Journal of Nonlinear Analysis and Optimization
Vol. 10(3) (2019), March 2019
https://ph03.tci-thaijjo.org/
ISSN : 1906-9685

J.Nonlinear
Anal. Optim

Simulation Of Image Retrieval Techniques For Effective Decision Making System

DR.P HUNDEEP, Associate Professor, Department of CSE DHRUVA INSTITUTE OF ENGINEERING & TECHNOLOGY, HYDERABAD, hundeepsurya11@gmail.com.
DR.G SRIVIDYA, Associate Professor, Department of CSE ELLENKI COLLEGE OF ENGINEERING & TECHNOLOGY, HYDERABAD, Veerabhadrasrividya@gmail.com.
DR.T SUNEEL, Professor, Department of CSE DHRUVA INSTITUTE OF ENGINEERING & TECHNOLOGY, HYDERABAD, suneelkumarvarma2013@gmail.com.

ABSTRACT

Map-reduce is a programming model used for processing data intensive applications. More than ten thousand distinct Map-reduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand Map-reduce jobs are executed on Google's clusters every day, processing a total of more than twenty petabytes of data per day. **Divisible Load Theory**(DLT) is applied to the existing Map-reduce system to increase the efficiency of the system. In this work, a new fault tolerant Map-reduce system is developed which is applicable to static type of scheduling(DLT). This paper proposes a new algorithm –**Two Level Fault Tolerant Partitioning** (TFTP) which identifies the faulty processor and re-executes the data by scheduling it to the straggler processors. This algorithm mainly ensures the completion of the job at the nearest estimated time. And Checkpointing along with TFTP reduces the redundancy in the system by re-executing the jobs from a saved state rather than re-executing it from the beginning. There is some overhead in checkpointing the data. By having a tradeoff between the Fault Probability and checkpointing interval, the efficiency of the system is improved by the proposed method. .

Keywords: Map-Reduce, Divisible Load Theory, Fault Tolerance, Straggler, Checkpoint.

1 INTRODUCTION

Map-reduce is a programming paradigm for data intensive applications over distributed environment. User submits the data and defines its computation as two functions, namely, Map and Reduce. Map function is typically executed by a number of map tasks, each of which, operate simultaneously over a split of the user input data, to generate a set of intermediate key-value pairs. The output lists of every individual mapper are hashed into a R space domain. Splits of these intermediate results are then read as input by the corresponding reduce tasks which perform the user defined Reduce function. Each reducer in the system takes the output list from the corresponding hash space and reduces them into a final list of output values. Hence this framework allows parallel processing over independent partitions of the input data by every Map task and Reduce task during the Map phase and Reduce phase respectively. This functional model with map and reduce operations allows us to parallelize large computations easily. The input data is split up into equal partitions and submitted to the mappers. Since each mapper has different efficiency some of the mappers finishes its job and waits for the other mappers to complete their execution which increases the idle time of the whole system. To increase the efficiency of the system, the split of input data should vary. Divisible Load Theory (DLT) is used to split the data based on the capacity and efficiency of the mappers.The data is splitted up such that all the mappers finishes their job at the same time.

Our work mainly concerns about the Fault Tolerance in a map-reduce system with Divisible Load Theory. A Fault is generally a system failure due to overloads, power failure, and data loss etc. The occurrence of fault generally results in either incompletion of the job or delay in completion of the job. Our work proposes a novel algorithm which ensures the completion of the job at the estimated time. The slow performing nodes are taken as backup nodes and are put up in stragglerpool(backup pool).

Whenever a node becomes faulty the data given to that node is assigned to various nodes taken from straggler pool. The number of nodes is increased to finish the job at the estimated time thus ensuring the completion of the job very closer to the deadline.

2 RELATED WORKS

The history of research and development in the field of Map-reduce systems started with the development of simplified data processing over large clusters[1], through the early adoption of efficient and fast search technique in Google search engine. Map-reduce was introduced by Google in 2004.It was later implemented as 'Cloud MapReduce' on top of Amazon Cloud OS by Accenture Technology Labs. An open source project supporting Map-reduce framework, by name 'HadoopMapReduce' was also developed by the Apache software foundation. Map-reduce could be applied for different kind of jobs. A few examples are Word Count, Distributed Grep, Count of URL Access Frequency, Reverse Web-Link Graph, Inverted Index and Distributed Sort[1]. The execution overview of Map-reduce system is that the Job(v) is splitted and submitted to the mappers as input key-value pairs.The mappers produce the intermediate results as intermediate key-value pairs which are then hashed into R space domain of reducers. The Reducers then take the intermediate key value pairs and produces the actual result [1]. Divisible Load Theory (DLT) is a framework for partitioning the divisible load into independent chunks for processing by homogeneous nodes. The partitioning ensures fair splitting of the input data in order to optimize the schedule length. The concept has been applied in linear algebra, image processing, video and multimedia broadcasting, database searching and in processing of large distributed files [2]. Azure Map-reduce is runtime architecture for Map-reduce clusters and the challenges in it includes data storage, consistency and scalability [3].With further advancements Map-reduce is implemented on top of a cloud operating system with minimal lines of code and increased efficiency, scalability, and speed [4]. The effect of several component inter-connect topologies, data locality, and software and hardware failures on overall Map-reduce application performance is explored in [5] .The results of using the proposed simulator indicate that network topology choices and scheduling decisions can have a large impact on performance. The simulator also helps in designing new high performance Map-reduce setups and in optimizing existing ones. The Divisible Load Theory models are categorized into different scenarios and present the mathematical model to each scenario in [6]. Some of the classifications discussed in this work are as follows. The application to which DLT is applied could be either star graph or tree graph model. Based on the number of rounds DLT is used, the implementation could be of one round or multi round type. If initialization cost is involved, as is the practical case, the model is termed to be using affine cost, else it is said to be of linear cost. Top ten reasons to use Divisible Load Theoryare listed in [7]. The work is a premiere enlightening the advantages of DLT. Few of the areas discussed include scalability, interconnection topologies and DLT's optimality principle. In the later Map-reduce era, many scheduling algorithms have been proposed. A decentralized algorithm performs intra cluster and inter cluster (grid) job scheduling are proposed in [8]. In their work, DLT is applied along with Least Cost. A detailed research on real time divisible load scheduling with set up costs and advance reservationsare discussed in [9].A novel algorithm is also presented for real time divisible scheduling based on feedback control and admission control. Various Fault Tolerant strategies are proposed in the Map-reduce system to increase the consistency of the system and various techniques like checkpointing, Replication etc. Various Checkpointing strategies like mean failure checkpointing, standard deviation based checkpointing are proposed in [10]. Combined fault tolerance is yet another efficient strategy in which a tradeoff should be made between checkpointing and replication. Passive replication is an improved fault tolerant strategy for the mapreducesystem and is discussed in [11].The heuristics to schedule backups, move backup instances, and select backups upon failure for fast recovery is also explored in [11]. Straggler nodes have been clearly defined which contributes a lot to increase the efficiency of the system is presented in [12]. A framework to analyze the tradeoff between communication costs and decoding accuracy is also discussed. The various trends in check pointing the data such as coordinated check pointing, Uncoordinated or independent check pointing are discussed in [13].

3 OVERALL ARCHITECTURE

If a mapper or reducer becomes faulty, which may leadto double the time to finish the job and the cost is also doubled. A general fault tolerant system only ensures that the job will be finished with the degraded performance. It does not give any assurances about the deadline. To overcome this we propose a Two Level Fault Tolerant Partitioning(TFTP) algorithm. The Architecture mainly consists of Fault Tolerance Daemon (FTD) which monitors all the nodes in the system

continuously and detects faulty nodes if any. The job given to that faulty node is identified and is given to the set of nodes from straggler pool to ensure the completion of that job in theestimatedtime.The proposed architecture is shown in Fig.1. The proposed system mainly contains two major systems such as Fault Tolerance system and map reduce system, which mainly consists of Fault Tolerance Daemon (FTD) and scheduling module. The functionality of each module is discussed in the following paragraphs briefly.

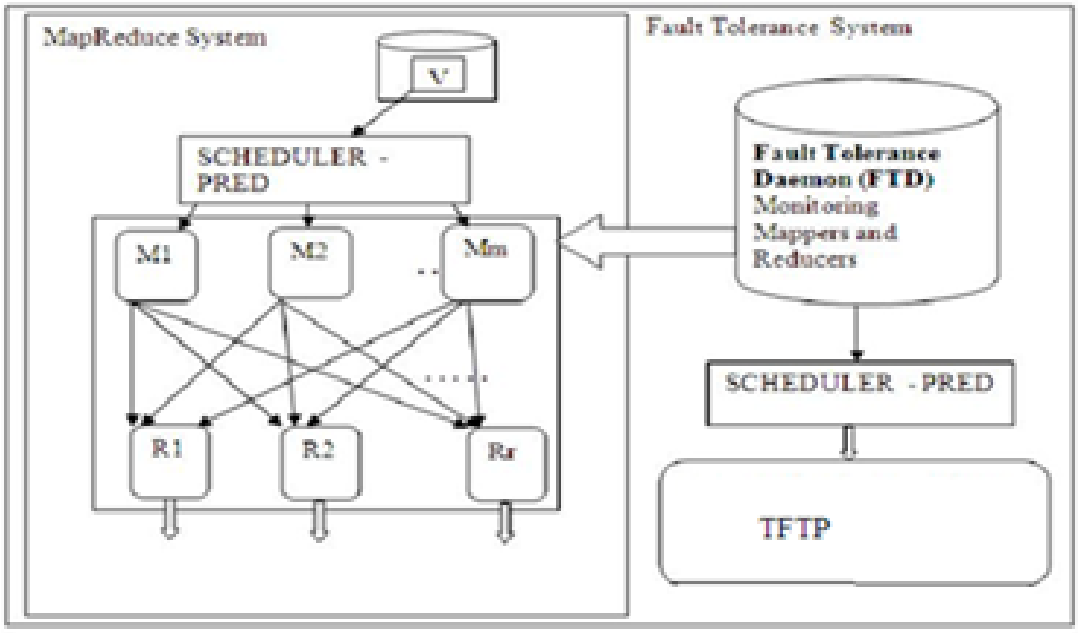


Fig 1.Architecture of the proposed system

3.1 FTD

FTD is a Fault Tolerance Daemon which constantly monitors the Mappers and Reducers for the occurrence of any fault. It constantly communicates with each node and provides the status of the node at regular intervals. If any node fails to response, the node is assumed to be the faulty node. The failure is mainly due to various reasons like hardware failure, communication failure, unexpected events like external power failure etc. The nodes which fail due to any of the above reasons are called Faulty nodes.

3.2 SCHEDULER

Scheduler is responsible for scheduling the nodes in case of occurrence of faults. FTD identifies faulty nodes if any, the information about it is passed to the scheduler. The scheduler then finds the job that was being carried out by the faulty node. It then selects a set of nodes from the straggler pool and estimate the efficiency of each node. Divisible Load Theory is applied over the stragglernodes. Based on the capabilities of straggler nodes, the data is splitted up and given. Since Divisible Load Theory is applied to the straggler nodes, all the nodes complete their execution almost at the same deadline. This scheduler can be implemented in both map and reduce phases.The Scheduler is also responsible for scheduling multi user jobs.

3.3 TFTP

TFTP refers to Two Level Fault Tolerant Partitioning algorithm. In TFTP, a Fault Tolerance Daemon constantly monitors the nodes.If any of the nodes become faulty during the execution of the job, the job given to those nodes are identified and re-executed on backups with applied DLT. Only the required number stragglers nodes are used from back up to meet out the deadline constraint as first level fault tolerant. Remaining straggler nodes in the straggler pool are kept as backup as the second level fault tolerant if there is any failure during the

execution of faulty jobs over straggler nodes which are selected at the first level. That is why it is named as Two Level Fault Tolerant Partitioning (TFTP), which may be extended to any number of levels. TFTP involves three phases namely Straggler node detection, Faulty node detection and Re-execution on backups. In the proposed system, user submits the job (V) to the master node. Then the straggler nodes are identified and are taken as backups. The master splits the job based on the capability of the mappers by applying the concept of Divisible Load Theory. The actual execution takes place then. When a node is reported to be faulty by Fault Tolerance Daemon, re-execution is done on straggler nodes present in the backups.The straggler node detection, faulty node detection, re-execution on backups are explained in the following paragraphs.

3.3.1. STRAGGLER NODE DETECTION

The map-reduce system has many nodes associated with it and the speeds of execution of each node vary widely (some of them are too fast while the other are slow performing).The slow performing nodes which are called as straggler nodes has to be excluded from the system to make it efficient and the excluded nodes can be used as backup nodes that can be used in case of any node failure. These straggler nodes have to be detected before the start of any process. The straggler nodes are slow performing both in computation and communication time. The total time for execution includes computation and communication time. Higher the total time, slower the performance. Hence the total time of execution plays a major role in the selection of straggler node. These nodes vary greatly in the performance. Forexample consider a node, counting number of words in a file. The progress rate of each node should be monitored at certain intervals and they should not vary widely. The nodes whose progress rate varies much (more than 10%) from the standard deviation of the progress rate of the all nodes are considered to be straggler nodes in our proposed method.Since the slow performing straggler nodes are taken as backups,the execution of the job is expected to be completed effectively within its deadline only with high performance computing nodes. And hence DLT can be applied efficiently over the nodes with high performance. For every node in the cluster, communication time and computation time are calculated. The Progress rate of each node is calculated at regular intervals with a sample data and the variation in progress rate should be within the limit. Based on this Probability of being Straggler node, P is calculated and finally straggler nodes are selected based on this P value.

3.3.2. FAULTY NODE DETECTION

A Fault Tolerance Daemon is made to monitor the system continuously to find the faulty nodes in the system. It continuously monitors by communicating with the nodes at regular intervals thus providing instant information about the status of the nodes. For each node, initially establish the connection and monitor it at intervals until the total job is completed, beyond which there is no need for monitoring i.e fault tolerance daemon (FTD) runs until our total job is accomplished. If incase a node completed its assigned work, then that particular node need not be monitored.

3.3.3 RE-EXECUTION ON STRAGGLER NODES

Whenever a node is found to be faulty, the job assigned to the faulty node is assigned to straggler nodes taken from the straggler pool thus making sure that the job gets completed. Previously the straggler nodes are identified, and the other nodes are constantly monitored. If a node is found to be faulty, the master has to identify the split that is assigned to the faulty node at the start of the execution, remove the straggler nodes from the back up pool. And then assign that split identified to the straggler nodesusing DLT and complete the execution of the job within the deadline constraint, without performance degradation.

5 TFTP WITH CHECKPOINTING

In TFTP, whenever a fault occurs, the job assigned to that faulty nodes is identified and assigned to the backup nodes and gets re-executed. This is not efficient in all scenarios. Consider a scenario where in 90% of the job execution is comleted and the node gets failed. In this case, the job has to be re-executed completely from the beginning which causes a big overhead. This overhead can be avoided by check pointing the data based on the failure rate of each node. Each node checkpoints the data at their specified interval in the master, during the process execution time. In case, where every node tries to checkpoint its data at the master at same time will create a bottleneck at the master. To avoid this bottleneck, the nodes have to obtain a lock on the master to checkpoints its data. Maximum number of nodes that can be handled by the master is fixed dynamically based on the capacity of the Master. When there is a failure, the master identifies the last saved data of that faulty node and assigns the remaining job to the backup nodes. Whenever a node

checkpoints its data, previously saved data is overwritten by the new data. Failure Rate in the algorithm denoted by FR is based on the previous failures of any particular node.This is maintained in a log for future reference.The Standard Deviation(SD) is calculated based on the Failure Rates of the nodes.Then, Checkpointing Time Interval(CTI) is computed. For every CTI seconds, the node backsup the data in the master.To avoid the bottle neck at the master, a lock and release strategy has been used.

6 ALGORITHM

The three phases of Two Level Fault Tolerant Partitioning (TFTP) are defined clearly in this part. The feasibility of the algorithms is tested in real-time scenarios.

Algorithm : STRAGGLER NODE DETECTION ALGORITHM

Name : Straggler Node Detection

Input : Set of nodes, Input data set (V) , number of nodes (n).

Output : Computation time, Communication time, response time, Deviation rate, Set of straggler nodes.

```
For(each node)
{
 //Initially every node is Normal node
 current_node.straggler=false
 //Start time of Progress
 S=0
 //Node processes the data
 Node.send(new File("SampleFile"));
 Master.receive("Result");
 //Finish time of the progress
 F=CurrentTimeInMaster
 //Estimated Progress Rate
 Epr=(F - S)/100
}
//Straggler Critical Time
SCT=0
//Straggler Count
S_count = 0
i=0;

For(each node in the cluster)
{
//Communication Start Time
 CST=0
 Node.send(new File("SampleFile"));
 Master.receive("Result");
 //Communication Finish Time
 CFT=CurrentTimeInSlave
 //Communication Time
 CommT=CFT-CST
 //Probability of being Straggler node
 P = 0
 //Job Start Time
 JST=0
 For(every 100 milliseconds && Job not done)
 {

 //Current Progress Rate
 Cpr = progress rate from the node.
 //Variation in Progress Rate
 Vpr = (Epr – Cpr) / Epr
 P = P + Vpr
 }
 //Job Finish Time
 JFT=CurrentTimeInMaster
 CompT= JFT - SFT
 TotalTime = (CompT + CommT)/1000
TotalTimeArray[i]=TotalTime
P_Array[i]=P
i=i+1
}
//sort nodes based on TotalTime and P values
For i=0 to n-1
{
For j=i+1 to n
{
If(TotalTimeArray[i]<TotalTimeArray[j])
{
//sorting in the decreasing order of total time
Swap(TotalTimeArray[i],TotalTimeArray[j]);
//sorting the corresponding probability value
Swap(P_Array[i],P_Array[j]);
}

//total time taken by two nodes are same
Else if(TotalTimeArray[i]==TotalTimeArray[j])
{
//sort based on the deviation rate(P value)
If(P_Array[i]<P_Array[j])
{
Swap(TotalTimeArray[i],TotalTimeArray[j]);
Swap(P_Array[i],P_Array[j]);
}
}
}
```

```
}

//number of straggler nodes dynamicallychoosen
//based on the size of the data, number of total nodes
//and also based on failure rate.
S_Count= sizeof(V)/n * n/3 *n/100
For i=1 to S_Count
{
//making the ith node as straggler
        Node[i].straggler=true
//adding the node to straggler pool
        Straggler_pool.add(Node[i])
}
```

Algorithm : FAULTY NODE DETECTION ALGORITHM

Name : Faulty Node Detection

Input : Set of Mappers and Reducers

Output : Faulty Nodes(if any)

```
For(each slave)
{
        Establish the connection
        For(every 1 second)
        {
                If(Total job completed)
                        Return (No need to
monitor)
                If(current slave completes its job)
                        Break (Go for next slave)
                If(connection not alive)
                        Report the client as faulty
        }
}
```

Algorithm : RE-EXECUTION ON STRAGGLER NODES

Name: Re-execution on stragglers

Input: Job of faulty node, Reference to Straggler pool

Output: Processed data

```
For(each slave)
{
If(fault occurred)
{
Identify the faulty client
Identify the split assigned to that faulty client
Retrieve the straggler nodes from the straggler
poolAssign the split to that straggler nodes using
DLT
For(each worker node)
{
        Read the split from the HDFS
        // w- word to be counted
        For(each word w in the split)
        {
        EmitIntermediate(w,(frequency==null)?1:fre
quency+1)
        }
        Send the result to the Reducer
        }
        Accumulate the results from the reducer
}
}
```

Algorithm : TFTP WITH CHECKPOINTING

WORKER NODE:

```
Compute the failure rate of the node as FR
//SD-Standard Deviation, TI-Time Interval
SD = √∑(mean(FR)² – TI²)
//Checkpoint Time Interval
CTI=(Estimated Completion time /SD)*Processing
time for 1MB data

For(every CTI seconds and Job not completed)
{
        If(MasterLock> 0)
        {
        //Obtain the lock on Master
                MasterLock = MasterLock – 1
                Serialize the data as object
                Send the Object to Master
                Send the reference to remain split
        //Release the lock on Master
                MasterLock = MasterLock+ 1

        }
        Else
        {
                Continue execution
                Wait for the release of lock
        }
}
```

MASTER NODE:

```
For(every node in the cluster)
{
Allocate the memory needed for check pointing
```

```
Store the reference of the split assigned
}
For(every request)
{
Receive the object from the Worker nodes
Receive the corresponding reference to the split
Data=CurrentData
ProcessedData=OldData+NewlyProcessedData
//Last Checkpointing Time
CPT=CurrentTime
}
```

Fig 3 TFTP with Checkpointing VS TFTP without Checkpointing

7 PERFORMANCE EVALUATION

The simulation is carried out in HDFS (Hadoop Distributed File System) with a cluster size of ten nodes and varying size of Jobs. Gutenberg data set is taken and is used to evaluate the system. The standard size of an e-book is taken as approximately 1M. The graph in Fig 2 is plotted by varying the number of e-books with execution time to complete the execution of word-count of e-books. The graph in Fig 2shows that TFTP takes a negligible additional time compared to default HadoopMap-Reduce system without fault. Thus even on occurrence of faults, the jobs are completed at the near estimated time.

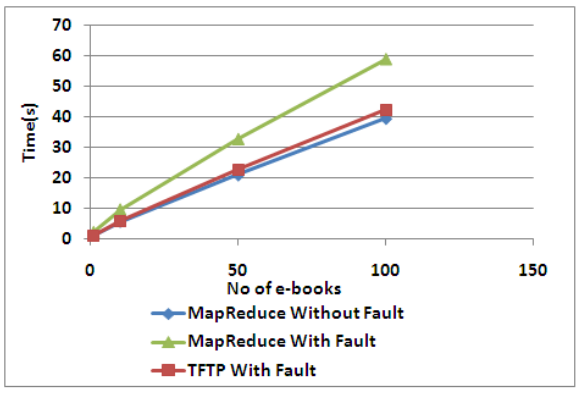
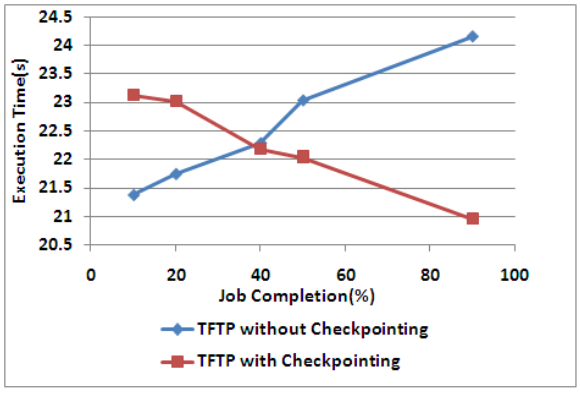


Fig 2 Normal Map-reduce system VS Map-reduce system with TFTP



The graph in Fig 3 compares the TFTP without Check pointing and TFTP with check pointing. When fault occurs during the initial stages, the checkpointing is inefficient compared to TFTP without Checkpointing. But as Job proceeds, the occurrence of fault at the later stages of job makes this method most efficient one compared to TFTP without Checkpointing.

The graph in Fig 3 is also plotted by varying the number of e-books with execution time to complete the execution of word-count of e-books. The graph in Fig 4 is about the check pointing overhead. And it shows that the check pointing time is very negligible. so a Map-reduce system without fault also tend to finish its job at the near estimated time.

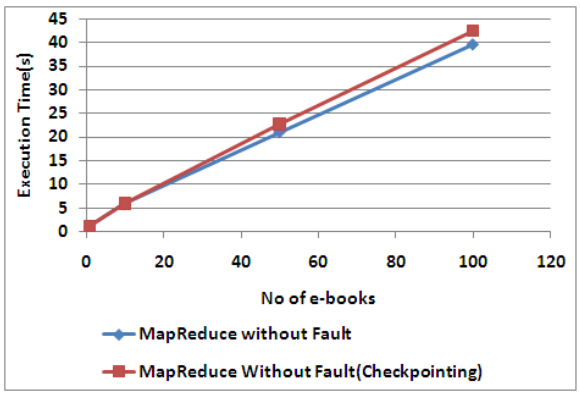


Fig 4 Checkpointing Overhead

8 CONCLUSION

As Parallel systems like Map-Reduce systems become increasingly prevalent, improving the fault behavior of these systems is very important. This paper initially proposes an algorithm TFTP for handling faults but at times the re-execution is considered to be a big overhead. To overcome that we propose a check pointing strategy which is dynamic in nature? The check pointing also has a overhead due to the unnecessary checkpoints. But later we proved that it is very negligible due to its dynamic nature. Since it is negligible, it does not affect the

efficiency of the system.

REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat (2008), 'MapReduce: Simplified Data Processing On Large Clusters', Communications of the ACM, Vol. 51, Issue 1, pp. 107-113.

[2] BerlińskaJandDrozdowski M (2011), 'Scheduling Divisible MapReduceComputations',Journal Of Parallel Distributed Computing, Vol.71, Issue 3,pp. 450-459.

[3] ThilinaGunarathne, Tak-Lon Wu, Judy Qiu and Geoffrey Fox (2010), 'MapReduceIn The Clouds For Science',2nd IEEE International Conference On Cloud Computing Technology and Science, pp. 565-572.

[4] Huan Liu and Dan Orban(2011), 'Cloud MapReduce: A MapReduceImplementation On Top Of A Cloud Operating System',International Symposium onCluster, Cloud and Grid Computing (CCGrid), 11th IEEE/ACM, pp. 467-47.

[5] Guanying Wang, Ali R. Butt, PrashantPandey and Karan Gupta(2009), 'A Simulation Approach To Evaluating Design Decisions In MapReduce Setups', IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems. MASCOTS '09, pp. 1-11.

[6] Beaumont, H. Casanova, A. Legrand, Y. Robert and Y. Yang (2005), 'Scheduling Divisible Loads On Star And Tree Networks: Results And Open Problems', IEEE Transactions on Parallel and Distributed Systems, Vol. 16, Issue 3, pp. 207-218.

[7] Thomas G. Robertazzi (2003), 'Ten Reasons To Use Divisible Load Theory',IEEE Journal on Computer, Vol.36, Issue.5,pp. 63-68.

[8] SivakumarViswanathan, BharadwajVeeravalli and Thomas G. Robertazzi (2007), 'Resource-Aware Distributed SchedulingStrategies for Large-Scale Computational Cluster/Grid Systems', IEEE Transactions On Parallel And Distributed Systems, Vol.18, Issue 10, pp.1450-1461.

[9] Mamat and Anwar (2011), 'Real-Time Divisible Load Scheduling for Cluster Computing', Computer Science and Engineering: Theses, Dissertations, and Student Research Paper 27.

[10] GaoShuai, Huang Ting-Lei and GanGuo-Ning(2010), 'An Improved Schedule Of MapReduce Programming Environment In Cloud Computing', International Conference on Intelligent Computing and Integrated Systems (ICISS), pp. 665-668.

[11] Qin Zheng(2010).., 'Improving MapReduce Fault Tolerance in the Cloud' ,IEEE conference on parallel and distributed computing.

[12] H.T.Kung, Chit-Kwan Lin and Dario Vlah(Harward university), 'Cloud Sense: Continuous fine grain cloud monitoring with compressive sensing', IEEE conference on parallel and distributed computing.

[13] Thomas C.Bressoud, Michael A.Kozuch(2009), 'Cluster Fault-Tolerance: An Experimental Evaluation of check pointing and Map Reduce through simulation', Journal of cluster cloud and grid computing.