, online processing, data storage, security and privacy. By analyzing MapReduce tasks in Big Data, this paper provides an analysis of the field facilitates better proposal of Big Data projects and identifies opportunities for future research [4].

12. Acknowledgement

I would gratefully and sincerely appreciate my supervisor: Prof. Rakesh Suryawanshi. Their inspiring guidance, rich experience and sustained encouragement enabled me to develop an intensive understanding of my research area. Without the generous help of my supervisor, this work would not have been possible. I am honored to have Prof. Rakesh Suryawanshi from A.C.Patil College as my opponent. I thank him for his kind support and helpful suggestions during the discussions in my MCA.

References

[1] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters.
[2] Dr. Siddaraju1 , Sowmya C L2 , Rashmi K3 , Rahul M4 1Professor & Head of Department of Computer Science & Engineering, 2,3,4Assistant Professor, Department of Computer Science & Engineering. Efficient Analysis of Big Data Using Map Reduce Framework.
[3] Jens Dittrich Jorge-Arnulfo Quiane-Ruiz. Efficient Big Data Processing in Hadoop MapReduce.
[4] Katarina Grolinger Western University, kgroling@uwo.ca Michael Hayes Western University Wilson A. Higashino Western University Alexandra L'Heureux alheure2@uwo.ca David S. Allison Western University. Challenges for MapReduce in Big Data.
[5] Abdelrahman Elsayed, Osama Ismail, and Mohamed E. El-Sharkawi. MapReduce: State-of-the-Art and Research Directions.
[6] Mrigank Mridul, Akashdeep Khajuria, Snehasish Dutta, Kumar N Prasad.M.R Dept of CSE,EWIT,VTU Asst.Prof, CSE Dept, EWIT. Analysis of Bidgata using Apache Hadoop and Map Reduce.
[7] https://en.wikipedia.org/wiki/MapReduce
[8] Paula Ta-Shma IBM Haifa Research Storage Systems. Big Data and Map Reduce.
[9] Shimin Chen, Steven W. Schlosser. Map-Reduce Meets Wider Varieties of Applications.
[10] http://www.websitemagazine.com/content/blogs/posts/archive/2012/08/04/the-future-looks-bright-for-hadoop-mapreduce.aspx
[11] Diana MacLean for CS448G. 2011. A Very Brief Introduction to MapReduce.
[12] Jimmy Lin and Chris Dyer University of Maryland, College Park. Data-Intensive Text Processing with MapReduce.
[13] Processing of massive data: MapReduce
[14] http://ksat.me/map-reduce-a-really-simple-introduction-kloudo/