A Dynamic File Replication and Consistency Mechanism for Efficient Data Center Operation and its Formal Verification

Manu Vardhan, Dharmender Singh Kushwaha

Department of Computer Science and Engineering
Motilal Nehru National Institute of Technology Allahabad, India
{rcs1002, dsk}@mnnit.ac.in

Abstract: Data centers are built to provide a highly available and scalable infrastructure on which the applications run. As enterprises grow, so do their application need, along with resources required for additional application-specific services. This increase in bottom line expense heightens the overall resource requirement. This paper provides solution to mitigate the impact of these expenses by proposing a file replication and consistency maintenance mechanism that enhances the manageability, scalability, and high availability of resource in these environments. To keep files consistent, changes made at one replica of the file are reflected on other replicas in minimum possible time. File replica is updated on-demand by only propagating the required partial updates. The results show that as compared to Google File System (GFS), the proposed partial write rate shows an improvement of 38.62% while updating the stale replicas. Time required for invalidating the replicas decreases by 34.04% and the time required to update the replica on FRS reduces by 61.75%. Process algebraic approach has been used to establish the relationship between the formal aspect of the file replicating server and its architectural model.

Keywords: Cluster Computing; Consistency; File Replication; Process algebra

1 Introduction

Data centers are emerging and finding acceptability due to convergence of several trends including the high performance microprocessors, high speed network, standard tools and availability with economical commodity-off-the-shelf components. This comprises of several servers and networking infrastructure. The server portion of the infrastructure is now far down the road of commoditization, and low-cost servers have replaced the high-end enterprise-class servers. Driven by the PC commoditization economics, the operating theme is “scaling out instead of scaling up”. Thus, data centers seem to gain popularity day by day. Data center based distributed systems provide a cost-effective solution to applications intended for High Performance Computing (HPC) [1]. It leads to the evolution of powerful Computer Supported Cooperative Working (CSCW) [2] environment that enables improving availability of resources and load sharing.

Replication is a practical and effective method to achieve efficient file access, and increasing the file availability. File replication is done to achieve high availability of resources. It is achieved by replicating the file and redirecting the requests of busy nodes to lighter ones. Optimal performance is achieved by replicating the resource among different clusters. This helps in reducing file access latency, and network congestion in order to enable the system to handle more requests. Replication of files and replica placement demand an effective and optimized replication approach so that neither remote nor local file request is dropped. Replicating critical data serves as the basis for disaster recovery.

The proposed File Replication approach tries to resolve the following issues: *(1)* Prevent the creation of file replica if a copy of the requested file is available on the other node. *(2)* In case of node failure, file request is redirected to other FRS, without any user intervention. *(3)* Limit the number of queries for a particular file below the threshold. *(4)* New request for a particular file will never be served, once the threshold value for that particular file has been reached. *(5)* Avoids unnecessary file replication. *(6)* Maintains consistency of replicated files.

Every node has an optimal capability of handling file requests. If the file request count for a particular file on any FRS reaches the threshold value, it replicates the file on other FRS's. The location of new replica is intimated to the requesting node. This maximizes the resource utilization by minimizing the message exchanges overhead, thus increasing the overall system performance.

Replication becomes mandatory in cluster computing, whenever there is an increase in number of requests (that a system can handle), for a particular resource. Buyya [4], discusses the major performance issue in large-scale decentralized distributed systems, such as grids, along with mechanisms to minimize latency in the presence of resource performance fluctuations. Buyya [5] addresses the problem of transferring huge amount of data among federated systems, thus facilitating a better way to support critical applications while minimizing the total number of rejected requests. Google's MapReduce [6] system runs on top of the Google File System [7], within which data is loaded, partitioned into chunks, and each chunk is replicated. Google by default replicates the data at three locations. The Google File System (GFS) enables the files to be moved transparently in order to balance the load that is in line with the proposed File Replication. Unlike GFS, proposed approach avoids the creation of redundant replicas on the same node and consumes less raw storage than GFS. By default GFS stores, three replicas of a file, but proposed approach creates replicas as and when demand arises. Proposed approach reclaims the physical storage only when the need arises, in case there is not sufficient storage space to store a file being replicated. GFS does this lazily during regular garbage collection. In contrast to the system like xFS [8], AFS [9], Intermezzo [10] and Frangipani [11], GFS and the approach proposed in this paper, does not provide any caching below the file system interface. Unlike GFS, in order to increase reliability and gain flexibility proposed approach does not maintain any centralized master replica for maintaining file consistency and management. Rather it uses a distributed architecture that manages the assignment of the role of master replica to the latest updated replica dynamically and propagates the same to the secondary replicas on-demand. This overcomes the issue of single point failure. Scalability and high availability (for read) are achieved by adding new servers as and when the need arises without affecting the current ongoing processing. Like proposed approach, GFS also addresses a problem similar to Lustre [12] in terms of delivering aggregate performance to a large number of clients. Bigtable [13] relies on a highly-available and persistent distributed lock service called Chubby [14]. A Chubby service consists of five active replicas, one of which is elected to be the master and actively serve requests. The service is live when a majority of the replicas are running and can communicate with each other. Chubby uses the Paxos algorithm [15], Lamport [16] to keep its replicas consistent in the face of failure. Each directory or file can be used as a lock, and reads and writes to a file are atomic. The Chubby client library provides consistent caching of Chubby files. Each Chubby client maintains a session with a Chubby service. A client's session expires if it is unable to renew its session lease within the lease expiration time. When a client's session expires, it loses any locks and open handles. Chubby clients can also register callbacks on Chubby files and directories for notification of changes or session expiration. The master is responsible for assigning tablets to tablet servers, detecting the addition and expiration of tablet servers, balancing tablet-server load, and garbage collection of files in GFS. Bigtable clients do not rely on the master for tablet location information; most clients never communicate with the master. As a result, the master is lightly loaded in practice. Primary-copy (master-slave) approach for updating the replicas says that only one copy could be updated (the master); secondary copies are updated by the changes propagated from the master. There is only one replica, which always has all the updates. Consequently the load on the primary copy is huge. Domenici [17] discusses several data consistency solutions, including Eager (Synchronous) replication and Lazy (Asynchronous) replication, Single-Master and Multi-Master Model, and pull-based and push-based. Guy [18] proposes a replica modification approach where a replica is designated either a master or a secondary. Only master replica is allowed to be modified whereas secondary replica is treated as read-only, i.e., modification permission on secondary replica is denied. A secondary replica is updated in accordance with the master replica if master replica is modified. Sun [19] proposes two coherence protocols viz., Lazy-copy and Aggressive-copy. Replicas are only updated as needed if accessed by the Lazy-copy protocol. For the aggressive-copy protocol, replicas are always updated immediately when the original file is modified. Compared with Lazy-copy, access delay time can be reduced by Aggressive-copy protocol without suffering from long updating time during each replica access. However, proposed approach has simplified the problem significantly by focusing on the application needs rather than building a POSIX-compliant file system. Most of the replication strategies are capitalized by

GFS, but there are some areas that leave ample scope for future work as discussed in this work. The next section discusses the proposed approach.

In order to ascertain architecture and framework of the proposed model, modeling and evaluation is necessary, before implementing a system to a large scale. The dynamic but complex behavior of the proposed model is analyzed by underlying communication protocol and characteristics of their components. The proposed model is verified through Calculus of Communicating System (CCS).

CCS [3] is a formal language for describing patterns of interaction in the concurrent systems. It allows the description of system in terms of sub processes that include primitives for describing composition and interaction among these, through message passing. Therefore, the motivation for using process algebra is to simplify the specification part and to verify the design structure of model while meeting its ultimate goal, i.e., file replication.

2 Proposed Approach

A scenario of three participating Cluster is considered. These clusters could be part of one organization or three different private clusters of a different organization. It is assumed that the nodes in these different clusters trust each other. The same is illustrated in Fig. 2. The proposed architecture is discussed below.

2.1 Architecture

A scenario is presented, though on a smaller scale where geographically disparate clusters interact with each other for information sharing through replication. One node in each cluster is designated as File Replicating Server (FRS). FRS can also be replicated on some other node in the cluster for backup and recovery. It consists of loosely coupled systems, capable of providing various kinds of services like replication, storage, I/O specific, computation specific and discovery of resources. Based on the application requirement, the resources are made available to other nodes.

2.2 Architecture Description

Fig. 1 shows a mini data center where each server is catering its services to the connected workstations. These workstations can be reorganized so as to form a cluster of nodes as shown in Fig. 2; it shows a network of three clusters that are connected to each other via intercommunication network. Each cluster consists of a group of trusted Requesting Nodes (RN) and a File Replicating Server (FRS) assigned to these nodes. Each node can presume the role of FRS. A FRS can be *local* or *remote*.

A file Replicating Server (FRS) assigned to a cluster having some nodes is known as local FRS, and FRS positioned outside that cluster, will be called as remote FRS. Each subset of nodes (denoted as requesting nodes) receives the list of IP-address of remote FRS's from the local FRS, but the nodes of a cluster will send the file request only to the local FRS. In case, if the local FRS fails to serve the request, it is automatically routed to a remote FRS in a transparent manner, and the remote FRS fulfills the request of the requesting node. This makes the model robust and capable of handling crashes in case local or even remote FRS fails. Each FRS maintains two tables: *(1)* File request count table with the following attributes: <file_id, file_name, request_count, metadata>. *(2)* FRS table with the following attributes: <FRS_IP, FRS_PORT>.

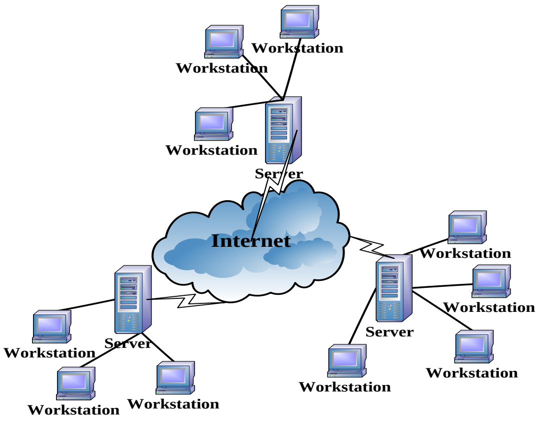

Figure 1
An abstract view of data center

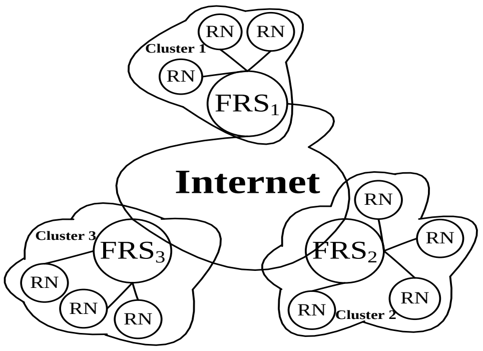

Figure 2
Architecture of file replicating server

Each FRS is informed whenever a new FRS is added to the network, so as to update its FRS table. FRS does not monitor the status of remote FRS periodically, instead FRS requests for the current status of remote FRSs on-demand. FRS status can either be *busy, ready or File not available*. *Busy* signifies that the *request count* is greater than file threshold value. *Request count* is described in section 3.5. *Ready* signifies that the *request count* is less than file threshold value. *File not available* indicates that the requested file is not available on the cluster. Number of thread a server can handle, and the server load balancing aspects are left on the application developer and deployment environment.

2.3 Description of File Replication Mechanism

Each local FRS is responsible for accepting the file request from the Requesting Node (RN). Local FRS checks its status against the requested file and redirects the request depending on its status in the following manner: *(1)* If the status of local FRS is *ready*, the local FRS will fulfill the request. *(2)* If the status of local FRS is *file not available*, the local FRS looks for remote FRS that can fulfill the file request as discussed below in 2(a) & 2(c). Otherwise, *(3)* In case, status of local FRS is *busy*, it looks for a remote FRS that can handle the request, by one of the following manner, described as under:

(a) Local FRS sends a message only to those FRSs that are present in the *replica location* field of the data structure for file request count table (Table 1) and requests for their status against the requested file. The local FRS redirects the request to the remote FRS having the status as *ready*. This remote FRS fulfills the request of the RN. (b) If not so, the local FRS contacts those remote FRS's on which the requested file is not available. In this case, file replication will be initiated, by the local FRS of the cluster and the file replica is created on the remote FRS having the status as *file not available*. (c) For both the cases mentioned above, IP address of the remote FRS that can handle the request will be sent by the local FRS to the requesting node. Now, the request is redirected to the remote FRS and RN shall receive the file, without any user intervention. Thus, the overhead of polling and broadcasting is reduced.

2.4 File Replication Strategy

Fig. 3 shows each FRS as a part of different cluster. To differentiate between the remote FRS and local FRS, dotted and solid lines are used. All FRS are logically interconnected with each other and update their FRS table as soon as a new FRS is added. Node N_1 is the Requesting Node (RN) that sends the file request to FRS. Fig. 3 shows the replication scenario for a file replicating server S_1.

FRS S_1 on successfully connecting to N_1 sends the list of remote FRS (S_2, S_3, S_4 ….S_{i-1}, S_i) to node N_1. Now, node N_1 sends file request to the local FRS, i.e., S_1. As FRS S_1 has reached the threshold, it cannot handle this request. FRS S_1 looks for a remote FRS that can handle the request. While looking for a FRS that can fulfill the request, some FRS's send their status as *busy,* and the rest of the FRS may not hold the requested file, i.e., file not available (*fna*). Now, local FRS will initiate the replication process for the requested file on the remote FRS. Replica will always be created on the FRS that does not contain the requested file. On successfully creating the replica of the requested file on a remote FRS, local FRS will send the IP-address of remote FRS to N_1 and request gets redirected and fulfilled by remote FRS.

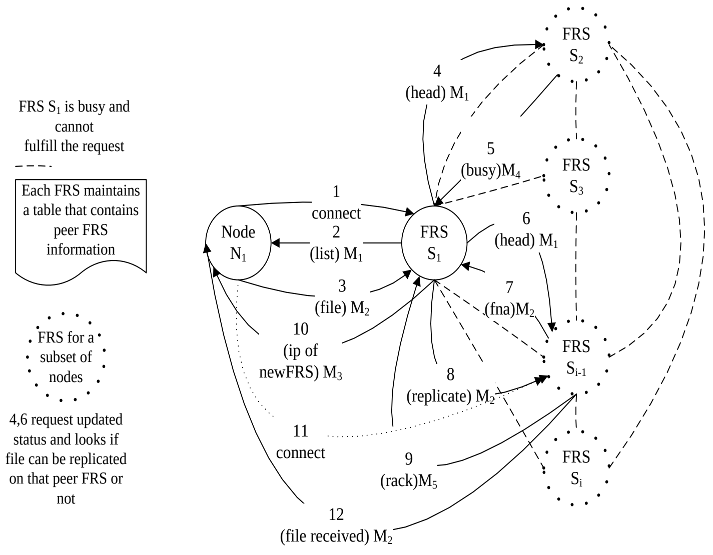

Figure 3
Replication scenario of a file replicating server

2.5 Data Structures Used

Each node maintains Table 1 to handle dynamic on-demand file replication and consistency mechanism. Table 1 keeps the following information about the files requested by the requesting nodes: *(1) File ID:* uniquely identifies the requested file. *(2) Filename:* name of the file requested by the node. *(3) Request Count:* is the number of request for a particular file that the FRS is currently handling. This count is incremented by one, whenever the FRS initiates the file transfer operation that is intended for the requesting node. As soon as the request is fulfilled this count is decremented by one. *(4) Metadata:* stores the data that identify the various file attributes. *(5) Valid*: It is a Boolean variable that signifies whether the file is stale or updated. *(6) Lock*: It is an integer variable that signifies that a FRS has acquired lock on the file and the file is being modified. (7) *Owner FRS*: It is a string variable. This field contains the IP address of the FRS that has most recently modified the file. *(8) Replica Location*: It is an integer variable. This specifies the node ID (FRS and requesting node) on which the file replica is present. *(9) Timestamp* (*t_f*): It is a string variable. It stores the timestamps of the file that is present on the node (FRS and requesting node).

Table 2 represents the format of the table maintained by each FRS, which contains the <IP address and port number> of FRS. FRS IP denotes the IP address of the FRS and FRS PORT denotes the port number of the FRS to which the network messages are forwarded.

2.6 Message Definitions for Proposed Approach

The proposed approach consists of following messages viz., M_1, M_2, M_3, M_4 and M_5. M_1 is request for sending or receiving a file. It consists of three tuples, which include the following details: *(1)* Machine Type (either Requesting Node or FRS) *(a)* Requesting Node requests a file from the file replicating server (FRS). *(b)* FRS uses the *head message* to initiate the replication request. *(2)* Request Type (either "*get*" or "*put*" or "*list*"), list will provide the IP address of the remote FRS. *(3)* Filename. Message M_2 Copies the file or list contents from FRS to Requesting Node or other FRS, in accordance with the type of request, i.e., whether file request is made by FRS for file replication or by requesting node. M_3 responds to the request based on the local FRS current status, namely: *(1)* It informs the requesting node if the local FRS is ready to serve the request, i.e., N_{ready}. Or, *(2)* If the local FRS is busy, it sends the IP and port address of the remote FRS having the requested file. M_4 informs the local FRS about the current status of the remote FRS, namely: *(1) Busy*: remote FRS is busy, so it cannot handle the current request. *(2) Ready*: file is present and remote FRS is ready to serve the request. *(3) File Not Available*: if the file does not exist on the remote FRS, this remote FRS will become the potential node for file replication. M_5 is reply acknowledgement message, i.e., "RACK" is sent to the local FRS, when file is transferred completely. M_6 is a multicast message for sending a request for the modification file to owner FRS. M_7 sends the modification file to those FRSs that has the replica of the modified file f. M_8 is the ACK message send by remote FRSs to FRS_i. M_9 grants permission to node N_i to modify the file.

File Replication approach discussed in section 3.4, 3.5 and 3.6 ensures that, implicit addressing is used, to fulfill the nodes request, for a logical resource and maintains the access, migration and performance transparency of the system.

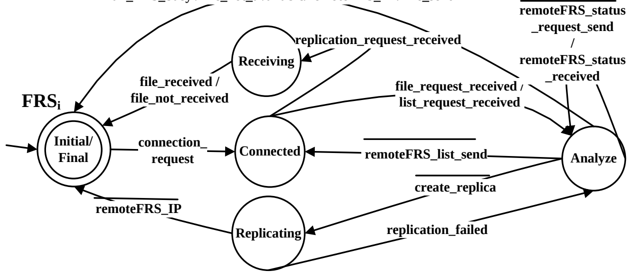
all_FRS_busy / file_not_available/remoteFRS_IP / file_send
remoteFRS_status _request_send / remoteFRS_status _received
replication_request_received
Receiving
file_received / file_not_received
FRSi
file_request_received / list_request_received
Initial/ Final
connection_ request
Connected
remoteFRS_list_send
Analyze
create_replica
remoteFRS_IP
Replicating
replication_failed

Figure 4

File replicating server (FRS) model

The various states of the file replication model are shown in Fig. 4. Here, the state *Connected* represents the state after the connection has been established between the Requesting Node (RN) and FRS_i. FRS_i can change its state either to *Receiving* or *Analyze* as follow: *(a)* In *Connected* state FRS_i accepts the file request, requested by RN and changes its state from *Connected* to *Analyze*. Also, FRS_i becomes the destination FRS if a remote FRS needs to replicate a file on this FRS_i. FRS_i on receiving the replication request changes its state from *Connected* to *Receiving*. After the state *Receiving* has been reached, the transition will only be made to *Final* state, which indicates that the connection has been closed. *(b)* FRS_i on changing its state from *Connected* to *Analyze,* depending on the FRS (local and remote) status the transition will take place either to *Final* state or *Replicating* state. In the state *Replicating*, if failure signal is received, the transition will be made to *Analyze* state, and other remote FRS will be selected for replication. When no failure signal is received, transition will be made to *Final* state.

The process algebraic approach is used to verify the correctness of proposed file replication model. It is a mathematical technique used for the verification of software and hardware systems. This is required, to confirm whether the proposed model is meeting the specifications or not.

2.7 Hybrid Consistency Mechanism Using Partial Update Propagation

The proposed consistency mechanism is called hybrid because the replica of the modified file is updated on FRS using partial update propagation and the write invalidate message is send to the requesting nodes having the replica of the modified file, as shown in Fig. 6. A Requesting Node (RN_i) requests to modify a file (f) present on File Replicating Server (FRS_i). It is assumed that the clocks of all FRSs are synchronized with each other, and all RNs synchronize their clocks with local FRS. In partial update propagation, write update is performed using modification file on FRS's and these FRS's perform write invalidate on RN's. During write update using modification file, owner FRS sends the modification file only to those FRSs that has the replica of the modified file. Now, each FRS on which the replica has been updated, will send an invalidate message to those requesting nodes that have the replica of the modified file.

Owner node is any FRS_J that has most recently modified the file (f) and contains the latest updated (valid) copy of file f, and FRS_J is not a centralized entity. A modification file contains the changes that have been performed on the original file. These changes include the line number on which the change has been made and the content of that line.

When a file on any FRS_i has to be modified by a RN, this FRS_i generates a request to acquire the lock on file (f). If FRS_i is the owner of file (f), it performs a check whether or not file (f) is locked by any other RN on FRS_i. If yes, it waits for the write operation to get completed. Once the write operation is completed, lock is released and the write permission is granted to the requesting node RN_i. If FRS_i is not the owner of file (f), FRS_i multicasts a message for acquiring the lock on file (f) called RW(f) message, to other FRSs. FRS_i multicasts the message only to those FRSs that are present in the *replica location* field of the data structure. On receiving this message if FRS_J has not locked file (f) it sends an acknowledgement for acquiring lock to FRS_i. If FRS_J has locked file (f), it waits for completion and responds by sending modification files. FRS_i updates the stale copy of (f) by patching it with the modification file. FRS_i acquire write lock on file (f) and gives write permission to requesting node RN_i. After modifying the file, the RN will update the file on its local FRS_i by sending the modification file. Now FRS_i will update the file using Hybrid consistency mechanism and becomes the owner of the file. Other FRSs makes an entry in the *Owner* field of the data structure that new file owner of file (f) is FRS_i. Now, these FRSs in turn send an invalidate message to the RN's having the replica of the modified file. If any of the RN's have to use the file later, these RN update their replica by sending a request for modification files to its local FRS as and when the need arises. Flow diagram is shown in Fig. 5.

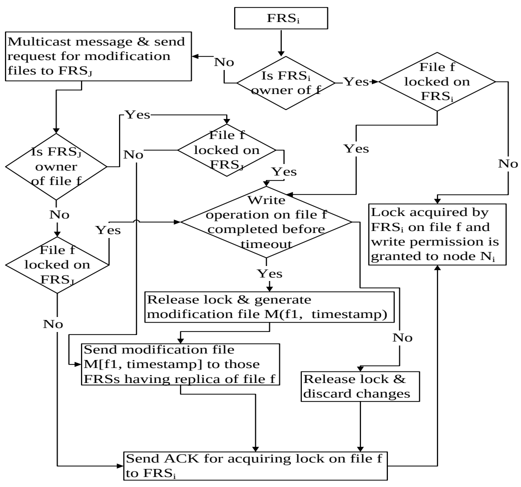
FRSᵢ
Multicast message & send request for modification files to FRSJ
No
Is FRSᵢ owner of f
Yes
File f locked on FRSᵢ
Yes
No
Is FRSJ owner of file f
No
Yes
File f locked on FRSJ
Yes
Write operation on file f completed before timeout
Yes
Lock acquired by FRSᵢ on file f and write permission is granted to node Nᵢ
File f locked on FRSJ
Yes
No
Release lock & generate modification file M(f1, timestamp)
No
Send modification file M[f1, timestamp] to those FRSs having replica of file f
Release lock & discard changes
Send ACK for acquiring lock on file f to FRSᵢ

Figure 5

Flow diagram for acquiring lock by FRS on the file and replica update & consistency maintenance mechanism on FRS using modification files

As discussed above, in case the file is modified, the replica is updated or invalidated on other nodes (FRSs and requesting nodes) using the above discussed hybrid consistency mechanism. Depending on the number of FRS on which the replica is updated, and the number of requesting nodes on which the replica is invalidated there arises two cases as follow: *Best case:* When only few FRS and requesting nodes have the replica of the file that has been modified file. *Worst case:* When all the FRSs and the requesting nodes have the file replica of the modified file, this is considered as the worst case scenario.

Consider a scenario as shown in Fig. 6, in which the file replica is present on FRS_1, FRS_2, FRS_3, RN_3, RN_6 and RN_9. Now, RN_9 makes a request to modify a file present on FRS_3. FRS_3 checks whether or not the requested file is locked by some other RN using the file locking mechanism, as discussed above. Fig. 6 shows the hybrid consistency mechanism in which the file is updated on those FRSs having the replica of the modified file by using the modification file and these FRSs in turn send the invalidate message to the requesting nodes having the replica of the file that has been modified.

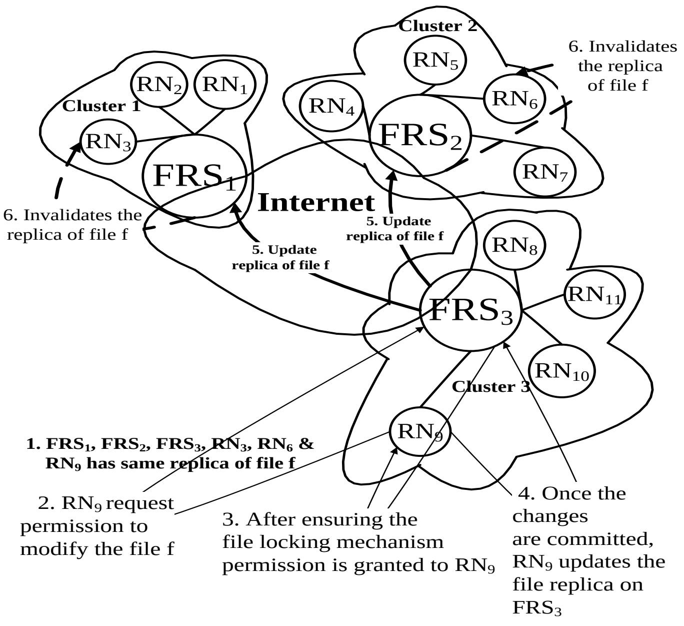
Cluster 1
Cluster 2
Cluster 3
RN2
RN1
RN3
FRS1
RN4
RN5
RN6
RN7
FRS2
Internet
RN8
RN11
FRS3
RN10
RN9
6. Invalidates the replica of file f
6. Invalidates the replica of file f
5. Update replica of file f
5. Update replica of file f
1. FRS1, FRS2, FRS3, RN3, RN6 & RN9 has same replica of file f
2. RN9 request permission to modify the file f
3. After ensuring the file locking mechanism permission is granted to RN9
4. Once the changes are committed, RN9 updates the file replica on FRS3

Figure 6

Hybrid consistency mechanism

If the owner FRS_J is down, FRS_i selects a remote FRS from the *Replica location* field of Table 1 as described in section 3.5 and checks the validity of file on that remote FRS. File validity is checked from the *Valid* field of Table 1. As soon as validity of file on the remote FRS is confirmed, that remote FRS becomes the new owner of that file.

3 Stability Analysis

Stability analysis of File Replication Model (FRM) using a process algebraic approach is carried out in this section.

3.1 Transition System Definition

Transition systems [20] are considered to perform external and internal actions. External actions are defined as observable actions, which are seen by the observer. However, an unobservable action is considered as an internal action which the observer cannot see. According to Milner [21], an agent C is a cell which may hold a single data item. It has two ports; an empty cell may accept an item or value from its left hand port labeled in; while it may deliver a value from its right hand port labeled [22] as shown in Fig. 7.

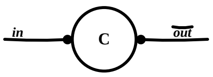

Figure 7
Labelled transition

3.2 File Replication Model (FRM) in Process Algebraic Framework

In the proposed formal model, the components of a system are identified as communicating agents. In the flow diagram as shown in Fig. 8, circle represents the communicating agents, i.e., Requesting Node (RN), i.e., N_{fr} and file replicating server (FRS). These are considered as communicating agent. The communication between agents is represented through transition graph. Transmission line (*trans*) is used to transfer messages from one node to other. The FRS receiving the file request (M_1) is termed as File Replicating Server (FRS), i.e., N_{frs} and is denoted by S_i. FRS S_i^r or S_{i-1}^r is the server on which file replica is either created or previously available. The RN, i.e., N_{fr} raises a file requisition via message $\overline{M_1}$ and receives its corresponding reply via M_2 or M_3, depending on the FRSs status. $M_4/\overline{M_4}$ shows the status of FRS as busy (N_{busy}) or ready (N_{ready}). Status of FRS depends on the number of request a FRS is currently serving for a meticulous file. FRSs having the status as busy cannot fulfill the file request. Ready FRS (N_{ready}) represents that FRS is ready to handle the file request. In case FRS S_i is busy, it requests the status information of remote FRSs and redirects the request to the FRS having the status as ready (N_{ready}) that can handle the request. If no such FRS is present, replica is created on FRS S_i^r or S_{i-1}^r that has the status as N_{fna}, i.e., file not available.

After the file is replicated on FRS S_i^r or S_{i-1}^r an acknowledgement message M_5 or $\overline{M_5}$ is sent by FRS S_i^r or S_{i-1}^r to FRS S_i, the file request gets redirected and fulfilled by FRS S_i^r or S_{i-1}^r.

The value of file threshold index prohibits the behavior of nodes as busy and ready. Similarly, the file availability index gives information about the file availability on that node. Here arise two scenarios, discussed as below: *(1)* Replica available *(2)* Replica created.

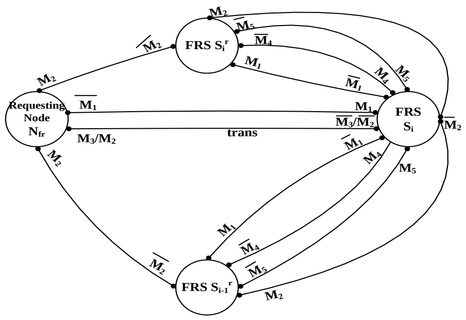

Figure 8

Flow graph for file replication model in process algebraic approach

Replica Available: FRS S_i, which receives the file request from the requesting node, fulfills it if and only if, its status is *ready*, i.e., N_{ready}. This means that, the requested file is present on S_i and it can handle the file request. If S_i is not able to fulfill the request, it looks for any S_i^r or S_{i-1}^r that has the requested file and status as N_{ready}. In this case, there is no need to create a replica of the requested file. *Replica created:* if all FRS, i.e., S_i, S_i^r or S_{i-1}^r having the copy of the requested file are busy, then its replica needs to be created. Replica will always be created, on the FRS S_i^r or S_{i-1}^r whose status is N_{fna}, i.e., file not available. The formal specification of two scenarios is described below:

Set of agents can be denoted by ϑ. Hence,

$\vartheta \in \{N, N_{fr}, N_{frs}, S, D, FRS\}$

Equ. (1) shows different status of FRS:

$$N_{status} \overset{\text{def}}{=} (N_{busy} \mid N_{fna} \mid N_{ready}) \quad (1)$$

FRS status depends on the number of requests a FRS is currently serving for the requested file known as file request load. If the file request load >= the file threshold value, the requested file is replicated on remote FRS, i.e., S_i^r or S_{i-1}^r.

Equ. (2), a node requesting for a file, hence called as Requesting Node (N_{fr}) denoted by S.

$$\mathrm{N}_{\mathrm{fr}} \overset{\text{def}}{=} \mathrm{S} \quad (2)$$

Equ. (3), FRS S_i receives file request. S_i responds to the requesting node via message $\overline{\mathrm{M}_3}$ or $\overline{\mathrm{M}_2}$, depending on the FRSs status, the state transition will take place.

$$N_{frs} \overset{\text{def}}{=} (file\_request\_received).(\overline{frs\_status} + \overline{file\_send}).N_{frs} \quad (3)$$

Equ. (4), FRS that affirms its status as N_{ready} will fulfill file request and is denoted by *D*.

$$N_{ready} \overset{\text{def}}{=} \mathrm{D} \quad (4)$$

$$D \overset{\text{def}}{=} file\_request\_received.\overline{file\_send}.D \tag{5}$$

$$S \overset{\text{def}}{=} \overline{file\_request\_send}.file\_received.S \tag{6}$$

According to Robin Milner [21] a labeled transition system can be thought of as an automaton without any Initial/Final state.

3.2.1 Language Specification for Replica Available (FRM_{ra}) Scenario

Now, a scenario of file replication model, i.e., replica available is discussed. Here, the file request is fulfilled by the available copy of the requested file on a remote FRS. Thus, unnecessary replica creation is avoided.

3.2.1.1 Language Specification for Local FRS

The language specification for the local FRS that can be described as follow: *(1)* State transition takes place from one state to the other depending upon the interrupt received by the current state. *(2)* After receiving the file request from the requesting node, the local FRS S_i fulfills the request if and only if its' status is N_{ready}. Otherwise, S_i checks with the remote FRS, i.e., S_i^r or S_{i-1}^r that can handle the request. *(3)* If a remote FRS is busy, it will not accept the file request and sends its status as busy to the local FRS S_i. *(4)* If S_i status is not marked as ready, this means, either S_i status is N_{fna} or N_{busy}, refer to equ. (8). In case, those FRSs having the replica of the requested file are busy, requested file is replicated on that remote FRS having the status as N_{fna}. *(5)* State LocalFRS` is reached after the connection with requesting node is closed.

$$LocalFRS \overset{\text{def}}{=} (list\_request\_received + file\_request\_received).Analyze \tag{7}$$

$$\begin{aligned} &\text{Analyze} \overset{\text{def}}{=} \\ &\overline{\text{remoteFRS\_status\_request}}.\text{Analyzing} + \overline{\text{send\_remoteFRS\_list}}.\text{LocalFRS} + \\ &(\overline{\text{all\_FRS\_busy}} + \overline{\text{file\_not\_available}} + \overline{\text{file\_send}}).\text{LocalFRS}' \end{aligned} \tag{8}$$

$$\begin{aligned} &\text{Analyzing} \overset{\text{def}}{=} \\ &(\overline{\text{remoteFRS\_busy}} + \overline{\text{remoteFRS\_fna}}).\text{Analyze} + \overline{\text{remoteFRS\_IPaddress}}.\text{LocalFRS}' \end{aligned} \tag{9}$$

3.2.1.2 Corresponding Language Specification for Requesting Node

The behavior of the requesting node that sends the request to FRS for a file is represented by equ. (10). Requesting node after sending the file request to FRS, changes its state from N_{fr} to N_{fr}'. In this state, the requesting node will wait for the reply from the FRS S_i. Once the response is received the transition is made to state N_{fr}''.

$$\text{N}_{\text{fr}} \overset{\text{def}}{=} (\overline{\text{list\_request\_send}} + \overline{\text{file\_request\_send}}).\text{N}_{\text{fr}}' \tag{10}$$

$$\begin{aligned} &\text{N}_{\text{fr}}' \overset{\text{def}}{=} (\text{list\_received} + \text{file\_received}).\text{N}_{\text{fr}} + (\text{remoteFRS\_IP} + \text{all\_FRS\_busy} + \text{file\_not\_available} + \\ &\text{remoteFRS\_list\_not\_received}).\text{N}_{\text{fr}}'' \end{aligned} \tag{11}$$

$$N''_{fr} \overset{\text{def}}{=} (\text{requesting\_node\_crashes} + \text{no\_remoteFRS\_available}).\, N''_{fr} \quad (12)$$

$$FRM_{(ra)} \overset{\text{def}}{=} \text{RequestingNode} \parallel \text{FRS} \parallel \text{Destination} \quad (13)$$

3.2.2 Language Specification for Replica Created (FRM$_{rc}$) Scenario

Now, second scenario of file replication model, i.e., replica creation will be discussed. It symbolizes a communicating system that consists of *Replicate* and *Receiving* agents, which represents the replication mechanism of file replicating server. Replicate agent is the FRS that creates the replica of the file on remote FRS. This remote FRS is known as receiving agent. *Replicate (F):* A file request is sent through the transmission line (*Trans*) by the requesting node and it is received by FRS. On receiving the request, FRS changes its state to *Replicate*, which denotes that the FRS is busy, and the requested file needs to be replicated, refer to (14) & (15). *Receiving (F):* The receiving agent, i.e., remote FRS that receives file request through the transmission line (*Trans*) and reaches the state FRS', on successful completion of file transmission. *FRS':* state FRS' is the final state after the file transfer is complete.

$$FRS \overset{\text{def}}{=} \text{replication\_request\_received}.\, \text{Receiving} + \text{file\_request\_received}.\, \text{Replicate} \quad (14)$$

$$\text{Replicate} \overset{\text{def}}{=} \overline{\text{replication\_request\_send}}.\, \text{Replicating} \quad (15)$$

$$\text{Receiving} \overset{\text{def}}{=} \text{replica\_created}.\, FRS' \quad (16)$$

$$\text{Replicating} \overset{\text{def}}{=} \text{replication\_failed}.\, \text{Replicate} + \overline{\text{replica\_created}}.\, FRS' \quad (17)$$

$$FRM_{(rc)} \overset{\text{def}}{=} \text{Replicate} \parallel \text{Trans}(\varepsilon) \parallel \text{Replica\_created} \quad (18)$$

Where Trans (ε) denotes initially empty transmission line and $\parallel$ denotes restricted composition. From equation (13) and (18), it is proved that replication mechanism of file replication model meets its specification with FRM$_{(ra)}$ and FRM$_{(rc)}$. Hence, FRM$_{(ra)}$ ≈ FRM$_{(rc)}$.

3.2.3 Bisimulation Proof for FRM$_{(ra)}$ and FRM$_{(rc)}$

The transition graph for local FRS and requesting node, refer to equ. (5) and (6). Equation shows the different FRS states (14), Replicate (15), Replicating (16) and Receiving (17). Fig. 9 shows the bisimulation graph by linking the related states of both the models on a state transition graph. Observations show that bisimulation graph represents the one to one transition of different state as per above mentioned equations.

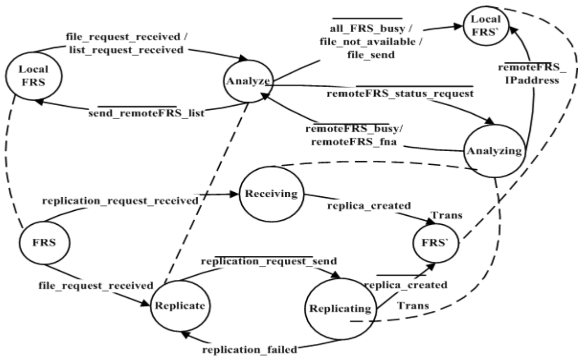

Figure 9

Bisimulation graph by linking the related states on a transition graph

Requesting node (N_{fr}) sends a file request to the local FRS (S_i), refer equ. (10) and receives the status from S_i followed either by the requested file or the IP address of the remote FRS having the copy of the requested file, refer equ. (11). In this case, the request is fulfilled by the available copy of the requested file either by the local FRS itself (8) or by remote FRS having a copy of that file (9). Remote FRS discards the file request in case the status of remote FRS against the requested file is N_{busy} or N_{fna}, refer equ. (9). If the status of any FRS is marked as *ready*, i.e., N_{ready}, the request gets redirected and fulfilled by this FRS, equ. (8).

State 'Replicate' represents that, the file replication is required as the status of the FRSs that have the replica of the requested file is busy N_{busy}. Intermediate state is represented by *Replicating* and FRS' represents the final state after the replication has been done, i.e., file has been transferred completely, and the connection has been closed. State FRS represents that the connection has been established between two nodes. State *Receiving* represents the intermediate state. The output would be sent from the transmission line (*Trans*).

4 Simulation and Results

The proposed model is simulated on JAVA platform. Threshold is fixed in accordance to the constraints and demands, depending on the application requirement. Experimental system configuration is illustrated in Table 4. Table 3 shows the request completion time in seconds for replicating the file of size 64.1 MB. Table 3 shows the worst case scenario, in which 100 requesting nodes send request for the file f simultaneously.

The first few requests handled by FRS takes more time because this time is inclusive of replication overhead from FRS1 to FRS2, and FRS1 to FRS3, but subsequently the service time taken by FRSs drops from 281.62 seconds to 245.62 seconds. The average request completion time decreases by 12.78%.

Table 3
Average request completion time (seconds)

Number of FRSs/ Number of Requesting Nodes →	1-20	21-40	41-60	61-80	81-100	Average
2 FRS	400.75	265.86	220.67	300.15	220.67	281.62
3 FRS	292.21	251.55	148.51	252.35	283.51	245.62

In replication scenario, all available FRS's are utilized to fulfill the request. As shown in Table 3, service time for requesting node 41-60 in case of 3 FRS is very less, this is because, by the time FRS receives this request, some of the previous requests gets completed, same is shown in Fig. 10. When the local FRS reaches the file threshold value and replicates the file on some other FRS, the replication overhead is compensated by the following benefits: (1) Avoid retransmission of request by the requesting node. (2) Reduces latency in case of load above threshold.

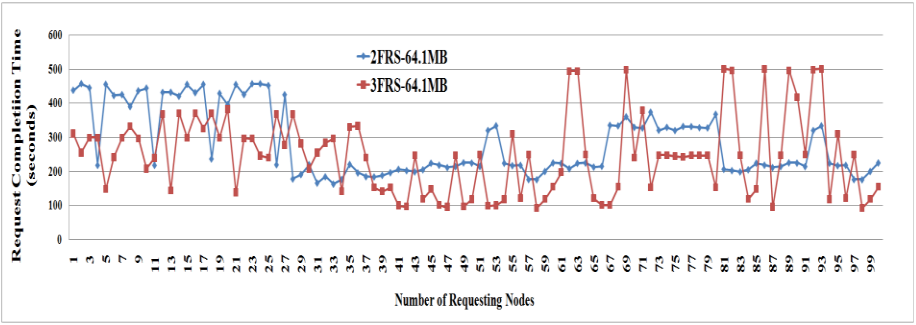

Figure 10
Request Completion Time for 64.1MB file

Fig. 11 shows the time taken by the local FRS, to invalidate the replica available on the Requesting Nodes (RNs) that are connected to this local FRS. Size of invalidate message is 15 bytes. The invalidate message to RNs is sent by the local FRS. Two cases are shown in the figure viz., Best case and Worst case. In the best case, replica does not exist on all the RNs. It is considered that out of 30RNs connected to the local FRS, 70% of the RNs have the replica of the modified file. In the worst case, replica of the modified file is available on all the RNs.

The average time required for invalidating the replicas using hybrid consistency approach in the best case is 13.43 msec and in the worst case is 20.36 msec. For some RNs, it is observed that there is a delay in the message delivery, due to which there is the formation of the crest as shown in the figure.

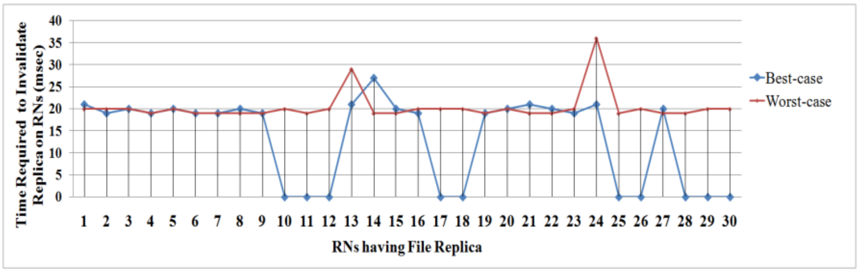

Figure 11

Time required for invalidating stale replica on requesting node

Fig. 12 shows the time required to update the replica on the FRS for file size of 128 kb, 677 kb, and 3.1 mb. Two cases that are shown in the figure are Complete File Transfer (CFT) and Hybrid approach. In CFT, complete file that has been modified is sent to the FRS having the replica of that file. In case of hybrid approach, only the modification that has been done is sent to the FRS having the replica of that file. Time required for updating replicas using hybrid consistency approach reduces from 153.33 msec to 58.64 msec.

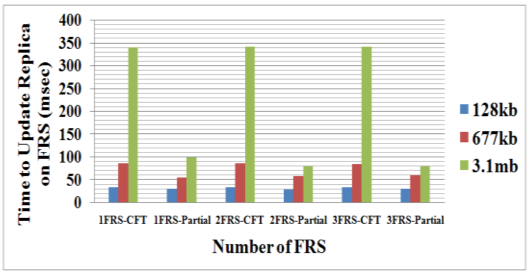

Figure 12

Time required for updating stale replica on FRSs

4.1 How Proposed Approach Weighs against Existing Approaches

The configuration used by the Google File System (GFS), and the proposed scheme in shown in the Table 4.

Table 4

Experiment configuration table

	Processsor	Memory	Hard Disk	Ethernet Connection	Switch
GFS	Dual 1.4 GHz PIII	2 GB	Two 80 GB 5400 rpm	100 Mb/s full-duplex	1 Gb/s link
Proposed	3.6 GHz P IV	1 GB	80 GB 5400 rpm	-	100Mb/s

In case, N clients write simultaneously to N distinct files. Fig. 13 shows that with GFS the average write rate reaches 21.8 MB/s for 10 clients and with the proposed Partial Write scheme the average write rate is 35.52 MB/s. The proposed partial write rate shows the improvement of 38.62% as compared to GFS write rate.

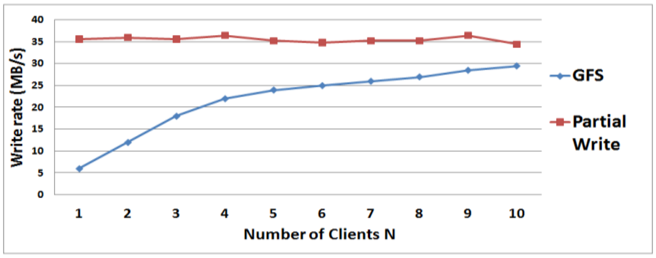

Figure 13

Write rate as the number of replicas to be updated increases (MB/s)

Fig. 14 shows that when N clients read simultaneously from a file, GFS average read rate reaches 44.5 MB/s for 10 clients and with the proposed scheme the average read rate is 14.57 MB/s. GFS read rate is 67.23% better than the proposed scheme. This is due to the system configuration as described in Table 4.

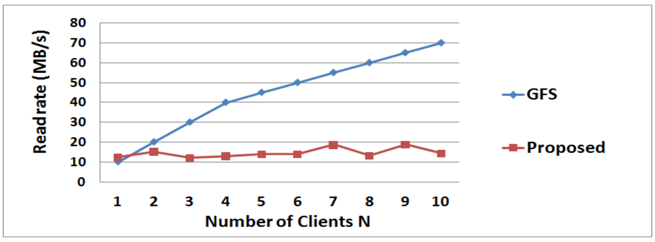

Figure 14

Read rate with an increasing number of readers (MB/s)

Conclusion

This paper makes an attempt to propose and establish the threshold based file replication approach in the distributed cluster computing environment, having a mini data centric view. Proposed file replication and consistency maintenance mechanism, autonomously determines the need for file replication based on the file threshold and availability of files on the nodes of cluster environment. The decentralized architecture for the proposed model eliminates the possibility of single point failure. Proposed approach ensures automatic retransmission of request to the remote FRS, in case the local FRS fails. The hybrid consistency mechanism that reduces the time for updating multiple replicas of a file by using modification propagation is also proposed. Experimental results show that as compared to Google File System (GFS), the proposed partial write rate shows an improvement of 38.62% while updating stale replicas. Time required for invalidating the replicas decreases by 34.04% and the time required to update the replica on FRS reduces by 61.75%. The replication overhead is compensated by the benefits like avoiding retransmission of request by the requesting node, and reducing file access latency.

Finally, a relationship between the formal aspect of file replication server and its architectural model, i.e., proposed file replication model is established through process algebraic approach. The stability and reliability analysis ensure that the system will run in the finite sequences of interaction and transitions. On the basis of these properties, a transparent, reliable and safe file replication model is built.

References

[1] R. Buyya, High Performance Cluster Computing, Vol. 1, Pearson Education, 2008

[2] J. Grudin, "Computer-Supported Cooperative Work: History and Focus". Computer 27 (5), 1994, pp. 19-26

[3] R. A. Milner, Calculus for Communicating System, in: Lecture notes in Computer Science, Vol. 272, Springer, 1980

[4] Rajkumar Buyya and Rajiv Ranjan, Federated Resource Management in Grid and Cloud Computing Systems, Future Generation Computer Systems, Volume 26, No. 8, ISSN: 0167-739X, Elsevier Press, Amsterdam, The Netherlands, Oct. 2010

[5] Ivona Brandic and Rajkumar Buyya, Recent Advances in Utility and Cloud Computing, Future Generation Computer Systems, Volume 28, No. 1, ISSN: 0167-739X, Elsevier Press, Amsterdam, The Netherlands, Jan. 2012

[6] Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, In Sixth Symposium on Operating System Design and Implementation (OSDI'04), San Francisco, CA, Dec. 2004

[7] S. Ghemawat, H. Gobioff and S. T. Leung, "The Google file system," SIGOPS Oper. Syst. Rev., Vol. 37, pp. 29-43, 2003

[8] Thomas Anderson, Michael Dahlin, Jeanna Neefe, David Patterson, Drew Roselli, and Randolph Wang. Serverless networkfil e systems. In Proceedings of the 15th ACM Symposium on Operating System Principles, pages 109-126, Copper Mountain Resort, Colorado, Dec. 1995

[9] John Howard, Michael Kazar, Sherri Menees, David Nichols, Mahadev Satyanarayanan, Robert Sidebotham, and Michael West. Scale and performance in a distributed file system. ACM Transactions on Computer Systems, 6(1):51-81, February 1988

[10] InterMezzo. http://www.inter-mezzo.org, 2003, accessed on 20 July 2012

[11] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani: A scalable distributed file system. In Proceedings of the 16th ACM Symposium on Operating System Principles, pp. 224-237, Saint-Malo,

France, October 1997

[12] Lustre. http://www.lustreorg, 2003, accessed on 20 July 2012

[13] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in Proceedings of OSDI 2006, Seattle, WA, 2004

[14] M. Burrows, "The Chubby Lock Service for Loosely-coupled Distributed Systems," in Proceedings of OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November 2006

[15] Jun Rao, Eugene J. Shekita, and Sandeep Tata. 2011. Using Paxos to Build a Scalable, Consistent, and Highly Available Datastore. Proc. VLDB Endow. 4, 4 (January 2011), pp. 243-254

[16] Lamport, L. The part-time parliament. ACM TOCS 16, 2 (1998), 133-169

[17] Andrea Domenici, Flavia Donno, Gianni Pucciani, Heinz Stockinger, Kurt Stockinger, "Replica Consistency in a Data Grid," Nuclear Instruments and methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Volume 534, Issues 1-2, pp. 24-28, 21 November 2004

[18] L. Guy, P. Kunszt, E. Laure, H. Stockinger and K. Stockinger "Replica Management in Data Grids", Technical report, GGF5 Working Draft, Edinburgh, Scotland, July 2002

[19] Yuzhong Sun and Zhiwei Xu, "Grid Replication Coherence Protocol", The 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop, Santa Fe, USA, pp.232-239. April 2004

[20] A. Y. Zomaya (Ed.), Parallel and Distributed Handbook, McGraw Hill Professionals, pp. 60-68, 1995

[21] R. Milner, Communication and Concurrency, Prentice Hall 1989

[22] S. Mishra, D. S. Kushwaha, A. K. Misra, "Hybrid Reliable Load Balancing with MOSIX as Middleware and its Formal Verification using Process Algebra," Future Generation Computer Systems, Volume 27, Issue 5, pp. 506-526, May 2011