The Emergence of Modified Hadoop Online-Based MapReduce Technology in Cloud Environments

Shaikh Muhammad Allayear[1](✉), Md. Salahuddin[1], Delwar Hossain[1], and Sung Soon Park[2]

[1] Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh
allayear@ewubd.edu, 2010-2-60-001@ewu.edu.bd, delwarhossain518@live.com
[2] Anyang University, Anyang, South Korea
sspark@anyang.ac.kr

Abstract. The exponential growth of data first presented challenges to cutting-edge businesses such as Goggle, Yahoo, Amazon, Microsoft, Facebook, and Twitter. Data volumes to be processed by cloud applications are growing much faster than computing power. This growth demands new strategies for processing and analyzing information. Hadoop MapReduce has become a powerful computation model that addresses those problems. MapReduce is a programming model that enables easy development of scalable parallel applications to process vast amounts of data on large clusters. Through a simple interface with two functions, map and reduce, this model facilitates parallel implementation of many real world tasks such as data processing for search engines and machine learning. Earlier versions of Hadoop MapReduce had several performance problems like connection between map to reduce task, data overload and slow processing. In this paper, we propose a modified MapReduce architecture – MapReduce Agent (MRA) – that resolves those performance problems. MRA can reduce completion time, improve system utilization, and give better performance. MRA employs multi-connection which resolves error recovery with a Q-chained load balancing system. In this paper, we also discuss various applications and implementations of the MapReduce programming model in cloud environments.

1 Introduction

Cloud computing is the delivery of computing services over the Internet. Cloud services allow individuals and businesses to use software and hardware that are managed by third parties at remote locations. Examples of cloud services include online file

This research (Grants NO. 2013-140-10047118) was supported by the 2013 Industrial Technology Innovation Project Funded by Ministry Of Science, ICT and Future Planning.
The source code for HOP can be downloaded from http://code.google.com/p/hop.

storage, social networking sites, webmail, and online business applications. The cloud computing model allows access information and computer resources from anywhere where a network connection is available. To make full use of big data, tens of terabytes (TBs) or tens of petabytes (PBs) of data need to be handled. To process vast amounts of data Hadoop MapReduce is the basic technologies for big data processing in cloud environments. Google proposed MapReduce. The MapReduce framework simplifies the development of large-scale distributed applications on clusters of commodity machines. MapReduce is typically applied to large batch-oriented computations that are concerned primarily with time to job completion. The Google MapReduce framework [1] and open-source Hadoop system reinforce this usage model through a batch-processing implementation strategy: the entire output of each map and reduce task is materialized to a local file before it can be consumed by the next stage. Materialization allows for a simple and elegant checkpoint/restart fault tolerance mechanism that is critical in large deployments, which have a high probability of slowdowns or failures at worker nodes and traditional MapReduce have some limitation like performance problem, connection problem etc.

To solve the above discussed problems we propose a modified MapReduce architecture that is MapReduce Agent (MRA). MRA provides several important advantages to the MapReduce framework. We highlight the potential benefits first:

- In the MapReduce framework, data is transmitted from the map to the reduce stage. So there may be connection problem. To solve this problem MRA creates iSCSI [2] Multi-Connection and Error Recovery Method [3] to avoid drastic reduction of transmission rate from TCP congestion control mechanism and guarantee fast retransmission of corruptive packet without TCP re-establishment.
- For fault tolerance and workload, MRA creates Q-chained cluster. A Q-chained cluster [3] is able to balance the workload fully among data connections in the event of packet losses due to bad channel characteristics.
- In Cloud computing environment a popular data processing engine for big data is Hadoop MapReduce due to ease-of-use, scalability, and failover properties.

The rest of this paper is organized as follows. Overview of the big data, cloud computing, iSCSI protocol, Hadoop MapReduce architecture and pipelining mechanism [5] are described in Sect. 2. In Sect. 3, we describe our research motivations. We describe our proposed model of MapReduce Agent briefly in Sect. 4. We evaluate the performance and discuss results in Sect. 5. Finally in Sect. 6, we provide the conclusion of this paper.

2 Background

In this section, besides the iSCSI protocol we review the big data implementation in the cloud environment, MapReduce programming model and describe the salient features of Hadoop, a popular open-source implementation of MapReduce.

2.1 Cloud Computing

Cloud computing is an emerging technology for large scale data analysis, providing scalability to thousands of computers, in addition to fault tolerance and cost effectiveness. Generally, a cloud computing environment is a large-scale distributed network system implemented based on a number of servers in data centers. The cloud services are generally classified based on a layer concept. In the upper layers of this paradigm, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) are stacked.

Infrastructure as a Service (IaaS): IaaS is built on top of the data center layer. IaaS enables the provision of storage, hardware, servers and networking components. The client typically pays on a per-use basis. The examples of IaaS are Amazon EC2 (Elastic Cloud Computing) and S3 (Simple Storage Service).

Platform as a Service (PaaS): PaaS offers an advanced integrated environment for building, testing and deploying custom applications. The examples of PaaS are Google App Engine, Microsoft Azure, and Amazon MapReduce/Simple Storage Service.

Software as a Service (SaaS): SaaS supports a software distribution with specific requirements. In this layer, the users can access an application and information remotely via the Internet and pay only for that they use. Salesforce is one of the pioneers in providing this service model. Microsoft's Live Mesh also allows sharing files and folders across multiple devices simultaneously.

2.2 Big Data Management System

Big data includes structured data, semi-structured (XML or HTML tagged text) and unstructured (PDF's, e-mails, and documents) data. Structured data are those data formatted for use in a database management system. Semi-structured and unstructured data include all types of unformatted data including multimedia and social media content. Hadoop, used to process unstructured and semi-structured big data, uses the MapReduce paradigm to locate all relevant data then select only the data directly answering the query. NoSQL, MongoDB, and TerraStore process structured big data.

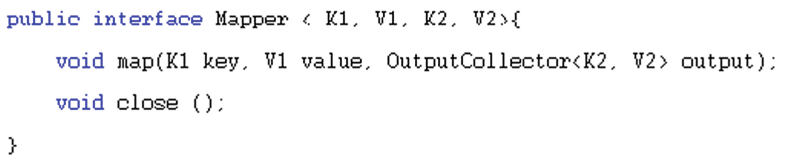
2.3 Hadoop Architecture

Hadoop [4] includes Hadoop MapReduce, an implementation of MapReduce designed for large clusters, and the Hadoop Distributed File System (HDFS), a file system optimized for batch-oriented workloads such as MapReduce. In most Hadoop jobs, HDFS is used to store both the input to the map step and the output of the reduce step. Note that HDFS is not used to store intermediate results (e.g., the output of the map step): these are kept on each node's local file system.

A Hadoop installation consists of a single master node and many worker nodes. The master, called the Job-Tracker, is responsible for accepting jobs from clients, dividing those jobs into tasks, and assigning those tasks to be executed by worker nodes.

2.4 Map Task Execution

Each map task is assigned a portion of the input file called a split. By default, a split contains a single HDFS block (64 MB by default) [4], so the total number of file blocks determines the number of map tasks. The execution of a map task is divided into two phases (Fig. 1).



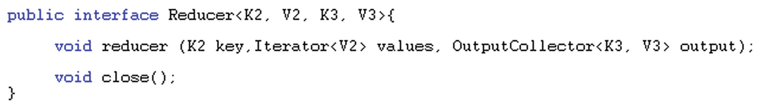
```
public interface Mapper < K1, V1, K2, V2>{

    void map(K1 key, V1 value, OutputCollector<K2, V2> output);

    void close ();

}
```

Fig. 1. Map function interface

- The map phase reads the task's split from HDFS, parses it into records (key/value pairs), and applies the map function to each record.
- After the map function has been applied to each input record, the commit phase registers the final output with the TaskTracker, which then informs the JobTracker that the task has finished executing.

2.5 Reduce Task Execution

The execution of a reduce task is divided into three phases (Fig. 2).



```
public interface Reducer<K2, V2, K3, V3>{
     void reducer (K2 key,Iterator<V2> values, OutputCollector<K3, V3> output);
     void close();
}
```

Fig. 2. Reduce function interface.

- The shuffle phase fetches the reduce task's input data. Each reduce task is assigned a partition of the key range produced by the map step, so the reduce task must fetch the content of this partition from every map task's output.
- The sort phase groups records with the same key together.
- The reduce phase applies the user-defined reduce function to each key and corresponding list of values.

2.6 Pipelining Mechanism

In pipelining version of Hadoop [5], they developed the Hadoop online prototype (HOP) that can be used to support continuous queries: MapReduce jobs that run continuously. They also proposed a technique known as online aggregation which can provide initial estimates of results several orders of magnitude faster than the final results. Finally the pipelining can reduce job completion time by up to 25 % in some scenarios.

2.7 iSCSI Protocol

iSCSI (Internet Small Computer System Interface) is a transport protocol that works on top of TCP [2]. iSCSI transports SCSI packets over TCP/IP. iSCSI client-server model describes clients as iSCSI initiator and data transfer direction is defined with regard to the initiator. Outbound or outgoing transfers are transfer from initiator to the target.

3 Motivations

When a company needs to store and access more data it has multiple choices. One option would be to buy a bigger machine with more CPU, RAM, disk space, etc. This is known as scaling vertically. Of course, there is a limit to the size of single machine that are available and at internet scale this approach is not viable. Another option is cloud computing, here we can store more data. In Cloud computing environment a popular data processing engine for big data is Hadoop MapReduce.

In Cloud computing environment various cloud clients store and process data in a wireless network that has important matter to think about total performance during data sending and receiving. As we know that in wireless network there bandwidth is narrow so during packet exchange time there is huge data overheads. So data sending and receiving parameters needs tuning to be optimized unnecessary packet exchange. Our proposed method is offering to remove unnecessary packet exchange of an iSCSI protocol and to reduce a network overhead.

In the pipelining mechanism of Hadoop MapReduce a naïve implementation is used to send data directly from map to reduce tasks using TCP [5]. When a client submits a new job to Hadoop, the JobTracker assigns the map and reduce tasks associated with the job to the available TaskTracker slots. In the modified Hadoop system each reduce task contacts every map task upon initiation of the job and opens a TCP socket, which will be used to send the output of the map function. The drawback of this solution is that TCP congestion during data transmission cab occur. In this case, TCP connections are being disconnected and after that data must be retransmitted, which takes a long time. To solve this problem, we propose MRA that can send data without retransmission using iSCSI multi-connection and also manages load balancing of data because iSCSI protocol works over TCP. Another motivation is that the iSCSI protocol is based on block I/O and Hadoop's map task also assigns HDFS blocks for the input process.

4 Proposed Model: MapReduce Agent

In cloud computing environments a popular data processing engine for big data is Hadoop-MapReduce due to its ease-of-use, scalability, and failover properties. But traditional MapReduce sometimes has poor performance due to connection problems and slow processing. To resolve those problems and improve the limitations of Hadoop MapReduce, we create MRA, which can improve the performance. Traditional MapReduce implementations also provides a poor interface for interactive data analysis, because they do not emit any output until the map task has been executed to completion. After producing the output of the map function, MRA creates multi-connections with the reducer rapidly (Fig. 3).

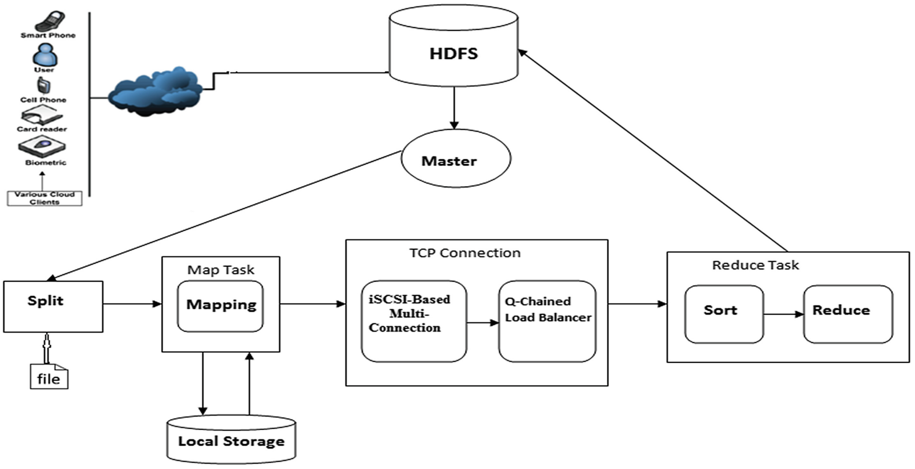


Fig. 3. Proposed MapReduce mechanism to process data in cloud environment.

If one connection falls or a data overload problem occurs then the rest of job will distributed to other connections. Our Q-Chained cluster load balancer does this job [3]. So that the reducer can continue its work, which reduces the job completion time (Fig. 4).

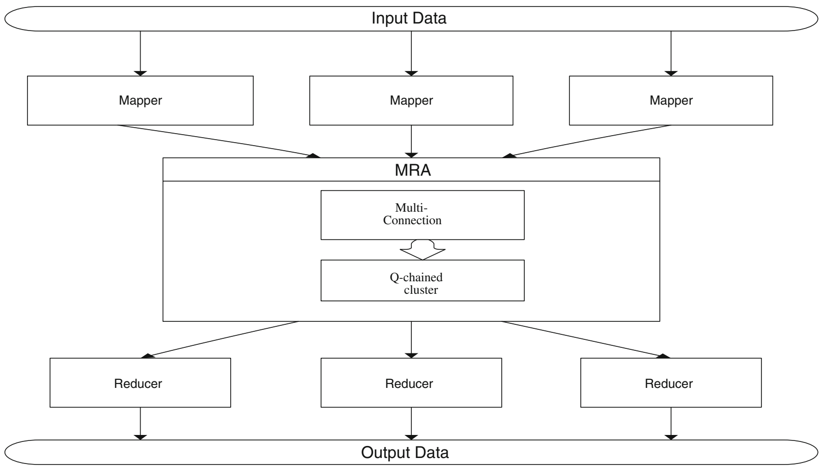


Fig. 4. Map Reduce Agent Architecture (MRA).

4.1 Multi-Connection and Error Recovery Method of ISCSI

In order to alleviate the degradation of the iSCSI-based remote transfer service caused by TCP congestion control, we propose the MRA Multi-Connection and Error Recovery method for one session, which uses multiple connections for each session. As mentioned in [3], in a single TCP network connection when congestion occurs by a timeout or the reception of duplicate ACKs (Acknowledgement) then one half of the current window size is saved in sstresh (slow start window). Additionally, if the congestion is indicated by a timeout, cwnd (congestion window) is set to one segment. This may cause a significant degradation in online MapReduce performance. On the other hand in the Multi-Connection case, if TCP congestion occurs within a connection, the takeover mechanism selects another TCP connection.

The general overview of the proposed Multi-Connection and Error Recovery based on iSCSI protocol scheme, which has been designed for iSCSI based transfer system. When the mapper (worker) is in active mode or connected mode for reduce job that time session is started. This session is indicated to be a collection of multiple TCP connection. If packet losses occur due to bad channel characteristics in any connection, our proposed scheme will pick out Q-Chained Cluster's balanced redistribute data by the other active connections.

We use a filter mechanism to control parameter. Figure 5 shows the parameter filter module, which checks the network status, and calculates the channel number, which is best suited for network resource. The filter also filters the parameter of iSCSI initiator and target. In iSCSI remote storage systems there are also device commands and iSCSI commands. The filter module checks the commands and network status of both initiator and target. If the command parameter carries the device command then it sends to the iSCSI device directly and if the command parameter is iSCSI command related like NOP IN, NOP OUT [7] then it does not need to send them to the device controller.

This way we can reduce the network overhead and increase the iSCSI remote storage system performance. The parameter controller measures the Round-Trip Time (RTT) in TCP two-way handshake to determine the appropriate number of TCP connections for a specific destination.

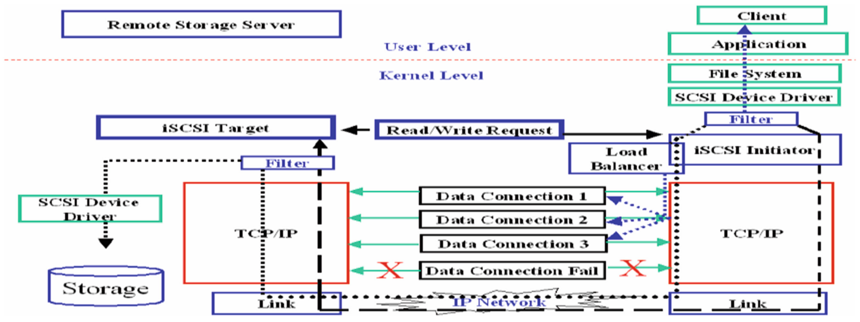


Fig. 5. Overview of Multi-connection and Error Recovery Method of iSCSI [3].

4.1.1 TCP Two Way Handshaking

With RTT we can measure TCP two-way handshaking between the initiator and the target, it can be more efficient at avoiding filtering and inflation of packets than ICMP probes. The multi-TCP connections controller negotiates the number of connections between the initiator and the target for data transmission, according to Eq. (2) using parameter (RTT), which was collected by the parameter collector.

As given: p is a packet drop rate. T bps: the maximum sending rate for a TCP connection. B (bytes): TCP connection sending packets with a fairly constant RTT of R seconds. Given the packet drop rate p, the minimum Round-trip time R, and the maximum packet size B, we can use the Eq. (1) to calculate the maximum arrival rate from a conformant TCP connection.

$$T \leq \frac{1.5 * \sqrt{2/3 * B}}{R * \sqrt{p}} \tag{1}$$

Equation (2) shows that the number of established TCP connections (N) used in Multi-Connection iSCSI depends on RTT (Rt) measured by the parameter collector. The minimum RTT can determine the number of connections to be opened as shown in Fig. 6.

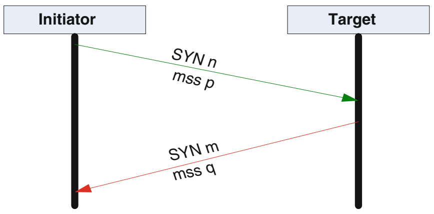


Fig. 6. Two-way handshaking of TCP Connection.

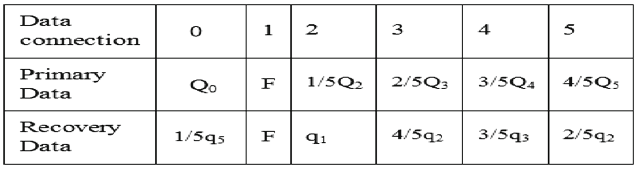
However, while the use of concurrent connections increases throughput, it also increases the packet drop rate. Therefore, it is important to obtain the optimal number of connections to produce the expected throughput.

$$T \leq \frac{1.5 * \sqrt{2/3 * B}}{R * \sqrt{p}} \leq \frac{N * W}{Rt} \tag{2}$$

Where W is window size of each TCP connection. The received acknowledgement from the initiator and the sent acknowledgement from target will be integrated with the next request and the next acknowledgement response.

4.2 Q-Chained Cluster Load Balancer

Q-chained cluster is able to balance the workload fully among data connections in the event of packet losses due to bad channel characteristics. When a congestion occurs in a data connection, this module can do a better job of balancing the workload, which is originated by congestion connection. It will be distributed among N-1 connections instead of a single data connection. Figure 7 illustrates how the workload is balanced in the event of congestion occurrence in a data connection (Data Connection 1 in this example) with Q-chained cluster. For example, with the congestion occurre nce of Data Connection 1, the primary data Q1 is no longer transmitted in the congested connection for the TCP input rate to be throttled and thus its recovery data Q1 of Data Connection 1 is passed to Data Connection 2 for conveying storage data. However, instead of requiring Data Connection 2 to process all data both Q2 and Q1, Q-chained cluster offloads 4/5ths of the transmission of Q2 by redirecting them to Q2 in Data Connection 3. In turn, 3/5ths of the transmission of Q3 in Data Connection 3 are sent to Q3. This dynamic reassignment of the workload results in an increase of 1/5th in the workload of each remaining data connection.



Data connection	0	1	2	3	4	5
Primary Data	Q_0	F	$1/5Q_2$	$2/5Q_3$	$3/5Q_4$	$4/5Q_5$
Recovery Data	$1/5q_5$	F	q_1	$4/5q_2$	$3/5q_3$	$2/5q_2$

Fig. 7. Q-Chained load balancer.

5 Performance Evaluation

As per as [5] we also evaluate the effectiveness of online aggregation, we performed two experiments on Amazon EC2 using different data sets and query workloads. In their first experiment [5], the authors wrote a “Top-K” query using two MapReduce jobs: the first job counts the frequency of each word and the second job selects the K most frequent words. We ran this workload on 5.5 GB of Wikipedia article text stored in HDFS, using a 128 MB block size. We used a 60-node EC2 cluster; each node was a “high-CPU medium” EC2 instance with 1.7 GB of RAM and 2 virtual cores. A virtual core is the equivalent of a 2007-era 2.5 Ghz Intel Xeon processor. A single EC2 node executed the Hadoop Job- Tracker and the HDFS NameNode, while the remaining nodes served as slaves for running the TaskTrackers and HDFS DataNodes.

A thorough performance comparison between pipelining, blocking and MRA is beyond the scope of this paper. In this section, we instead demonstrate that MRA can reduce job completion times in some configurations. We report performance using both large (512 MB) and small (32 MB) HDFS block sizes using a single workload (a wordcount job over randomly-generated text). Since the words were generated using a uniform distribution, map-side combiners were ineffective for this workload. We performed all experiments using relatively small clusters of Amazon EC2 nodes. We also did not consider performance in an environment where multiple concurrent jobs are executing simultaneously.

5.1 Performance Results of ISCSI Protocol for Multi-Connection

Experimental Methodology:

Our scheme's throughput in different RTTs are measured for different numbers of connections in Fig. 8. We see the slowness of the rising rate of throughput between 8 connections and 9 connections. This shows that reconstructing the data in turn influences throughputs and the packet drop rates are increased when the number of TCP connections is 9 as the maximum use of concurrent connections between initiator and target.

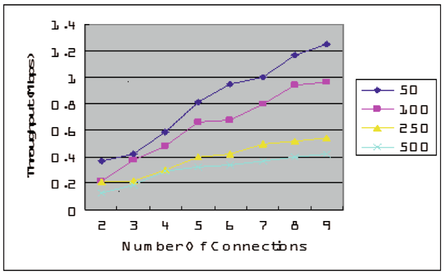


Fig. 8. Throughput of Multi-Connection iSCSI System. Y axis is containing throughput easurement with Mbps & X axis is for number of connections. 50, 100, 250 and 500 RTT are measured by ms.

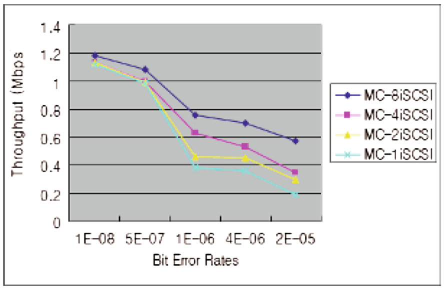


Fig. 9. Throughput of Multi-Connection iSCSI vs iSCSI at different error rates. Y axis is throughput & X axis is for bit error rate.

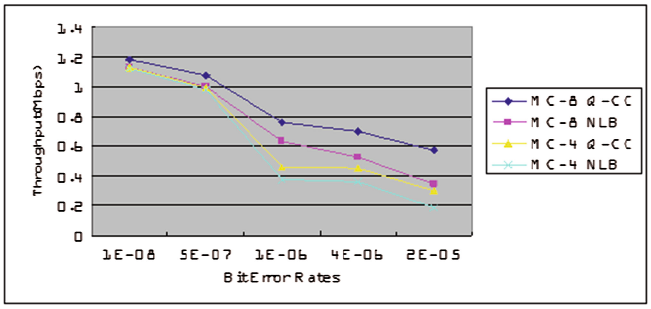


Fig. 10. Q-Chained cluster load balancer vs No load balancer. MC: Multi Connection, Q-CC: Q-Chained cluster NLB: No load balancer.

Therefore, 8 is the maximum optimal number of connections from a performance point of view. Multi-Connection iSCSI mechanism also works effectively because the data transfer throughputs increase linearly when the round trip time is larger than 250 ms.

In Fig. 9, the performance comparison of Multi-Connection iSCSI and iSCSI at different bit-error rates is shown. We see that for bit-error rates of over 5.0 × 10-7the Multi-Connection iSCSI (2 connections) performs significantly better than the iSCSI (1 connection), achieving a throughput improvement about 24 % in SCSI read.

Moreover, as bit-error rates go up, the figure shows that the rising rate of throughput is getting higher at 33 % in 1.0 × 10-6, 39.3 % in 3.9 × 10-6and 44 % in 1.5 × 10-5. Actually, Multi-Connection iSCSI can avoid the forceful reduction of transmission rate efficiently from TCP congestion control using another TCP connection opened during a service session, while iSCSI does not make any progress. Under statuses of low bit error rates (< 5.0 × 10-7), we see little difference between Multi-Connection iSCSI and iSCSI. At such low bit errors iSCSI is quite robust at handling these.

In Fig. 10, Multi-Connection iSCSI (8 connections) with Q-Chained cluster shows average performance improvement of about 11.5 %. It can distribute the workload among all remaining connections when packet losses occur in any connection. To recall an example given earlier, with M = 6, when congestion occurs in a specific connection, the workload of each connection increases by only 1/5. However, if Multi-Connection iSCSI (proposed scheme) establishes a performance baseline without load balancing, any connection, which is randomly selected from takeover mechanism, is overwhelmed.

5.2 Experimental Results of TCP Two Way Handshaking

Comparing two way handshaking method with three way handshaking, we have achieved better performance which is shown in Fig. 11.

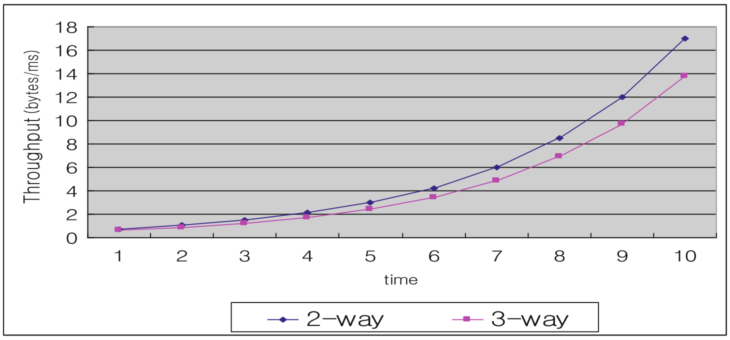


Fig. 11. Comparison of data transmission, in between two-way handshaking and three-way handshaking mode.

5.3 Performance Results and Comparison on MapReduce

In the Hadoop map reduce architecture [4, 5]; their first task is to generate output which is done by map task consume the output by reduce task. The whole thing makes the process lengthy because reduce tasks have to wait for the output of the map tasks. Using pipelining mechanism [5], they send output of map task immediately after generation of per output to the reduce task so it takes less time than Hadoop MapReduce. During the transmission (TCP) if any problem occurred then they retransmit again which takes more time and drastically reduces the performance of the MapReduce mechanism (Figs. 12, 13, 14, 15, 16, 17).

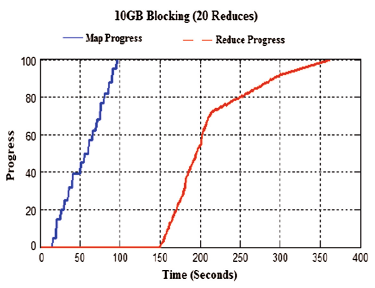


Fig. 12. CDF of map and reduce task completion times for a 10 GB wordcount job using 20 map tasks and 20 reduce tasks (512 MB block size). The total job runtimes were 361 s for blocking.

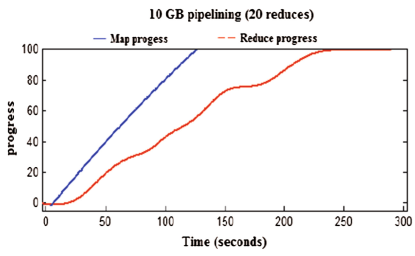


Fig. 13. CDF of map and reduce task completion times for a 10 GB wordcount job using 20 map tasks and 20 reduce tasks (512 MB block size). The total job runtimes were 290 s for pipelining.

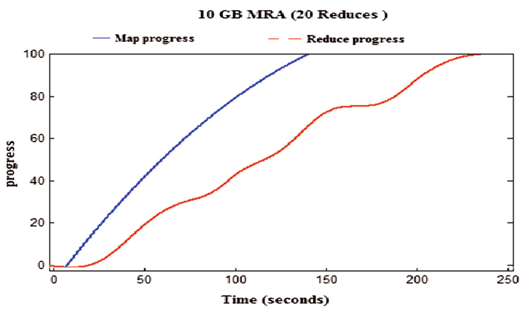


Fig. 14. CDF of map and reduce task completion times for a 10 GB wordcount job using 20 map tasks and 20 reduce tasks (512 MB block size). The total job runtimes were 240 s for MRA.

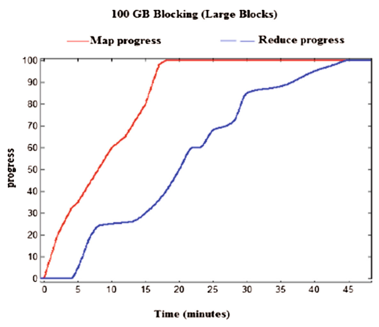


Fig. 15. CDF of map and reduce task completion times for a 100 GB wordcount job using 240 map tasks and 60 reduce tasks (512 MB block size). The total job runtimes were 48 min for blocking.

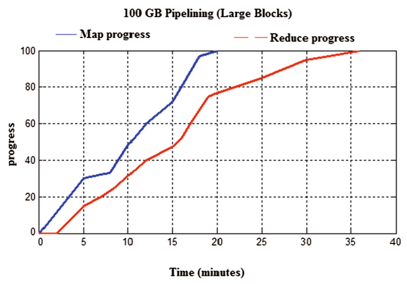


Fig. 16. CDF of map and reduce task completion times for a 100 GB wordcount job using 240 map tasks and 60 reduce tasks (512 MB block size). The total job runtimes were 36 min for pipelining.

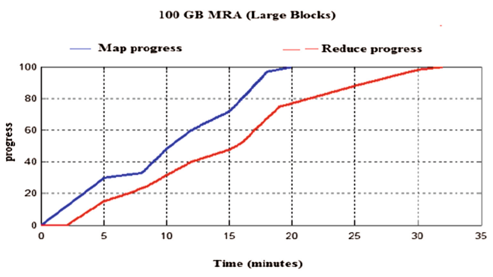


Fig. 17. CDF of map and reduce task completion times for a 100 GB wordcount job using 240 map tasks and 60 reduce tasks (512 MB block size). The total job runtimes were 32 min for MRA.

On the other hand our proposed mechanism (MRA) recovers the drawback by using multi-connection and Q-chained load balancer method. In these circumstances MRA may prove its better time of completion.

6 Conclusion

Cloud technology progress & increased use of the Internet are creating very large new datasets with increasing value to businesses and processing power to analyze them affordable. The Hadoop-MapReduce programming paradigm has a substantial base in the big data community due to the cost-effectiveness on commodity Linux clusters and in the cloud via data upload to cloud vendors, who have implemented Hadoop/HBase.

Finally we can say that our proposed model MRA resolves all the limitation of Hadoop Map Reduce and it can reduce the time to job completion. Our modified MapReduce architecture that can play an important role to process big data in cloud environment efficiently.

References

1. Dean, J., Ghemawat, S.: MapReduce: Simplified dataprocessing on large clusters. In: OSDI (2004)
2. SAM-3 Information Technology – SCSI Architecture Model 3, Working Draft, T10 Project 1561-D, Revision7 (2003)
3. Allayear, S.M., Park, S.S.: iSCSI multi-connection and error recovery method for remote storage system in mobile appliance. In: Gavrilova, M.L., Gervasi, O., Kumar, V., Tan, C., Taniar, D., Laganá, A., Mun, Y., Choo, H. (eds.) ICCSA 2006. LNCS, vol. 3981, pp. 641–650. Springer, Heidelberg (2006)
4. Hadoop. http://hadoop.apache.org/mapreduce/
5. Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M.: UC Berkeley: MapReduce Online. Khaled Elmeleegy, Russell Sears (Yahoo! Research)

6. Allayear, S.M., Park, S.S.: iSCSI protocol adaptation with NAS system via wireless environment. In: International Conference on Consumer Electronics (ICCE), Las Vegus, USA (2008)
7. RFC 3270. http://www.ietf.org/rfc/rfc3720.txt
8. Daneshyar, S., Razmjoo, M.: Large-Scale Data Processing Using Mapreduce in Cloud Computing Environment
9. Changqing Ji∗†, Yu Li‡, Wenming Qiu‡, Uchechukwu Awada‡, Keqiu Li‡ : Big Data Processing in Cloud Computing Environments
10. Rabi Prasad Padhy: Big Data Processing with Hadoop-MapReduce in Cloud Systems
11. Chan, J.O.: An Architecture for Big Data Analytics
12. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online aggregation. In: SIGMOD (1997)
13. Caceres, R., Iftode, L.: Improving the performance of reliable transport protocols inMobile computing environments. IEEE JSAC
14. Laurila, J.K., Blom, J., Dousse, O., Gatica-Perez, D.: The Mobile Data Challenge: Big Data for Mobile Computing Research
15. Satyanarayanan, M.: Mobile computing: the next decade. In: Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond (MCS) June 2010
16. Verma, A., Zea, N., Cho, B., Gupta, I., Campbell, R.H.: Breaking the MapReduce Stage Barrier*
17. Stokely, M.: Histogram tools for distributions of large data sets
18. Lu, L., Shi, X., Jin, H., Wang, Q., Yuan, D., Wu, S.: Morpho: A decoupled MapReduce framework for elastic cloud computing
19. Hao, C., Ying, Q.: Research of Cloud Computing based on the Hadoop platform. Chengdu, China, pp. 181–184, 21-23 October 2011
20. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: a Berkeley View of Cloud Computing, Tech. Rep., University of California at Berkeley (2009)
21. Palanisamy, B., Singh, A., Liu, L., Jain, B.,: Purlieus: locality-aware resource allocation for MapReduce in a cloud. In: Proceedings of the ACM/IEEE Conference on High Performance Computing Networking, Storage and Analysis, SC 2011, Seattle, WA, USA (2011)
22. Lu, L., Jin, H., Shi, X., Fedak, G.: Assessing MapReduce for internet computing: a comparison of Hadoop and BitDew-MapReduce. In: Proceedings of the 13th ACM/IEEE International