I.J. Information Technology and Computer Science, **2015, 04, 73-78**
Published Online March 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2015.04.08

Delay Scheduling Based Replication Scheme for Hadoop Distributed File System

S. Suresh
Department of Computer Applications, National Institute of Technology, Tiruchirappalli - 620015, India
Email: sureshtvmalai85@gmail.com

N.P. Gopalan
Department of Computer Applications, National Institute of Technology, Tiruchirappalli - 620015, India
Email: npgopalan@nitt.edu

Abstract**—**The data generated and processed by modern computing systems burgeon rapidly. MapReduce is an important programming model for large scale data intensive applications. Hadoop is a popular open source implementation of MapReduce and Google File System (GFS). The scalability and fault-tolerance feature of Hadoop makes it as a standard for BigData processing. Hadoop uses Hadoop Distributed File System (HDFS) for storing data. Data reliability and fault-tolerance is achieved through replication in HDFS. In this paper, a new technique called Delay Scheduling Based Replication Algorithm (DSBRA) is proposed to identify and replicate (dereplicate) the popular (unpopular) files/blocks in HDFS based on the information collected from the scheduler. Experimental results show that, the proposed method achieves 13% and 7% improvements in response time and locality over existing algorithms respectively.

Index Terms**—** Dynamic Replication, HDFS, Delay Scheduling, Hadoop Mapreduce

I. Introduction

As data grows rapidly, the complexity of processing becomes a challenge. Applications are need to process very large amount of data of different type in short time to achieve better user experience. To provide abstracted data services to the application programs, several solutions are proposed ranging from traditional databases to current BigData managements systems. The performance of the application is mainly based on these backend data management systems. To enable distributed processing with high availability, fault-tolerance and load balancing, replication mechanism is the evergreen solution. On the other hand, maintaining consistency among the replicas in distributed environments is a time consuming process which intern affects the availability and performance.

Most of the data generated and processed by the current BigData applications follow the ‘write once and read many’ patterns which eliminates the complexity of maintaining consistency among replicas. Recent emerging distributed file systems such as Google File System (GFS) [1], Hadoop Distributed File Systems (HDFS) [2] use replication mechanisms to enable fault tolerant, high performance parallel processing. Blindly replicating all files/blocks at many place increases the availability and fault-tolerance. But will increase memory requirement proportionally. Finding hotspot and replicating them may yield better performance with less demand on memory. Determining optimal number of replica is a challenging and an active research problem for a long time as it addresses application load, data size and quality of service, etc. Current distributed computing environments such as grid computing, cloud computing are designed to process peta bytes of data in a massively parallel style. As processing speed increases rapidly with advent of multi core processors, the underlying file systems determine the performance of computing environments. To support stream like data access, modern file systems (Bigtable [3], Cassandra [4]) use very simple data model supporting limited number of operations. Some of the popular distributed file systems and

Hadoop [5] is an emerging open source platform for parallel data processing for large scale data intensive applications supported by HDFS. In this paper, a new technique called Delay Scheduling Based Replication Algorithm (DSBRA) is proposed to identify and replicate the popular files/blocks (hotspots) in HDFS using the information collected from Delay Scheduling technique. The performance of proposed algorithm is evaluated by exhaustive experiments. It is observed that, it excels in terms of response time, locality and fairness.

The paper is organized as follows: Section 2 gives background on Hadoop and HDFS. Section 3 is dedicated to related works. Section 4 elaborates the proposed replication algorithm. Sections 5 describe the simulation environment and discuss the simulation results. Section 6 concludes the paper and highlights the future research directions.

II. Hadoop and HDFS background

Hadoop is a popular parallel processing framework for cloud environments. It is an open source implementation of MapReduce [6] and GFS [1]. Due to simplicity and scalability it becomes a de-facto standard for data-intensive applications. Hadoop provides an abstracted distributed fault tolerant environment for BigData processing. The jobs submitted to the system are divided into small tasks and executed parallelly on a cluster of commodity hardware machines. Hadoop adopts the master slave architecture. Users need to write only two functions: *map* and *reduce* for their applications. All other operations such as synchronization, parallelization and handling failures are handled by the framework.

Hadoop contains two major components: (i) MapReduce is a runtime environment for parallel processing and (ii) HDFS is a distributed file system for storing input and output files. MapReduce has two major components: Jobtracker and Tasktraker. Jobtracker is the master component to keep track of all the jobs submitted into the system and scheduling. Tasktracker is a node level component which is responsible for monitoring and executing the tasks assigned to the corresponding node. All the files in HDFS are divided into fixed sized blocks and distributed across the cluster. There are two types of nodes in HDFS called namenode and datanode. Namenode is the master which is responsible for all file system operations such as creating files, deleting files, taking back up periodically, replicating missing blocks, etc. Datanodes are the slaves which are responsible for block level storage and operations such as creating, deleting and replicating blocks upon instruction from namenode. Also datanode sends block reports to namenode periodically.

III. Related Work

Much of the work concentrates on Meta data and log files for taking scheduling decisions. However, the inherent knowledge gained during scheduling is not considered much for improving data related services. This approach works well for data with diverse popularity and mixed workloads ranging from simple queries to large batch oriented jobs. Hadoop is originally designed and configured for batch oriented jobs. Due to the widespread adoption of Hadoop by various industries and academia for simplicity and scalability, several real-time user facing applications are executed on Hadoop platform. Maintaining fixed number of replicas for blocks leads to heavy load on popular blocks which affects the jobs response time. To provide better user experience, the availability of blocks is to be maintained at high level. Sometimes the terms ‘file’ and ‘block’ are used interchangeably.

Feng Wang et al. [7] proposed a method to increase the availability of Hadoop through metadata replication. To avoid single point of failure, all required metadata of critical nodes are replicated into backup node(s). This work only concentrating on metadata replication to overcome from failure and does not consider the replication of applications data. In [8], two heuristics are proposed to solve the file allocation problem in parallel I/O systems. The load balance across all disks and variance of the service time at each disks are simultaneously minimized to achieve better response time. The product of file access rate and service time, called heat of the file, is used as an objective function. In case of HDFS, files are stored as a fixed size blocks and hence, the service time may probably same for all blocks. The metrics such as service times are not suitable in HDFS and the work only considers the problem of file allocation not replication.

Jiong Xie et al. [9] presented a data placement method to balance the processing load among the nodes in heterogeneous Hadoop clusters. However, replication is not considered in their work. Wenhao Li et al. [10] proposed an incremental replication strategy to meet reliability requirement and reduce the storage cost. This work aims to meet required reliability and works well for temporary data or data with low reliability requirement. The high availability requirements of popular data blocks and load balancing are not considered. Q. Wei et al. [11] proposed a model to capture the relationship between availability and replica number. This method dynamically maintains required number of replicas to meet a given availability requirement. Sai-Qin et al. proposed a multi-objective replication strategy for cloud storage cluster [12] which is closest to our work. The objective function includes mean file unavailability, mean service time, load variance, energy consumption and mean latency. The artificial immune algorithm is used to finding replication factor and replica placement. The main problem here is setting proportionate values of objectives for getting an optimal solution. This work also does not consider the dynamic workload and load balancing.

Several other works [13-17] are presented to optimize the replication in distributed file systems. Some of them aim to optimize the replica number and some of them concentrates on replica placement with respect to various goals such as load balancing, availability, reliability and energy efficiency. Providing fault-tolerance with techniques other than replication such as erasure codes [18], are not suitable for Hadoop Framework. Because, replication is not only useful for fault-tolerance service, but also increases the availability of the data which is essential for Hadoop like systems. The performance of Hadoop is also based on various other factors, such as block placement, other than replication. For the sake of simplicity, they are not considered and considering the factors other than replication is also beyond the scope of this work.

IV. Proposed Delay Scheduling Based Replication Algorithm

Scheduling processes at nodes where the data resides is the classical solution for achieving better performance as it reduces data transfer in distributed environments. Separating data management related activities from scheduling makes scheduling and data management easier for implementation. Also it is easy for exploring new ideas for scheduling and data management in research perspective. However, taking decisions without considering both together may yield sub optimal solutions in certain cases. Knowing applications’ requirement and providing data related services accordingly is the responsibility of the underlying file system/database to achieve consistent system performance. Identifying popular data blocks (hotspots) and increasing the availability of those by replication is the classical solution for load balancing, fault-tolerance and better performance.

A. HDFS Replication

By default HDFS replicates each block into 3 locations. Two copies are stored on different nodes of the same rack and one copy in different rack for reliability. Datanodes are responsible for sending block reports on regular intervals. In case of block failures, namenode instructs the datanode to create new replicas to maintain the required replication level.

B. Delay Scheduling

Delay scheduling [19] is a simple technique to improve the locality in shared cluster environments. Delay scheduler postpones the resource allocation to the job for a while (say D sec) to achieve higher data locality if local node is not available. By slightly relaxing the fairness, delay scheduler yields a significant improvement in response time. After waiting for certain amount of time, if there is no local node found free for the job, delay scheduler starts launching non local tasks to avoid starvation until next local node is found. Approximately, running tasks on non-local node takes twice as much as the time taken in the local node. So, waiting time of the job is compensated by assigning local node in the near future. By reducing the number of tasks running on non-local node, delay scheduling also decreases the network traffic. In general, delay scheduling technique can be applied to various distributed systems beyond Hadoop scheduling.

C. Delay Scheduling Based Replication Algorithm

In this paper, the information gathered from delay scheduling technique in Hadoop is used to find the hotspots and replicates them. This algorithm is called Delay Scheduling Based Replication Algorithm (DSBRA). The number of jobs accessed a block and out of that the number of tasks delayed by delay scheduling and their consolidated delay time are recorded for each block/file in the HDFS. The scheduler maintains a list containing the number of jobs accessed, the number of job delayed and their consolidated delay time of the all the blocks. Whenever a task is assigned to a node for processing, the scheduler updates the all the aforementioned details of particular block(s)/file(s) and sends to HDFS at regular intervals. Upon receiving messages from the scheduler, HDFS stores the information as a Meta data and executes the proposed DSBRA to replicates the hotspot based on the information received from the scheduler. To consider the past history and present trend, two different values for all the afore mentioned values are maintained, one representing the exponential average delay time since the block creation and another representing the current trend since last replication.

The block's replication factor is calculated as follows: Replication Factor of block *i*,

$$RF_i = (\alpha\ (X_i) + (1\text{-}\alpha)\ (Y_i)\ )/\ 2 \tag{1}$$

where X_i represents the current load of the block *i* and Y_i represents the past history.

$$X_i = (\ M_i\ /\ N_i + DT_i\ /\ (D{*}N_i)\ )\ /\ 2 \tag{2}$$

$$Y_i = ((\ Y_i * n\ ) + RF_i\ )/(n{+}1) \tag{3}$$

where *n* is the no. of times the replication process invoked since the block creation. N_i is the no. of tasks accessed the block *i* since last replication process. M_i is the number of delayed tasks by delay scheduling for block *i* since last replication process. DT_i is the consolidated delay time of tasks by delay scheduling for block *i* since last replication process and α is history parameter in the range 0 to 1. The initial value of α and N_i is fixed as 0.5 and 1 respectively.

Algorithm-1
Algorithm: DSBRA

```
Initialize $NSum[] = 0$, $sum = 0$, $\alpha = 0.5$
when time interval is elapsed:
    //Finding Global Average of Replication Factor
    for i in blocks do
        //Replication Factor Calculation
        calculate $X_i = (\ M_i / N_i + DT_i / (D*N_i)\ ) / 2$
        $RF_i\ =\ (\alpha\ (X_i) + (1-\alpha)\ (Y_i)\ ) / 2$
        $sum = RF_i + sum$
          Add $RF_i$ with NSum values of the datanodes
                which contains block i
        $Y_i\ =\ ((\ Y_i * n\ ) + RF_i\ ) / (n+1)$
    end for
    $GAvg = sum$ / no. of Blocks
    for j in datanodes do
        $DNAvg_i = NSum_j$ / no. of blocks in node j
    if $0.2 \geq GAvg$ then
        //No replication. Only dereplication
        for i in blocks do
            if $RF_i \leq 0.1$ and $RC_i > DRC_i$ then
                delete the replica of block i located
                on node with largest DNAvg value
    else        //Replication and dereplication
        for i in blocks do
            if $RF_i \leq 0.2$ and $RC_i > DRC_i$ then
                delete the replica of block i located
                on node with largest DNAvg value
            if $RF_i > (0.2 + GAvg)$ then
                $r = (RF_i - GAvg\ ) / 0.2$
                create r replica(s) of block i and
                place one by one on the nodes
                according to DNAvg value in
                decending order
            end if
        end for
    end if
```

If the calculated block's replication factor value is 20% greater than global average block delay time of the file system then the block is identified as a hotspot. Similarly the blocks whose replication factor is low (less than 10% of the delay threshold) and its replica count is more than the default replication count are dereplicated to save the memory space. The proposed DSBRA is given in Algorithm-1.

When the global average delay time of all the blocks in HDFS is less than or equal to the 20% of the delay threshold of delay scheduling, then the proposed DSBRA only tries to dereplicates the excess replicas. If it is more than 20% of the delay threshold, the algorithm replicates/dereplicates the blocks based of replication factor value. The number of replicas is calculated based on replication factor value. DSBRA creates a replica for every 20% increase of the RF_i value and store on least loaded nodes.

Mostly, the new born data are accessed by many processes in social networking and news related applications. They are popular for short period of time and replicating such data items wastes the memory space without any benefit. So, deleting less popular data are also very important. The memory limit of the node is not considered in this algorithm. But its inclusion is a trivial addition.

V. Performance Evaluation

The performance of the proposed algorithm is evaluated by an extensive set of experiments. The simulation experiments were run on a 20 node cluster. Each node has 2.4GHz quad core processor and 4GB of RAM. All nodes have 500GB sata hard disk drives of 7200RPM and nodes are connected with 1 Gbps Ethernet. Hadoop-1.0.4 version and the IO-intensive text search job in Hive benchmark [20] is used in the experiments. The job sizes and inter-arrival time are taken from Facebook trace [19]. The schedule has 100 jobs with 14 seconds inter-arrival time which takes around half hour time for running.

The grep dataset is generated as per Hive benchmark [20]. The size of the dataset is 100GB, which is distributed throughout the cluster. As the cluster size is small, the replication factor is decreased to 2 from default value of 3 to reduce the availability of the data to observe the benefit of the replication. According to the jobs size, the input block(s) are randomly chosen from the dataset. To emulate the situation of the hotspot, around 10% of the blocks in the dataset are chosen randomly time to time and included in the input of the jobs. The performance of the proposed algorithm is evaluated by measuring the jobs' response time, data locality and waiting time caused by delay scheduling with and without DSBRA replication.

The CDF of jobs' runtimes with DSBRA and without are given in Fig.1. The maximum improvement in jobs response time is 24% over the one without replication. This is because of the proposed DSBRA is more sensitive to workload of the system and replicates the data according to the system requirements immediately. Due to DSBRA, locality and availability is increased and most of the tasks inputs are read from local disks. Overall, the DSBRA improves the response time around 13% which is significant in a shared cluster environment.

By default, delay scheduler aims to achieve better locality. But most of the attempts of the delay scheduling end unsuccessfully in the case of high demand of some files. Because, most of the jobs needs to run their tasks on the nodes which containing hotspot and delay scheduler keeps them to waits for locality. The slots of the nodes which contain hotspot are busy always due to heavy competition and most jobs are launching non-local tasks after waiting predefined delay threshold. So, delay scheduling causes more delay and hence increases the overall response time. As a result of replication of hotspots by DSBRA, scheduler has more choice of the local node and achieves high locality. This also leads to less network traffic and increase the performance.

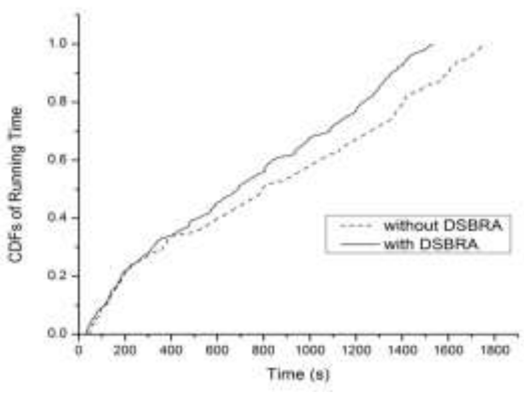

Fig. 1. CDFs of jobs' running time is with DSBRA and without DSBRA

The improvement of achieved locality is given in Fig.2. The small size jobs are mostly affected in the case of heavy load on some blocks. Because the popular data in applications, such as social networks, are small size in nature and accessed by the very large number small jobs. In addition, achieving locality for small jobs in large clusters is difficult because their input is available in few nodes. Also, these jobs require fast response to provide better user experience. The DSBRA achieves around 8% and 6% improvements over delay scheduling in the case of jobs size 1-2 maps and 30-20 respectively. The key success of the DSBRA algorithm comes from its ability to identify hotspots as earlier and replicates them immediately to avoid the performance degradation.

As a result of DSBRA, the amount of jobs' waiting time caused by delay scheduler is also decreased significantly. Because of replicating hotspots, the scheduler has more choice of the local nodes and the probability of getting a local node in less time is increased as compare to the one without hotspot replication. This can be easily observed by perceiving the improvement achieved in jobs' response time. Fig.3. shows the waiting time of delay scheduling for various job sizes with and without DSBRA. At the maximum, around 25% of waiting time is reduced by DSBRA for jobs with 1-20 map tasks. This is because of around 60% of total jobs are falls under aforementioned category and have their input data is available in less number of node. Over all, on the average, 17% of delay scheduler's waiting time is reduced by DSBRA. The benefit of delay scheduling actually comes from the cost of relaxing fairness. So by reducing delay time, DSBRA also achieves equal amount improvement in fairness of resource allocation among the jobs. The improvement in fairness is presented in Fig.3. Around 17% improvement is achieved in fairness on the average over the one without DSBRA.

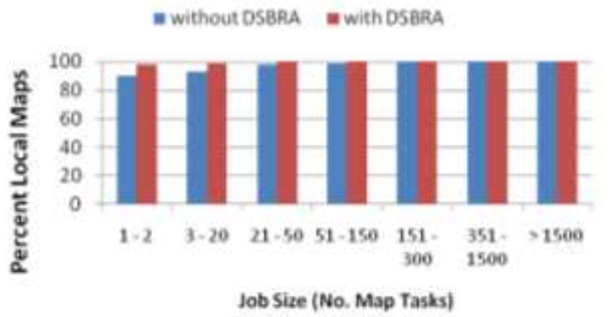

Fig. 2. Data locality of delay scheduling for various jobs sizes with and without DSBRA

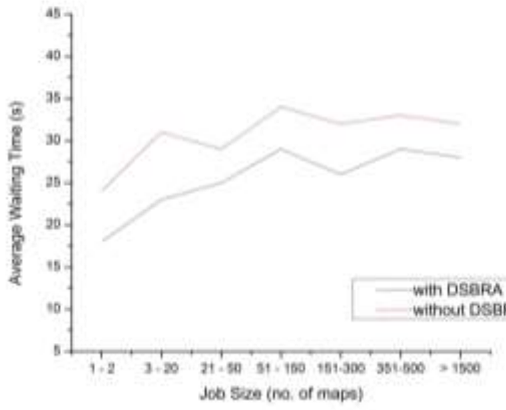

Fig. 3. Average waiting time caused by delay scheduling for various jobs sizes with and without DSBRA

To know the storage requirement of the DSBRA, its disk memory usage is compared with the default replication scheme. The percentage of excess memory used by DSBRA over default replication scheme is presented in Fig.4. The interval time between two consecutive executions is fixed as one minute. After starting the simulation, the memory requirement of DSBRA for first few minutes is same as the default replication scheme or negligible difference. In the initial stage, the delay scheduling takes few minutes to start its impact on DSBRA after starting the simulation. This is because of the popularity of the blocks is measured based on the waiting time of tasks caused by delay scheduling for locality. Also, in the starting stage cluster takes time to reach its full capacity. Hence, there is no much competition for slot among tasks.

After few minutes, the cluster reaches its full capacity and much of the tasks are starts competition for resources. So the delay scheduler makes the tasks to wait for local node. The tasks' waiting for popular blocks are starts increasing suddenly and those blocks are replicated in the subsequent iteration of DSBRA execution. At maximum, DSBRA takes around 27% excess disk memory as compare to the default fixed replication method. On the average, around 22% of excess disk memory is used by DSBRA algorithm. This is 22% of excess storage usage is not an issue in production environment as nodes are usually loaded to have significant percentage of free storage for runtime requirement.

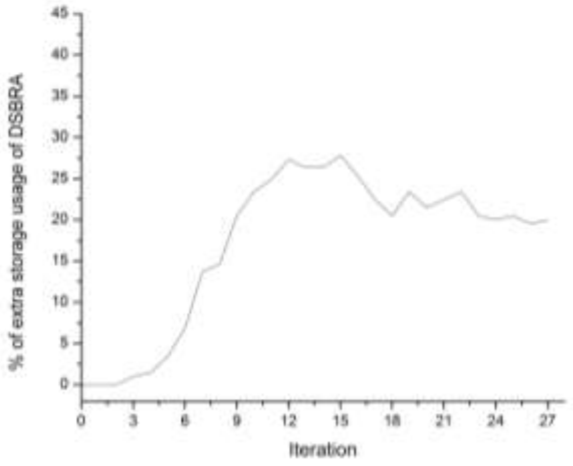

Fig. 4. Percentage of extra memory usage of DSBRA over fixed replication scheme

The number of popular blocks replicated and the number of unpopular blocks deleted is approximately as same on the average with respect time. This feature DSBRA ensures the stability of the cluster by maintaining the storage requirement as constant. Tuning few parameters with respect to the cluster load, delay threshold of the delay scheduler and data popularity yields better consistence performance of DSBRA. Even the case of heavy load and with diverse workload and data popularity, the proposed method achieves 13% and 7% improvements in response time and locality over existing one respectively.

VI. Conclusion

Modern distributed computing systems hands enormous amount of data with varying workload fluctuations. To support stream like data availability, efficient replication and load balancing techniques are required. In this paper, a new novel DSBRA algorithm is proposed for HDFS file system which replicates and/or dereplicates the files/blocks based on the information gathered from scheduling process.

DSBRA deals replication problem at block level. Mostly, the blocks of the same file have same level of workload and hence same popularity. But some blocks are stored in least loaded node and some are in heavily loaded nodes. Obliviously the data blocks on heavily loaded nodes have higher delay in access time. Replicating such blocks, even it is unpopular, improves the data locality and load balancing. If DSBRA replicates the unpopular blocks stored on heavily loaded nodes (unpopular blocks stored along with more popular blocks in a node) also popular by seeing waiting time of jobs, that will be deleted in subsequent. Experimental results show that, the proposed method achieves 13% and 7% improvements in response time and locality over existing method respectively.

REFERENCES

[1] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, “The Google File System”, In *19th Symposium on Operating Systems Principles*, Lake George, New York, pp. 29–43, 2003.

[2] Konstantin Shvachko, Hairong Kuang, Sanjay Radia and Robert Chansler, "The Hadoop Distributed File System", IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp.1-10, 2010.

[3] A. Lakshman and P. Malik, “Cassandra: A decentralized structured storage system”, SIGOPS Operating Syst. Rev., vol. 44, no. 2, 2010.

[4] F. Chang, et al., “Bigtable: A distributed storage system for structured data,” ACM Trans. Comput. Syst., vol. 26, no. 2, 2008.

[5] Apache Hadoop. http://hadoop.apache.org/. Accessed on 13 June, 2014.

[6] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters”, In Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 137–150, 2004.

[7] Feng Wang et al., “Hadoop high availability through metadata replication”, In Proceedings of the first international workshop on Cloud data management (CloudDB '09), ACM, New York, NY, USA, pp. 37-44, 2009.

[8] Lin-Wen Lee et al, “File Assignment in Parallel I/O Systems with Minimal Variance of Service Time”, IEEE Transactions on Computers, vol. 49, no. 2, Feb 2000.

[9] Jiong Xie et al., “Improving MapReduce performance through data placement in heterogeneous Hadoop clusters”, Symposium on Parallel and Distributed Processing, pp.1-9, 2010.

[10] W.H. Li et al., “A novel cost-effective dynamic data replication strategy for reliability in cloud data centres”, in: IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, 2011.

[11] Q. Wei et al., “CDRM: a cost-effective dynamic replication management scheme for cloud storage cluster”, in: Proc. 2010 IEEE International Conference on Cluster Computing, Heraklion, Crete, Greece, September 20–24, pp. 188–196, 2010.

[12] Sai-Qin Long, Yue-Long Zhao and Wei Chen, “MORM: A Multi-objective Optimized Replication Management strategy for cloud storage cluster”, Journal of Systems Architecture, vol. 60, no. 2, pp. 234–244, Feb 2014.

[13] K. Ranganathan, I.T. Foster, Identifying dynamic replication strategies for a high-performance data grid, in: Proc. Second Int’l Workshop Grid Computing (GRID), 2001.

[14] H. Lamehamedi, Z. Shentu, B. Szymanski, Simulation of dynamic data replication strategies in data grids, in: Proc. 12th Heterogeneous Computing Workshop (HCW2003) Nice, France, April 2003, IEEE Computer Science Press, Los Alamitos, CA, 2003.

[15] R.S. Chang and H.P. Chang, “A dynamic data replication strategy using access weights in data grids”, J. Super comput. Vol. 45, No. 3, pp. 277–295, 2008.

[16] S.C. Choi and H.Y. Youn, “Dynamic hybrid replication effectively combining tree and grid topology”, J. Supercomput. vol. 59, pp. 1289–1311, 2012.

[17] T. Xie, Y. Sun, A file assignment strategy independent of workload characteristic assumptions, ACM Trans. Storage, vol. 5, no. 3, 2009.

[18] L. Hellerstein et al., "Coding techniques for handling failures in large disk arrays", Algorithmica, vol. 12, vo. 3-4, pp. 182-208, 1994.

[19] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling”, In Proceedings of the 5th European Conference on Computer systems (EuroSys), 2010.

[20] Hive performance benchmarks. http://issues.apache.org/jira/browse/HIVE-396. Accessed on 17 June, 2014.

Authors’ Profiles

S.Suresh: Research Scholar at Department of Computer Applications, National Institute of Technology, Tiruchirappalli. He received MCA and M.Phil from University of Madras, Chennai. His areas of interest include Algorithms, Cloud Computing, Parallel and Distributed Computing, BigData Processing and Analytics.

N.P.Gopalan: Professor of Computer Applications Department, National Institute of Technology, Tiruchirappalli, TamilNadu, India. Done PhD from IISC Bangalore. Interested in Data mining, Web Technology, Distributed Computing and Theoretical Computer Science.

How to cite this paper: S. Suresh, N.P. Gopalan,"Delay Scheduling Based Replication Scheme for Hadoop Distributed File System", International Journal of Information Technology and Computer Science(IJITCS), vol.7, no.4, pp.73-78, 2015. DOI: 10.5815/ijitcs.2015.04.08