Volume: 07 Issue: 03 Pages: 2741-2751 (2015) ISSN: 0975-0290

QoS-Aware Data Replication in Hadoop Distributed File System

Dr. Sunita Varma
Department of ComputerTechnology and Application
S. G. S. I. T. S.
Indore, (M. P.), India
sunita.varma19@gmail.com
Ms. Gopi Khatri
Department of Computer Engineering
S. G. S. I. T. S
Indore, (M. P.), India
gopikhatri149@gmail.com

ABSTRACT

Cloud computing provides services using virtualized resources through Internet on pay per use basis. These services are delivered from millions of data centers which are connected with each other. Cloud system consists of commodity machines. The client data is stored on these machines. Probability of hardware failure and data corruption of these low performance machines are high. For fault tolerance and improving the reliability of the cloud system the data is replicated to multiple systems.

Hadoop Distributed File System (HDFS) is used for distributed storage in cloud system. The data is stored in the form of fixed-size blocks i.e. 64MB. The data stored in HDFS is replicated on multiple systems for improving the reliability of the cloud system. Block replica placement algorithm is used in HDFS for replicating the data block. In this algorithm, QoS parameter for replicating the data block is not specified between client and service provider in the form of service level agreement.

In this paper, an algorithm QoS-Aware Data Replication in HDFS is suggested which considers the QoS parameter for replicating the data block. The QoS parameter considered is expected replication time of application. The block of data is replicated to remote rack DataNodes which satisfies replication time requirement of application. This algorithm reduces the replication cost as compared to existing algorithm thus, improving the reliability and performance of system.

Keywords-**Cloud computing; quality of service; data replication; Hadoop distributed file system; replication cost.**

Date of Submission: July 09, 2015 Date of Acceptance: Aug 12, 2015

I. INTRODUCTION

Computing is being transformed to a model consisting of services that are commoditized and delivered in a manner similar to utilities such as water, electricity, gas and telephony. In such a model, users access services based on their requirements regardless of where they are hosted. Cloud computing is the most recent paradigm promising to turn the vision of computing utilities into realities. Cloud computing is a technological advancement that focuses on the way in which the user can design computing system, develop applications and utilizing existing services for building software. It is based on concept of dynamic provisioning which is applied not only to services but also to compute capability, storage, networking, and information technology infrastructure in general. Resources are made available through the Internet and on a pay-per-use basis from cloud computing vendors. Today, anyone with a credit card can subscribe to cloud services, deploy and configure servers for an application in hours, growing and shrinking the infrastructure serving its application to the demand and paying only for the time these resources have been used.

Cloud computing system consists of millions of data centers which are located throughout the world. The cloud services are provided by these data centers. Moreover, these data centers are capable of storing and performing large amount of data which are generated by data intensive applications. The key challenge of cloud computing system is to store and process large amount of data.

The Apache Hadoop is a framework for cloud computing system. It allows distributed processing of large amount of data across cluster of computers. One of the module of Hadoop is Hadoop Distributed File System (HDFS) which organizes files and stores their data on distributed computing. HDFS has a master/slave architecture containing a single NameNode as a master and number of DataNodes as workers (Slaves). To store file in this architecture, HDFS splits the file into fixed size blocks of 64MB and stores them on DataNodes. The mapping of blocks to DataNodes is stored on NameNode. The NameNode also manages the files system's metadata and namespace.

Hadoop framework runs on millions of commodity machines. Hence, the probability of failure of DataNodes, network link and storage media is high which makes the data unreliable. For proving reliability of data in HDFS, it replicates the data on multiple DataNodes. The block replica placement algorithm is used for selecting the

DataNodes in which random remote DataNodes are selected.

In this paper the quality of service (QoS) parameter is considered for replicating the data on DataNodes. The QoS parameter considered for data replication is replication time. The block of data is stored on those DataNodes which satisfies the minimum replication time of the client application. This algorithm reduces the replication cost which results in improved reliability and performance of system.

The rest of paper is organized as follows. Section II describes the system overview. Section III overviews the related work. Section IV presents the data replication in HDFS. Section V evaluates the performance of proposed algorithm. Finally Section VI concludes the paper.

II. SYSTEM OVERVIEW

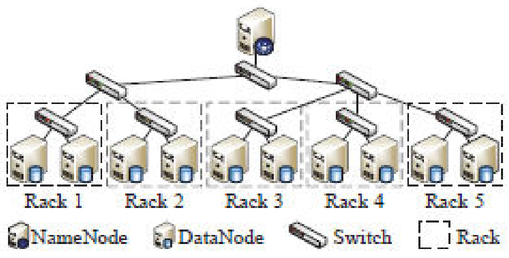
Several distributed file system (DFS) like Ceph, Google File System (GFS), gfarm, GlusterFS, OpenAFS, XtreemFS, MooseFS and Lustre are being developed [1]. Hadoop Distributed File system(HDFS) is one of popular DFS which is used by more than one hundred organization include yahoo, Adobe, A9.com - Amazon, Able Grape, BabaCar, Cloudspace, Detektei Berlin, EBay, Facebook, FOX Audience Network and so on[2]. It is inspired by GFS that organizes files and stores their data on distributed computing system.

A. Overview of Hadoop Architecture

It consists of single NameNode and multiple DataNodes. NameNode manages the file system's metadata and namespace. In such systems, the namespace is the area maintaining the metadata, and metadata refers to all the information stored by a file system that is needed for overall management of all files i.e. permission, modification and access time of files/directories. The file content is split into blocks and each block of file is replicated at different DataNodes. Hence, NameNode maintains the namespace tree and the mapping of file blocks to DataNodes (physical location of file data). Each DataNode is responsible for storing and retrieving its file block. DataNodes are deployed within a rack as shown in fig. 1. Each DataNode has associated rack number to represent in which rack it is located [3].

Fig. 1: Architecture of HDFS

B. HDFS Fault Tolerance



One of the aspect of HDFS is its fault tolerance characteristics. Since Hadoop is designed to be on low-cost hardware by default, a hardware failure in this system is considered to be common rather than exception. Therefore, Hadoop considers the following issues to fulfill reliability requirement of the file system [4]

1)Block Replication: To reliably store data in HDFS, file blocks are replicated in this system. In other words, HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster. The replication factor is set by the user and is three by default.

2)Replica Placement: The placement of the replica is another factor to fulfill the desired fault tolerance in HDFS. Although storing replicas on different DataNodes located in different racks across the whole cluster provides more reliability, it is sometimes ignored as the cost of communication between two DataNodes in different racks is relatively high in comparison with that of different DataNodes located in the same racks. Therefore, sometimes HDFS compromises its reliability to achieve lower communication costs. For example, for the default replication factor of three, HDFS stores one replica in the same DataNode from where HDFS client sends a write request, one replica on a different DataNode but in same rack, and one replica on a different DataNode in different rack to provide three copies of data[5]

3)Heartbeat and Blockreport messages: Heartbeat and Blockreport messages are periodic send to the NameNode by each DataNodes in cluster. Receipt of a Heartbeatmessage implies that the DataNode is functioning properly. The default heartbeat message interval is three second. If the NameNode does not receive a heartbeat message from a DataNode in ten minutes the NameNode considers DataNode to be out of service and the block replicas hosted by that DataNode to be unavailable. While, each Blockreport message contain a list of all blocks on a DataNode[5]. The NameNode receives such messages because it is the sole decision maker of all replicas in the system.

1) HDFS High-Throughput Access to Large Data Sets (Files):

Because HDFS is primarily designed for batch processing rather than interactive processing, data access throughput in HDFS is more important than latency. Also, because applications run on HDSF typically have large data sets, individual files are broken into large blocks (e.g. 64 MB) to allow HDFS to decrease the amount of metadata storage required per file. This provides two advantages: The list of blocks per file will shrink as the size of individual blocks increases, and by keeping large amount of data sequentially within a blocks, HDFS provides fast streaming reads of data.

2) HDFS Operation

The control flow of HDFS operations such as write and read can properly highlight roles of the NameNode and DataNodes in the managing operations. In this section, the control flow of the main operations of HDFS on files is further described to manifest the interaction between HDFS client, the NameNode, and the DataNodes in such system[6].

1) *Reading a file:* To read a file in HDFS, a user sends an "open" request to the NameNode to get the

location of file blocks. For each file block, the NameNode returns the address of a set of DataNodes containing replica information for the requested file. The number of addresses depends on the number of block replicas. Upon receiving such information, the user calls the *read* function to connect to the closest DataNode containing the first block of the file. After the first block is streamed from the respective DataNode to the user, the established connection is terminated and the same process is repeated for all blocks of the requested file until the whole file is streamed to the user.

2) *Writing to a file:* To write a file in HDFS, a user sends a "create" request to the NameNode to create a new file in the file system namespace. If the file does not exist, the NameNode notifies the user and allows him to start writing data to the file by calling the *write* function. The block of the file is written to an internal queue termed the data queue while a data streamer monitors its writing into a DataNode. Since each file block needs to be replicated by a predefined factor, the data streamer first sends a request to the NameNode to get a list of suitable DataNodes to store replicas of the first blocks.

The Streamer then stores the block in the first allocated DataNode. Afterward, the block is forwarded to the second DataNode by the first DataNode. The process continues until all allocated DataNodes receive a replica of the first block from the previous DataNode. Once this replication process is finalized, the same process starts for the second block and continues until all blocks of the file are stored and replicated on the file system.

III. RELATED WORK

In [7],Gao and Diao discussed the problem of maintaining the consistence of data replication in cloud computing system. The Author used Lazy update approach to separate the process of data replica updating and data access in cloud which improve the throughput of the information, data service and reduce response time.

NameNode is single point of failure (SPOF) in Hadoop. For this reason studies have been undertaken to make the NameNode resilient to failure. Hadoop provided two mechanisms [8], the first way was to back up the files that make up the persistent state of file system metadata; second was to run secondary NameNode to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large.

HDFS federation [9] allowed a cluster to scale by adding NameNodes, each of which managed a portion of the filesystem namespace. The DataNodes were used as common storage for blocks by all the NameNodes. Each DataNode registered with all the NameNodes in the cluster. NameNodes were federated, that is, the NameNodes were independent and didn't require coordination with each other. Hence, failure of one NameNode did not affect the availability of the namespaces

A subproject of Hadoop, named Zookeeper [10], supported replication among a set of servers and provided a coordination mechanism for leader election among the servers, but it focused on providing a coordination service for distributed applications instead of a high availability solutionmanaged by other NameNodes.

Wang et al. proposed a metadata replication based to enable Hadoop high availability by removing SPOF[11]. This solution involved three phases: initialization phase, replication phase and failover phase; and also presented several unique features for improving availability of Hadoop e.g. reconfigurable synchronization mode and corresponding adaptive decision method.

QoS-aware data replication problem has been studies in the literature. It has been studied in content distribution system [12]-[13]. Tang and Xu[12] were suggested two heuristic algorithms called l-greedy insert and l-greedy delete. l-greedy insert algorithm was started with an empty replication strategy and continued to insert replicas into replication strategy until all QoS requirement were satisfied. l-greedy delete algorithm was started from complete replication strategy and continued to remove replicas from replication strategy provided that no QoS requirement was violated. In [13] Wang et al. were proposed another heuristic algorithm called greedy cover that determined the position of replicas in order to satisfy the quality requirements imposed by data requests. The cover set c(u) of a server u is the set of servers that were within the QoS requirement q(u) from u.

QoS-aware data replication problem has been discussed in data grid [14]. This paper proposed a dynamic placement strategy named Least Value Replacement (LVR), which was based on the future value prediction method. The LVR framework could automatically decide which replica file to be deleted whenever the grid site was full. Replication and deletion of file were based on data-access frequency and free space on storage elements. It also added some effective factor for describing condition replica prediction probabilities.

In [15], Fu et al. addressed QoS-aware data replication problem in a tree based mobile grid environments to meet the QoS requirements of all mobile users and load balancing of replicas. Authors proposed two-step solution: the first step was a bottom-up dynamic programming approach for placing replicas to satisfy the QoS requirements of mobile users and workload constraint of replicas; then, a binary search based algorithm was used to finish the k replica placement problem in mobile grid environments.

Ref.[16] explored QoS-aware distributed replica placement in hierarchical data grids by introducing a new highly distributed and decentralized replica placement algorithm that determined locations for placing replicas to meet the QoS requirement while minimizing overall replication cost and maximizing QoS satisfaction for a given traffic pattern.

There were also other algorithms that provide QoS-aware placement for systems such as distributed and mobile databases; and P2P networks. In general, the characteristics of these algorithms make them inappropriate for use in our target environment and they are not discussed in this paper.

Data replication was done to make sure the availability of data in case of loss of original copy of data due to failure of network link or hardware in system. Block replica placement technique[17] was used in HDFS, to replicate the data, but they didn't consider Quality of service (QoS) requirement of applications. Also, QoS-aware data replication problem was addressed in various areas including grid mobile environment using different techniques but, they ignored replication cost as a QoS parameter.

In this paper, we proposed a QoS-aware Data replication in HDFS which considered replication cost as a QoS parameter and reduced the replication cost of system and improve reliability of system.

IV. DATA REPLCATION IN HDFS

A. *Block Replica Placement Policy*

HDFS consists of a single NameNode and a set of DataNodes. The NameNode and DataNodes are deployed within a number of racks. The NameNode mainly manages the file system namespace and the locations of data blocks (the mapping of data blocks to DataNodes). Applications are executed in DataNodes. When an application would like to write or read a file, it acts as HDFS client. As the client writes file, this file is split into data blocks of size 64MB by DFSOutputStream. These blocks is written to data queue. The data queue is consumed by DataStreamer. Then DataStreamer sends write request to NameNode. After receiving write request, NameNode finds the location of target DataNodes by block replica placement strategy, where block of data will be hosted. DataNode from where DataStreamer requests to NameNode is called Local DataNode. Rack of Local DataNode is called Local rack whereas, racks other than Local rack are called remote racks. Moreover, DataNodes that are in remote racks or in different (remote) rack than Local rack are called remote rack DataNodes.

In block replica placement mechanism, location of target DataNodes are return by NameNode to client for storing data block. Number of target DataNodes depend on replication factor. By default replication factor in HDFS is three, therefore location of three target DataNodes are returned by NameNode. First target DataNode is the local DataNode. In additional to, second and third target DataNodes are remote rack DataNodes. Location of these three target DataNodes are returned to client by NameNode. Then, the list of target DataNodes forms a pipeline. The DataStreamer streams the block to the first target DataNode in the pipeline, which stores the block and forwards it to the second target DataNode in the pipeline. Similarly, the second target DataNode stores the block and forwards it to the third (and last) target DataNode in the pipeline as shown in fig. 2.

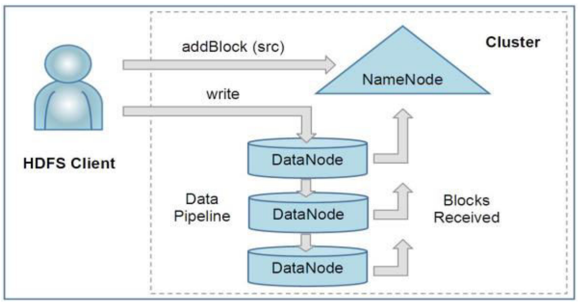
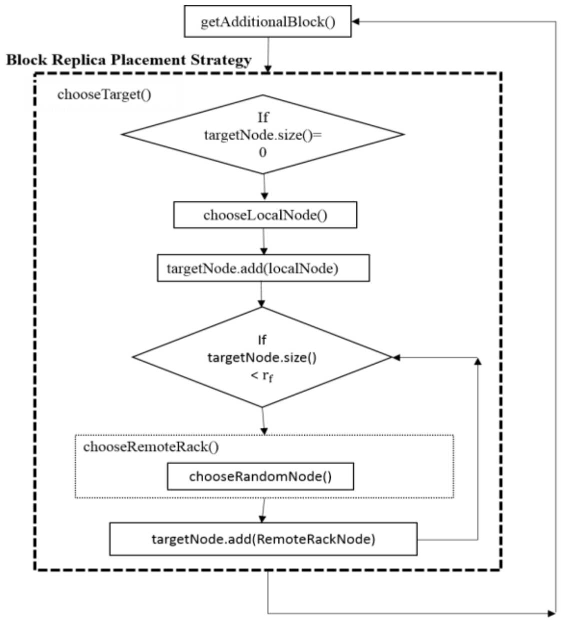


Fig. 2:Interactions among the Client, the NameNode and the DataNodes

1) *Flow chart of Block Replica Placement Policy:* Flow chart of block replica placement mechanism in HDFS is shown in fig. 3. The methods used in block replica placement strategy are as follow:

a) *getAdditionalBlock() method:*
This method is used to obtain an additional block for the file (which is being written-to). It returns an array that consists of the block, plus a set of machines. It returns an empty array if NameNode wants the client to "try again later". This method is described in FSNameSystem.java file. This file is in org.apache.hadoop.hdfs.server.name node package.


getAdditionalBlock()
Block Replica Placement Strategy
chooseTarget()
If targetNode.size()= 0
chooseLocalNode()
targetNode.add(localNode)
If targetNode.size() < r_f
chooseRemoteRack()
chooseRandomNode()
targetNode.add(RemoteRackNode)
Fig 3. Flow chart of block replica placement in HDFS

b) *chooseTarget() method:* This method is responsible for choosing the 'rf' number of targets DataNodes for placing block replicas, where 'rf' is the number of replica of block. By default it is three. It returns an array that consists of target DataNodes. These target DataNodes are chosen according to replica placement strategy. This method is described in BlockPlacementPolicyDefault.java file. This file is in org.apache.hadoop.hdfs.server.name node package.

c) *chooseLocalNode() method:* This method chooses local machine as the target DataNode. If local machine is not available, choose a DataNode on the same rack. It returns the chosen DataNode. This method is also described in BlockPlacementPolicyDefault.java file.

d) *chooseRemoteRack() method:* This method chooses DataNodes from the remote racks. If not enough nodes are available, choose the remaining ones from the local rack. This method is also described in BlockPlacementPolicyDefault.java file.

e) *chooseRandom() method:* This method randomly chooses target from DataNodes. This method is also described BlockPlacementPolicyDefault.java file.

When DataStreamer sends write request to NameNode, getAdditionalBlock() method is called to obtain an additional block 'b' of file 'src'. As shown in fig. 3, this getAdditionalBlock() method calls chooseTarget() method, for choosing 'rf' target DataNodes from all available DataNodes based on block replica placement strategy.

Here 'targetNodes' is variable which contains location of target DataNodes where block will be stored. If number of location of target DataNodes in 'targetNodes' is zero, chooseLocalNode() method is called. This method selects the local machine as a target DataNode and stored location of local machine in 'targetNodes'. If number of location of target DataNodes stored in 'targetNodes' are less than 'rf', chooseRemoteRack() method is called, which in turn calls chooseRandomNode() method that randomly selects a DataNode from remote rack and stored in 'targetNodes'. chooseRemoteRack() method is called till number of location of target DataNodes in 'targetNodes' is equal to 'rf'. If number of location of target DataNodes in 'targetNodes' is equal to 'rf', 'targetNodes' is returned by chooseTarget() method to getAdditionalBlock(). Then target DataNodes organizes a pipeline and DataStreamer streams the block to the first target DataNode, second target DataNode and third target DataNode one by one to write in the disk of target DataNodes.

2) *Algorithm of Block Replica Placement Policy:*

Table 1 lists the notations used in algorithms. The pseudo code of Block Replica Placement algorithm is given in fig. 4, whose operations are elaborated as follows:

Block replica placement algorithm takes 'src', 'replicationFactor' 'clientNode' and 'blockSize' as input. Output of this algorithm is the 'result', which holds the locations of target DataNodes where block will be hosted as per block replica placement mechanism.

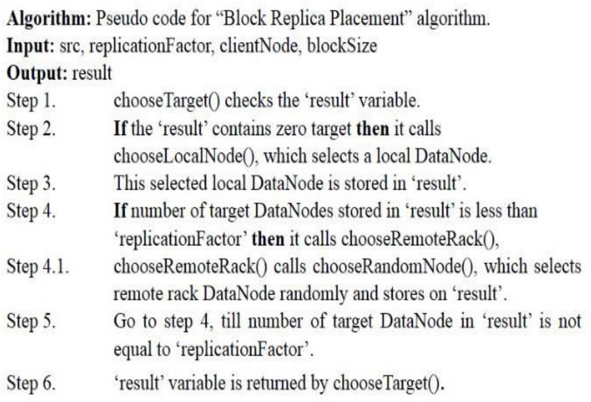
As DataStreamer sends write request of file 'src' to NameNode for location of target DataNodes, NameNode calls getAdditionalBlock() method to obtain an additional block of the file 'src' where, size of block is 'blockSize'. This method calls chooseTarget() method which selects 'replicationFactor' target DataNodes for placing the block of file 'src'. The selection of target DataNodes are based on block replica placement strategy.

Table 1
Notation used in Block Replica Placement Algorithm

Notation	Description
clientNode	DataNode from where client requests for storing blocks of file.
Src	Name of the file, client wants to write on the DataNodes.
replicationFactor	Number of replicas of data block to be stored on DataNodes.
blockSize	Size of each data block stored on DataNode.
Result	An array holds the address of target DataNodes based on block replica placement

In block replica placement algorithm, chooseTarget() method checks the result variable and finds number of target DataNodes in 'result' variable as given in step1. If 'result' variable doesn't contain the location of DataNode, it calls chooseLocalNode() method in step2. This method selects the local DataNode as a target DataNode and stores address of local DataNode in 'result' variable in step 3. If

number of target DataNodes in 'result' variable is less than 'replicationFactor', it calls chooseRemoteRack()



```
Algorithm: Pseudo code for "Block Replica Placement" algorithm.
Input: src, replicationFactor, clientNode, blockSize
Output: result
Step 1.     chooseTarget() checks the 'result' variable.
Step 2.     If the 'result' contains zero target then it calls
            chooseLocalNode(), which selects a local DataNode.
Step 3.     This selected local DataNode is stored in 'result'.
Step 4.     If number of target DataNodes stored in 'result' is less than
            'replicationFactor' then it calls chooseRemoteRack(),
Step 4.1.   chooseRemoteRack() calls chooseRandomNode(), which selects
            remote rack DataNode randomly and stores on 'result'.
Step 5.     Go to step 4, till number of target DataNode in 'result' is not
            equal to 'replicationFactor'.
Step 6.     'result' variable is returned by chooseTarget().
```

Fig. 4. The block replica placement algorithm

method which in turn calls chooseRandomNode() method that randomly selects the remote rack DataNode as a target DataNodes.It then stores the address of this target DataNode in 'result' variable as given in step 4. chooseremoteRack() method is called till number of target DataNodes in 'result' variable is equal to 'replication Factor'. As number of target DataNodes in 'result' variable is equal to 'replicationFactor', result is returned by chooseTarget() method in step 6. In this way, 'result' variable contains address of target DataNodes, where blocks of files 'src' has to be stored.

3) *Example of Block Replica Placement Policy:*A small scale HDFS is depicted in fig. 5. The HDFS consists of NameNode 'NN'; SecondaryNameNode 'SN'; six DataNodes 'D1' 'D2' 'D3' 'D4' 'D5' and'D6'; and three racks 'R1' 'R2' and 'R3'. Two DataNodes are deployed with in each rack. 'D1' and 'D2' are in 'R1', 'D3' and 'D4' are in 'R2' and 'D5' and 'D6' are in 'R3'. Consider, applications 'A1' 'A2' and 'A3' are executing on 'D1' 'D2' and 'D3' respectively. Replication factor is three.Suppose ‘A1’ issues a write request to NameNode. After receiving write request NameNode finds target DataNodes by block replica placement algorithm. First target DataNode is ‘D1’ because ‘D1’ is local DataNode. Other target DataNodes are ‘D3’ and ‘D4’ or ‘D5’ and ‘D6’ because these DataNodes are in different racks. Hence either address of ‘D1’, ‘D3’ and ‘D4’ or address of ‘D1’, ‘D5’ and ‘D6’ DataNodes are returned by NameNode to ‘D1’ client DataNode.

Similarly, the application ‘A2’ issues write request to NameNode. Address of ‘D2’, ‘D3’ and ‘D4’ or address of ‘D2’, ‘D5’ and ‘D6’ DataNodes are returned by NameNode to ‘D2’ client DataNode.

Similarly, the application ‘A3’ issues write request to NameNode. Address of ‘D3’, ‘D5’ and ‘D6’ or address of ‘D3’, ‘D1’ and ‘D2’ DataNodes are returned by NameNode to ‘D3’ client DataNode.

B. *QoS-Aware Data Replica Placement Policy:*

When an application would like to write a data block, the DataStreamer would issue a write request for the data block to the NameNode. The QoS parameter considered in QoS-aware data replication policy is *replication time*.

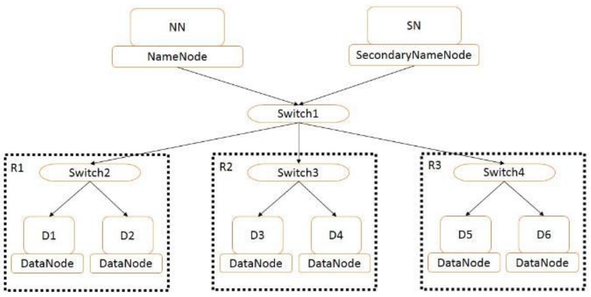


Fig.5. A small scale Hadoop distributed file system example

The information about expected replication time requirement of the requested application is also attachedwith the write request.This write request is called QoS-aware write request. Suppose ‘DR_i’ is requested DataNode on whichrequested application is run. When processing a QoS-aware write request from the requested DataNode ‘DR_i’ by the NameNode, it requires to find the corresponding target DataNodes that satisfy the replication time requirement of the requesting application running on ‘DR_i’.Notation used in QoS-aware Data Replication policy is given in table 2.

TABLE 2
Notation used in QoS-aware Data Replication policy

Notation	Description
D_i	ith DataNode in Hadoop Distributed File system.
N	Total Number of DataNodes in HDFS.
DR_i	ith Requested DataNode in HDFS.
$T(DR_i)$	Replication time requirement of a requested application, running on ith requested DataNode in HDFS.
DT_j	jth Target DataNode of DR_i.
$R(DR_i)$	Rack in which ith requested DataNode is located.
$R(DT_j)$	Rack in which jth target DataNode is located.
$T_{disk}(D_j)$	Disk storage time for storing a data block to the disk of DataNode D_j.
$T_{com}(D_i\ D_j)$	Network communication time for transmitting a data block replica from D_i to D_j.
SLA	Service level agreements

Let the replication time requirement of the requested application running on the requested DataNode ‘DR_i’ is $T(DR_i)$ time units. If the DataNode ‘DT_j’ is one of the target DataNode of ‘DR_i’, it needs to meet the following two conditions:

- The DataNodes ‘DR_i’ and ‘DR_j’ cannot be located within the same rack. This condition is for considering the possible rack failure.

$$R(DR_i)\,!= R(DT_j) \qquad (1)$$

Where, R is the function to determine in which rack a DataNode is located.

- The replication time of data block $T_{replication}(DR_i, TD_j)$ from DR_i to DT_j needs to meet the $T(DR_i)$ constraint.

$$T_{replication}(DR_i, DT_j) \le T(DR_i) \qquad (2)$$

The *replication cost* is the total replication time taken by all the requested DataNodes to store their respectively data block replica.

1) Assumptions for QoS-Aware Data Replica Placement Policy:

When an application wants to write file, corresponding data blocks are written to data queue. After that DataStreamer sends a QoS-aware write request to NameNode. NameNode is replied by giving address of DataNodes where block will be hosted. NameNode finds target DataNodes based on QoS-aware data replication algorithm. Before describing the details of QoS-aware data replication algorithm, the following assumptions are made:

a) It is assumed that only one application runs in DataNode during a given time interval. Only one file is opened to execute the operation by application.

b) A file is divided into number of blocks. Size of each of block is 64MB.

c) Each data block by default has three replicas. QoS-aware data replication algorithm also by default has three replicas of block including original block.

d) NameNode has a table having replication time between any two DataNodes. The replication time between ith DataNode and jth DataNode is calculated as:

$$T_{replication}(D_i, D_j) = T_{disk}(D_j) + T_{com}(D_i, D_j) \qquad (3)$$

Where, $T_{replication}(D_i, D_j)$ is the time to store a data block replica from D_i to D_j, D_i is ith DataNode, D_j is jth DataNode, $T_{com}(D_i, D_j)$ is network communication time for transmitting a data block replica from D_i to D_j and $T_{disk}(D_j)$ is disk storage time for storing a data block to the disk of DataNode D_j.

e) The replication time is the QoS parameter considered in QoS-aware data replication algorithm. The requirement of data replication time of an application can be specified in the service level agreements (SLA). The replication time requirement of application is passed as parameter with the QoS-aware write request to NameNode.

f) In this mechanism, one DataNode cannot have more than one replica of same data block and one rack cannot have more than two replicas of the same data block.

2) Flow chart of QoS-Aware Data Replica Placement Policy:

Flow chart of QoS-aware Data replica placement mechanism in HDFS is shown in fig 6. The methods used in QoS-aware Data replica placement strategy are as follow:

chooseTarget() method: This method is responsible for selecting the targets DataNodes for placing block replicas. Where number of target DataNodes is equal to 'r_f' and 'r_f' is the number of replica of block. It returns an array that contains target DataNodes. These target DataNodes are chosen according to QoS-aware data replica placement strategy. This method is described inBlockPlacementPolicyDefault.java file. This file is in org.apache.hadoop.hdfs.server.namenode package.

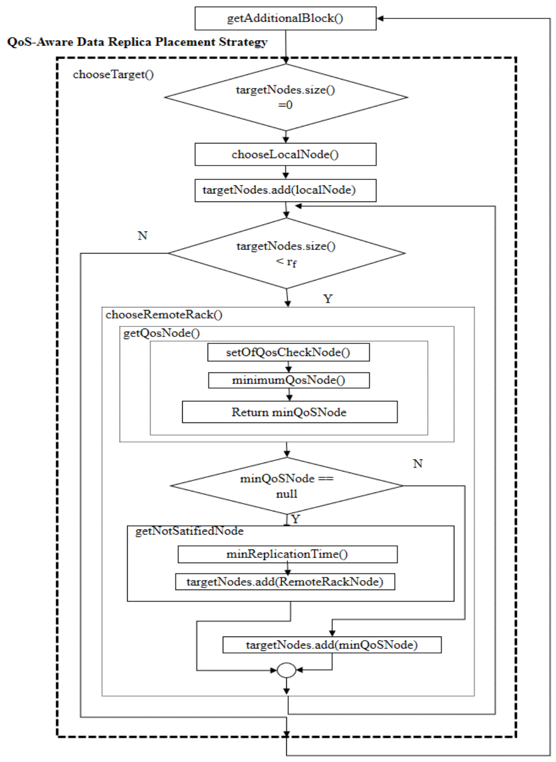


Fig 6. Flow chart of Qos-aware data replication in HDFS

a) *getQosNode() method:* This method is used for selecting a DataNode as a target DataNode among remote DataNodes which satisfy the replication time requirement of requesting application based on operation(2).It returns a 'QoSNode'. Where 'QoSNode' is a DataNode among all the remote rack DataNodes which satisfy(2). Replication time of 'QoSNode'from requested DataNode is minimum among all satisfy DataNodes. This method is called by chooseRemoterack() method by passing remote DataNodes as a argument. getQoSNode() method is described in QoS.java file. This QoS.java file is in org.apache.hadoop.hdfs.server.namenode package.

b) *setOfQosCheckNode() method:*It may be possible number of DataNodes which satisfy the replication time requirement of requesting application based on (1) and (2) are greater than one.This method is responsible to find all DataNodes, which satisfy the replication time requirement of requesting application i.e. $T_{replication}(DR_i, D_j) \leq T(D_i)$. This method is also described in QoS.java file.

c) *minimumQosNode() method:* This method is used for selecting a DataNode which has minimum replication time among all remote DataNodes which satisfy the replication time requirement of application. This method is described in QoS.java file.

d) *minReplicationTime() method:* This method is used for selecting a DataNode as a target DataNode among remote DataNodes which doesn't satisfy replication time requirement of requesting application based on (2) but has minimum replication time among all remote rack DataNodes. If no one DataNode is selected by getQoSNode() method,this method is called by chooseRandom() method. minReplicationTime() method is described in QoS.java file. This QoS.java file is in org.apache.hadoop.hdfs.server.namenode package.

As client sends QoS-aware write request to NameNode, getAdditionalBlock() method is called to obtain address of target DataNodes for storing an additional block 'b' of file 'src'. As shown in fig. 6, this getAdditionalBlock() method calls chooseTarget() method, for selecting target DataNodes from all available DataNodes based on QoS-aware replica placement strategy, where number of target DataNodes is equal to 'r_f' and 'r_f' is replication factor. If number of target DataNodes in 'targetNodes' are zero, chooseLocalNode() method is called. This method selects the local DataNode as a target DataNode. Address of this local DataNode is stored in 'targetNodes'. If number of selected target DataNodes in 'targetNodes' are less than 'r_f', chooseRemoteRack() method is called to find all remote rack DataNodes based on (1). After finding all remote rack DataNodes getQosNode() method is called, which in turn calls setofQosCheckNode() method to select all the remote rack DataNodes which meets the replication time requirement of application based on (2). getQosNode() method further calls minimumQosNode() method that selects one DataNode among the DataNodes returned by setofQosCheckNode(). This selected DataNode 'QoSNode' has minimum replication time from requested DataNode. This target DataNode 'QoSNode' is returned by getQosNode() method to chooseRemoteRack() method.

If no one remote rack DataNode is satisfied the replication time requirement of application based on (2), value of 'QoSNode' is null. If'QoSNode' is not null, it stored in 'targetNodes' by chooseRemoteRack() method otherwise, getNotSatifiedNode() method is called by chooseRemoteRack() method. This method is used to find a targetDataNode which don't satisfy the QoS replication time requirement of application based on (2) but has minimum replication time among all remote rack DataNodes from requested DataNode. To finding target DataNode getNotSatifiedNode() method in turn call minReplicationTime() method to find unsatisfied DataNode which has minimum replication time from requested DataNode. The address of this minimum replication time unsatisfied target DataNode is stored in 'targetNode'.

As number of target DataNodes in 'targetNode' is equal to 'replicationFactor', 'targetNode' is returned by chooseTarget() method to getAdditionalBlock() method. In this way, 'targetNode' contains address of target DataNodes, where block 'b' of a file 'src' has to be stored. Then target DataNodes organizes a pipeline and DataStreamer streams the block to the first target DataNode, second target DataNode and third target DataNode one by one to write in the disk of target DataNodes.

3) Algorithm of QoS-Aware Data Replica Placement Policy:

QoS-aware data replica placement algorithm takes 'src', 'replicationFactor' 'clientNode', 'blockSize', 'replicationTime' as input. Output of this algorithm is the 'result', which holds the locations of target DataNodes where block will be hosted. Selection of target DataNodes is based on QoS-aware data replica placement mechanism. Number of target DataNodes in 'result' is equal to the 'replicationFactor'.

The variables used in 'QoS-aware data replication in HDFS' algorithm are given in table 3.

TABLE 3
Variables used in Qos-Aware Data Replication Algorithm

Variable Name	Description
clientNode	DataNode from where client requests for storing blocks of file.
Src	Name of the file, client wants to write on the DataNodes.
replicationTime	Expected replication time requires by requested application to replicate a data block.
replicationFactor	Number of replicas of data block to be stored on DataNodes.
blockSize	size of each data block stored on DataNode.
result	Array which holds the address of target DataNodes based on QoS-aware data replication strategy.
remoteNode	Set of DataNodes of remote rack.
qosCheckNodes	Set of DataNodes which meets the replication time requirement of requested application.
minQosNode	A DataNode among qosCheckNodes which has minimum replication time from client Node.

When the client sends QoS-aware write request of file 'src' to the NameNode for location of target DataNodes, NameNode calls getAdditionalBlock() method to obtain an additional block of the file 'src'. The size of the block is equal to 'blocksize'. This method calls chooseTarget() method which selects 'replicationFactor' number of target

```
Algorithm: Pseudo code for "QoS-aware data replication" algorithm.
Input: src, clientNode, replicationTime, replicationFactor, blockSize
Output: result
Step 1        chooseTarget() checks the 'result'.
Step 2        If the 'result' contains zero target DataNodes then it
              calls chooseLocalNode() method, which selects a local
              DataNode.
Step 3        This selected local DataNode is stored in 'result'.
Step 4        If number of target DataNodes in 'result' is less than
              'replicationFactor' then it calls chooseRemoteRack()
              method.
Step 4.1      chooseRemoteRack() method selects all remote rack
              DataNodes and stores in 'remoteNode'
Step 4.1.1    chooseRemoteRack() method calls getQosNode() method
              which takes 'remoteNode' as argument.
Step 4.1.1.a  getQosNode() method calls setOfQosCheckNode()
              method which saves DataNodes in 'qosCheckNodes'
              among 'remoteNode' , which meet the replication time
              requirement of requested application.
Step 4.1.1.b  After that getQoSNode() method calls
              minimumQosNode() method which save a DataNode in
              'minQosNode' among all 'qosCheckNodes' which has
              minimum replication time from clientNode.
Step 4.1.1.c  getQoSNode() method returns 'minQosNode' to
              chooseRandom() method. If no 'remoteNodes'
              DataNode meet the replication time requirement of
              application then null is return to chooseRemoteRack()
              method.
Step 4.1.2    chooseRemoteRack() method checks the value of
              'minQoSNode' , returned by getQoSNode() method.
Step 4.1.3    If value of 'minQoSNode' is null then
              getNotSatifiedNode() method is called by
              choosRemoteRack() method which takes 'remoteNode'
              as parameter.
Step 4.1.3.a  getNotSatifiedNode() method calls
              minReplicationTime() to select a remote rack DataNode
              which has minimum replication time from client Node
              but doesn't meet the replication time requirement of the
              requested application.
Step 4.1.3.b  This selected Datanode is stored in 'result'.
Step 4.1.4    else chooseRemoteRack() saves addess of 'minQosNode'
              in 'result' .
Step 5        Go to step 4, till number of target DataNode in 'result'
              is not equal to 'replicationFactor'.
Step 6        'result' is returned.
```

Fig.7. The Qos-aware data replica placement algorithm

DataNodes for placing the block of file 'src'. The selection of target DataNodes are based on QoS-aware data replica placement strategy.

In QoS-aware data replica placement algorithm, chooseTarget() method checks the result variable and finds number of target DataNodes in 'result' variable. If 'result' variable doesn't contain any location of DataNode, it calls chooseLocalNode() method in step2. This method selects the local DataNode as a target DataNode and stores address of local DataNode in 'result' variable in step 3. If number of target DataNodes in 'result' variable is less than 'replicationFactor', it calls chooseRemoteRack() method in step 4 to selectall remote rack DataNode based on (1) and stores it in the 'remoteNode' variable in step 4.1. chooseRemoteRack() method which in turn calls getQosNode() method in step 4.1.1. This getQosNode() method takes 'remoteNode' as argument and calls setOfQosCheckNode() method which selects DataNodes among 'remoteNode'which satisfy the replication time requirement of application based on (2) and stores in 'qosCheckNodes' variable in step 4.1.1.a. After successfully execution of setOfQosCheckNode() method, getQosNode() method calls minimumQosNode() method and passes 'qosCheckNodes' variable as an argument.

setOfQosCheckNode() method finds DataNode among ‘qosCheckNodes’ variable that has minimum replication time from client DatatNodein step 4.1.1.b. Then it stores the selected DataNode in ‘minQosNode’ variable. ‘minQosNode’ variable is returned by getQosNode() method to chooseRandomNode() method. If remote rack DataNodes doesn’t satisfy the replication time requirement of application, null is returned to chooseRemoteRack() method in step 4.1.1.c. If null is returned to chooseRemoteRack() method, getNotSatifiedNode() method is called. Otherwise ‘minQoSNode’ is stored in ‘result’ variable in step 4.1.4. getNotSatifiedNode() method calls minReplicationtime() method to select a DataNode among remote rack DataNodes which has minimum replication time but doesn’t satisfy the (2) in step 4.1.3.a and address of selected DataNode is stored in ‘result’ variable in step 4.1.3.b. chooseremoteRack() method is called till number of target DataNodes in ‘result’ variable is equal to ‘replicationFactor’. If number of target DataNodes in ‘result’ variable is equal to ‘replicationFactor’ variable, result is returned by chooseTarget() method.

In this way, ‘targetNode’ contains address of target DataNodes, where block ‘b’ of a file ‘src’ has to be stored. Then target DataNodes organizes a pipeline and DataStreamer streams the block to the first target DataNode, second target DataNode and third target DataNode one by one to write in the disk of target DataNodes.

4) *Example of QoS-Aware Data Replica Placement Policy*

Consider the same example given in fig. 5, in which applications ‘A1’ ‘A2’ and ‘A3’ are executing on ‘D1’ ‘D2’ and ‘D3’ respectively. Replication factor is also three. Replication time requirement of application ‘A1’, ‘A2’ and ‘A3’ are 0.0520 ms, 0.0750 ms and 0.050 ms respectively.

TABLE 4
Replication time between each DataNode in millisecond

	DataNode1	DataNode2	DataNode3	DataNode4	DataNode5	DataNode6
DataNode1	0.0365	0.0520	0.0535	0.0501	0.0500	0.0490
DataNode2	0.0500	0.0370	0.0750	0.0780	0.0730	0.0725
DataNode3	0.0500	0.0530	0.0380	0.0755	0.0830	0.0860
DataNode4	0.0495	0.0465	0.0530	0.0365	0.0845	0.0810
DataNode5	0.0485	0.0480	0.0470	0.0465	0.0375	0.0735
DataNode6	0.0480	0.0490	0.0530	0.0510	0.0505	0.0390

Replication time from each DataNode to each DataNode is giventable 4. This replication time is calculated by (3). Suppose ‘A1’ issues a QoS-aware write request to NameNode. After receiving QoS-aware write request, NameNode finds target DataNodes based on QoS-aware data replica placement algorithm. First target DataNode is ‘D1’ because ‘D1’ is local DataNode. After that NameNode finds other DataNodes which satisfy the replication time requirement of application. ‘D4’, ‘D5’ and ‘D6’ satisfy the replication time requirement of application because these DataNodes are in different rack as well as replication time of ‘D4’, ‘D5’ and ‘D6’ are 0.501 ms, 0.500 ms and 0.0490 ms respectively from the ‘D1’ client DataNode which is less than replication time requirement of application i.e. 0.0520ms. But among these three DataNodes, NameNode selects ‘D6’ DataNodes because replication time of ‘D6’ DataNode from ‘D1’ requested DataNode is minimum. Similarly ‘D5’ DataNode is selected by NameNode. Hence, address of ‘D1’, ‘D5’ and ‘D6’ DataNodes are returned by NameNode to ‘D1’ requested DataNode. The replication cost of application ‘A1’ is 0.0365 ms +0.0500 ms +0.0490 ms i.e. 1.355 ms

Similarly, the application ‘A2’ and application ‘A3’ issue QoS-aware write request to NameNode. Replication cost of application ‘A2’ and ‘A3’ are 1.825 ms and 1.410 ms respectively.

V. PERFORMANCE EVALUATION

We used Oracle Solaris Zone technology[18] to evaluate the performance of the block replica placement strategy and QoS-aware data replication strategy. Our simulation environment has 2000 DataNodes and 101 racks.

A. Simulation Environment

Oracle Solaris Zones technology is used to configure Hadoop Distributed File System on multiple DataNodes. Oracle Solaris Zones technology follows the concept of virtualize operating system services and provide an isolated and secure environment for running independent or dependent applications.A Zone is a virtualized operating system environment which is created within a single instance of the Oracle Solaris operating system. It allows to create multiple isolated systems in single system. These isolated system are secure from external attacks and internal malicious programs. Oracle Solaris uses Zone File System (ZFS) encryption concept in which all data and file system metadata (such as ownership, access control lists, quota information, and so on) is encrypted when it is stored persistently in the ZFS pool. Each Oracle Solaris Zone contains a complete resource-controlled environment and it allows to allocate resources such as CPU, memory, networking, and storage. In simulation environment, NameNode, SecondaryNameNode and 2000 DataNodes are created using this oracle Solaris Zone. These Nodes are secure, isolated and complete resource-controlled Nodes. It also provides scalable infrastructure which helps to create new DataNodes.

Oracle Solaris also provides Network virtualization with virtual NICs (VNICs), virtual switching and Network resource management. In simulation environment, 101 racks are created using virtual switching. One rack is specified as a central rack for connecting all the remaining 100 racks, in addition 20 DataNodes are connected to each rack.

QoS-aware data replication strategy is implemented as mentioned in part B of section IV by changing the source code of HDFS. Moreover, open source IDE Eclipse is used to change and rebuilt the source code of HDFS. Original HDFS having block replication strategy and rebuilt HDFS having QoS-aware data replication strategy is installed in simulation environment.

In simulation run, we randomly select a DataNodes to execute applications, not all 2000 DataNodes. Application generates a QoS-aware write request to NameNode to write a block of data. NameNode handles this write request. Disk storage time is calculated by tools like hdparm/dd. Next, Communication time is calculated by UDP programor ping command. QoS requirement of application is set by randomly selecting a number from 0.050 to 0.0200.

B. Simulation Testing

QoS-aware Data Replica Placement strategy were developed in Hadoop in such a way that data block is replicated on DataNodes which satisfy the replication time requirement of application. The existing algorithm 'block replica placement' and modified algorithm 'QoS-aware data replica placement' of Hadoop Distributed File System (HDFS) were tested in Oracle Solaris Operating System. This HDFS consists of NameNode, SecondaryNameNode and 2000 DataNodes. Twenty DataNodes were deployed with in each rack. Replication time requirement of different application running on each DataNode is selected randomly.

Different size of files were stored in HDFS for testing. These files are selected in such a way that number of blocks formed by these files were different. Number of blocks formed by first file, second file, third file etc. was one, two, three etc. respectively.

C. Simulation Result

This section presents the results of the testing done in previous section.

Files were stored in Hadoop distributed file system (HDFS). Replicas of block were placed which were based on 'Block replica placement algorithm'. In this algorithm first block was stored on local DataNode, second and third replicas were stored on random remote rack. Replication time of first file was 0.1400 ms, second file was 0.3800 etc.

Similarly, replication of blocks were done by 'QoS-aware data replica placement algorithm'. In this algorithm first block was stored on local DataNode, second and third were stored on DataNodes which satisfy the replication time requirement of application. Replication time of first file was 0.1355 ms, second file was 0.3650 ms etc. Replication time taken for placement of replicas based on Block Replica Placement algorithm and QoS-aware data replica placement algorithm for different files stored on HDFS is shown in table 5.

Table 5
Replication time of blocks based on 'Block Replica Placement Algorithm' and 'QoS-Aware Data Replica Placement Algorithm'

File	No. of blocks	Replication time of file as per Block Replica Placement (ms)	Replication time of file as per QoS-Aware Data Replica Placement (ms)
First file	1	0.1400	0.1355
Second file	2	0.3800	0.3650
Third file	3	0.4890	0.423
Forth file	4	0.6690	0.5300
Fifth file	5	0.6610	0.6580
Sixth file	6	0.8300	0.8160
Seventh file	7	0.9665	0.9485
Eighth file	8	1.4975	1.2385
Ninth file	9	1.5990	1.2945
Tenth file	10	1.6725	1.3285

Replication Cost of Block Replica Placement algorithm was 8.9045 ms. Whereas, Replication Cost of QoS-aware data replica placement algorithm was 7.7375 ms.

D. Analysis of Result

Graph is plotted between number of blocks of different files stored in HDFS and replication time taken to store these files. As shown in fig. 8, replication time to store the block using 'QoS-aware data replica algorithm' is less as compared to 'Block Replica Placement algorithm'.

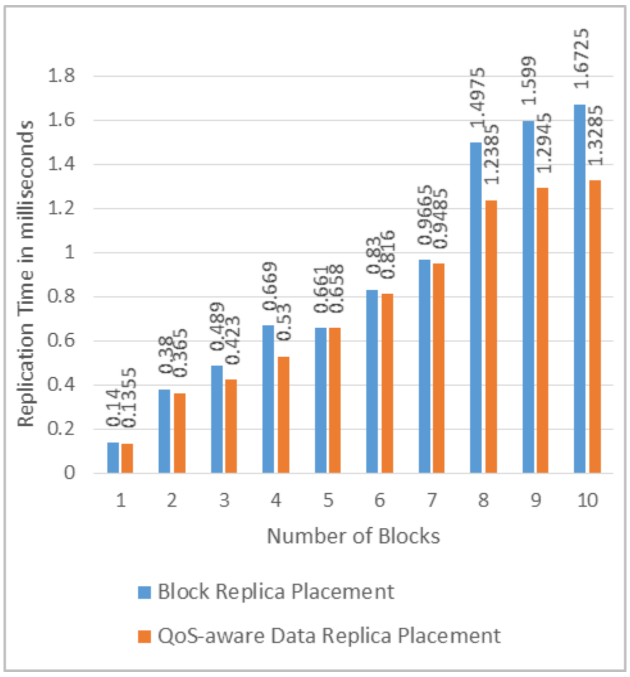


Fig. 8. Graph of Block Replica Placement Algorithm vs. QoS-Aware Data Replica Placement Algorithm

The placement of replica using QoS-aware replica placement algorithm has less replication cost as compared to block replica placement algorithm for approximate 3 GB size of files and 2000 DataNodes. The replication time of larger files based on QoS-aware data replica placement algorithm was improved as compared to block replica placement algorithm as shown in fig. 8. Although storage and number of DataNodes can rise up to petabytes of data and millions of DataNodes. It will give significant result.

In block replica placement algorithm, the selection of target remote DataNode is random. But, in QoS-aware replica placement algorithm, the selection of target DataNode is based on replication time requirement of application. Block is replicated to remote DataNode which satisfies the replication time requirement of application, which reduces the replication cost of algorithm. Hence QoS-aware data replica placement is better than block replica placement algorithm.

VI. CONCLUSION

In QoS-aware Data Replication algorithm, first replica of block was stored on local DataNode while second and third replica were stored on DataNodes which satisfy the replication time requirement of applications. This algorithm was tested on Oracle Solaris operating system using Oracle Solaris Zones technology by storing different size of files. The existing block replication algorithm was compared with QoS-aware Data Replication algorithm. It was observed that replication Cost of 'Block Replica Placement algorithm' and 'QoS-aware data replica placement algorithm' was 8.9045 ms and 7.7375 ms. The replication cost of 'QoS-aware Data Replica Placement algorithm' in HDFS minimizes the data replication cost and improves system performance.

The QoS-aware Data Replication algorithm considers 'replication time' as QoS parameter. Other QoS parameter like access time can also be considered for accessing the data block.

In future, replication algorithms can be extended to concern heat dissipation, power dissipation and temperature of DataNodes. The number of DataNodes in HDFS are large, hence consideration of energy parameter for block placement will be significant. Blocks should be replicated to DataNodes having low temperature and low power consumption.

REFERENCES

[1] http://jehiah.cz/a/distributed-filesystems

[2] https://wiki.apache.org/hadoop/PoweredBy

[3] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System" , conference IEEE 2010.

[4] Apache, HDFS Overview, http://hadoop.apache.org/hdfs/, 2010

[5] Apache, Hadoop Mapreduce, http://hadoop.apache.org/mapreduce/docs/current/index.html,2010

[6] T. White, Hadoop: The Definitive Guide, Second ed., Yahoo Press, 2010

[7] Aiqiang Gao and Luhong Diao, "Lazy Update Propagation for Data Replication in Cloud Computing", 5th International Conference, Pervasive Computing and Applications (ICPCA), pp. 250 – 254, Dec. 2010

[8] Hadoop: The Definitive Guide.

[9] http://www.slideshare.net/cloudera/hadoop-world-2011-hdfs-federation-suresh-srinivas-hortonworks

[10] Apache Zookeeper. http://hadoop.apache.org/zookeeper/

[11] F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li, "Hadoop High Availability through Metadata Replication," Proc. First Int'l Workshop Cloud Data Manage, pp. 37-44, 2009.

[12] QoS-Aware Replica Placement for Content Distribution Xueyan Tang, Member, IEEE, and Jianliang Xu, Member, IEEE

[13] H. Wang, P. Liu, and J.-J. Wu, "A QoS-Aware Heuristic Algorithm for Replica Placement," Proc. IEEE/ACM Seventh Int'l Conf. Grid Computing, pp. 96-103, Sept. 2006.

[14] A.M. Soosai, A. Abdullah, M. Othman, R. Latip, M.N. Sulaiman, and H. Ibrahim, "Dynamic Replica Replacement Strategy in Data Grid," Proc. Eighth Int'l Conf. Computing Technology and Information Management (ICCM), pp. 578-584, Apr. 2012.

[15] X. Fu, R. Wang, Y. Wang, and S. Deng, "A Replica Placement Algorithm in Mobile Grid Environments," Proc. Int'l Conf. Embedded Software and Systems (ICESS '09), pp. 601-606, May 2009.

[16] M. Shorfuzzaman, P. Graham, and R. Eskicioglu, "QoS-Aware Distributed Replica Placement in Hierarchical Data Grids," Proc. IEEE Int'l Conf. Advanced Information Networking and Applications, pp. 291-299, Mar. 2011.

[17] Apache Hadoop. [Online] [2014].
Available:
http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[18] Oracle Solaris Documentation [Online]
Available: http:// http://www.oracle.com