A Demand Based Load Balanced Service Replication Model

Manu Vardhan
CSED, MNNIT Allahabad
Motilal Nehru National Institute of Technology
Allahabad, India
rcs1002@mnnit.ac.in

Shrabani Mallick
CSED, MNNIT Allahabad
Motilal Nehru National Institute of Technology
Allahabad, India
shrabani@mnnit.ac.in

Shakti Mishra
Asstt. Professor
Institute of Development & Research in Banking Technology
Hyderabad, India
smishra@idrbt.ac.in

D. S. Kushwaha
CSED, MNNIT Allahabad
Motilal Nehru National Institute of Technology
Allahabad, India
dsk@mnnit.ac.in

Abstract—**Cloud computing allows service users and providers to access the applications, logical resources and files on any computer with ease. A cloud service has three distinct characteristics that differentiate it from traditional hosting. It is sold on demand, typically by the minute or the hour; it is elastic. It is a way to increase capacity or add capabilities on the fly without investing in new infrastructure, training new personnel, or licensing new software. It not only promises reliable services delivered through next-generation data centers that are built on compute and storage virtualization technologies but also addresses the key issues such as scalability, reliability, fault tolerance and file load balancing. The one way to achieve this is through service replication across different machines coupled with load balancing. Though replication potentially improves fault tolerance, it leads to the problem of ensuring consistency of replicas when certain service is updated or modified. However, fewer replicas also decrease concurrency and the level of service availability. A balanced synchronization between replication mechanism and consistency not only ensures highly reliable and fault tolerant system but also improves system performance significantly. This paper presents a load balancing based service replication model that creates a replica on other servers on the basis of number of service accesses. The simulation results indicate that the proposed model reduces the number of messages exchanged for service replication by 25-55% thus improving the overall system performance significantly. Also in case of CPU load based file replication, it is observed that file access time reduces by 5.56%-7.65%.**

Keywords-component; Service Replication, Consistency, File Replication Server, Load balancing, Request-Reply Protocol

I. INTRODUCTION

Cloud computing systems fundamentally provide access to large amounts of data and computational resources through a variety of interfaces. These resources are provided on demand basis through various file servers available across cloud service providers. Further, in addition to reliability and scalability, a fault tolerant mechanism ensures functioning of system even in the case of failure. The one way to achieve this is through service replication across different machines. Setting up a replication configuration is a fairly standard way to enable disaster recovery (DR) to recover from critical failure. Replication means replicating the critical software components or software services on to other machines, so that if one of these fail, the others can be used to continue.

Though replication potentially improves fault tolerance, inconsistency may lead to the condition where clients can have stale files. As a result, each time a service is modified, consistency mechanisms ensure that all existing copies are consistent. In distributed file system this happens usually either by transmitting the modified service or invalidation message to the various sites, or by transmitting only the modified section service. However, fewer replicas also decrease concurrency and the level of service availability. While developing robust systems, maximizing service replication enables minimized costs of consistency related message transmission. Clearly, there is a strong relation between the service consistency mechanism, message transfer, and file availability. As a result, system can handle large number of requests as several copies of the service exist. Demand based Service Replication Model proposed in this paper avoids unnecessary service replication and tries to resolve the following issues:

1. If a copy of the requested file is available on a peer node, preventing unnecessary replication,
2. Involuntary routing the file request in case of node failure, without any user intervention.
3. Dynamic load distribution among peer servers.

The proposed mechanism uses asynchronous communication that also ensures that the system will keep accepting the requests without blocking its state.

The rest of the paper is organized as follows. The next section discusses a brief literature survey of existing theories and work done so far. Section 3 proposes a Demand based Service Replication Model. Section 4 carries out the simulation and results. Finally, section 5 concludes the work followed by references.

II. RELATED WORK

Replication means high availability of resources. Resources can be physical or logical. Physical resources include memory and storage capacity, whereas logical resources include file, data and services that need to be replicated or made available on demand, depending upon the application requirement. Resource replications are basically of two types, active and passive. The Passive replication is like primary copy and all updates are redirected to the primary copy. The updates can be propagated after the transaction has been committed. In active replication, mutual information about the peer nodes is maintained and the replicated resources can be accessed at any site. The traditional resource replication is passive, that does not participate in the decision on when to replicate, where to replicate and number of copies to replicate. In a blind-replica service model [13], request routing is independent of where the replicas are located. Each replica simply serves the requests

DOI: 10.5176/2251-3043_2.4.202

flowing through it under a given routing strategy. Various replication strategies have been proposed on the basis of the relative popularity of individual files based on their query rate. [6] proposed a query based file popularity approach for replication. Common techniques include the square-root, proportional, and uniform distributions. The approaches in [10] and [4], consider static replication in combination with a variant of Gnutella searching. Static strategies are applied for replication when there is little gain from using dynamic strategies if the resource conditions are fairly stable over a long period of time. Dynamic strategies are able to recover from failures such as network partitioning and easily adapt to changes on demand, bandwidth and storage availability.

Clarke, et al. [3], replicates objects both on insertion and retrieval on the path from the initiator to the target mainly for anonymity and availability purposes. The methodology in [14] addresses data replication and considers that adaptive replication algorithms change the replication scheme of an object to reflect the read-write patterns and eventually converge towards the optimal scheme. The adaptive data replication algorithm aims at decreasing the bandwidth utilization and latency by moving data closer to clients. A dynamic replication strategy is proposed in [5] by the name of push caching. A server knows how popular its files are and so it decides when to 'push' one of its popular files to a remote 'friend' server. It decides where to replicate them by using the access history that it has stored. Our strategy of maintaining detailed access histories to determine the popular files is modeled on this study.

File clustering based replication algorithm in a grid environment is proposed by [8], which presents the location based replication mechanism. The files stored in a grid environment, are grouped together based on the relationship of simultaneous file accesses and based on the file access behavior the location and movement of replicas is determined. Similarly, locality aware file replication is proposed by [2], to ensure data reliability and availability through the parallel I/O system. The approach in [7] discussed granularity of replication, which means the unit of data that may be replicated independently of other units of data. For a replicated file system, unit of data include a file, a record within a file. To ensure synchronized file replication across two loosely connected file systems, a transparent service for synchronized replication across loosely connected file system is developed in [11] that propagate the modification of replicated files and directories from either file system.

The authors in [9] proposed file replication and migration policy, by which the total mean response time for a requested file at a particular site can be reduced. Similarly [1], propose an adaptive file replication policy, which is capable of reacting to changes, by dynamically creating or deleting replicas.

An approach has been proposed in [15] wherein the job arrivals are characterized by correlation among their dependencies. The author's have proposed a policy wherein the migrated processes are queued based on the size of the processes and even considers the autocorrelation among these. The proposed approach does not consider the priority of processes, which is the major limitation of the cited approach. Zhang and Pande [16] discuss about minimization of the transfer cost and define strategy as to which parts of the program should migrate. Many of the researchers [17] have tried to resolve issues like longer freeze time that may be due to unavailability of competing resources but their approach requires pre-fetching of memory pages for process transfer. A hybrid load balancing policy underlying grid computing environment [18] proposes a dispatcher and agent based approach. The dispatcher performs maintenance, status monitoring, node selection and assignment and adjustment task for each node. The author's consideration of load balancing restricts the system to the ''join and leave'' decision of nodes. This suits P2P system but not CSCW. Dynamic Load Balancing (DLB) [19] provides application level load balancing using system agents and DLB agent. The approach requires a copy of system agents on all the system so that DLB agent may collect load information from these systems and perform load balancing. The other contemporary work includes grid load balancing using Intelligent Agents [20] that proposes a combined approach using intelligent agents and multi-agents for effectively scheduling the local and global grid resources that also incorporate peer to peer advertisement and service discovery to balance the workload. The approach requires a copy of system agents on all system so that DLB agent may collect load information from these systems and perform load balancing. Yagoubi and Slimani [21] puts forward a dynamic tree based model to represent grid architecture and proposes Intra-site, Intracluster and Intra-grid load balancing. The authors in [22] describe a small cluster along with efforts to improve the efficiency of parallel scientific computation on that cluster.

III. THE PROPOSED DEMAND BASED SERVICE REPLICATION (DBSR) APPROACH

In the proposed Demand Based Service Replication (DBSR) model, client nodes requests for files and the Service File Replication Server replicates the service on other servers on the basis of the number of requests it receives. A group of such Service File Replicator Servers works in peer-to-peer manner to provide the most updated version of file to clients and to make the replication process smooth and non-complicated.

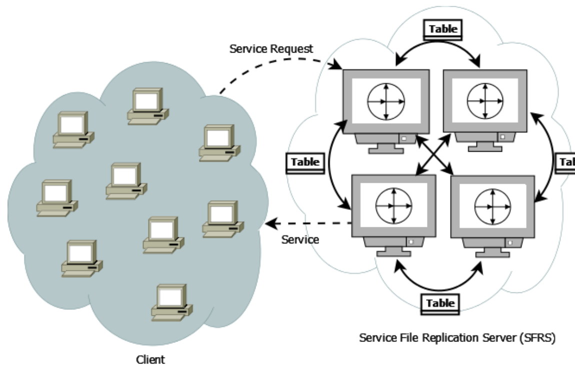

Figure 1: The Proposed Demand Based Service Replication (DBSR) Model

A. *Data Structures Used:*

To exchange the local information and to keep the databases of peer servers update, each FRS maintains a lookup table (cf. Table 1) and a local service table (cf. Table 1).

1. *Look-up Table*

The lookup table contains the following entries:
Server IP: The Server_IP field typically contains the IP address of peer servers.
Service_id: This field contains the Service_id of those replicated services which are currently available on the servers.
Last Update: The last update shows the last time when the service was modified. If a table contains a new version of this service, the server itself initiates the update process with the peer server.

Table 1: Lookup Table Structure

Server_IP	Service_id	Last Update

2. *Local Service Table*

Each FRS constructs a Local Service Table, and regularly updates it as a request for a cloud service reaches to it. It contains three fields:
Service_id: This contains the Service_id of those services which are available on the server.
Replica_Available: The number of replicas available for a particular service.
Last Update: The last update shows the last time when the service was modified.

Table 2: Local Service Table Structure

Service_id	Last Update	Replica_Available

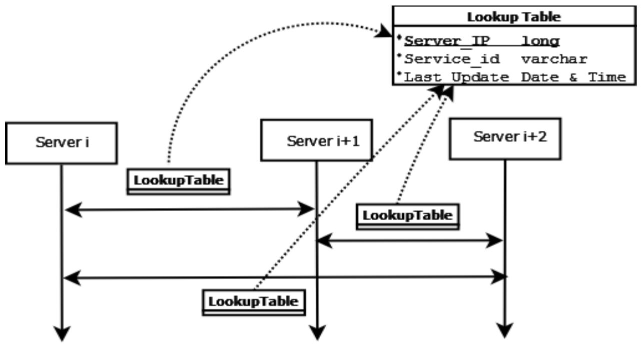

Figure 2: Exchange of Lookup Table among Peer Servers

B. *DBSR Replication Mechanism*

The DBSR Model works as follows:

The peer servers are connected with each other using mesh topology. The Servers authenticate each other by finding a certificate in common and testing to be sure that certificates are authentic.
1. Replication is initiated by a server or a workstation in one of the following ways:

- A client sends a request for a particular service to any of the Service File Replication Server.
- A connection is established between server and client.
- If the requested service is available with the server, it sends the replicated service to user.
- If the service is not available at server, it route the request to the peer server that is available with the requested file.

2. Each SFRS constructs a lookup Table and Local Service Table as discussed above.
3. Each server exchanges their lookup table with other servers as shown in Figure 2.
4. When a server receives a request but is not capable of fulfilling the request, it searches the lookup table and contacts the peer server. The peer server sends its local service list to requesting server. Upon receiving this table, if server finds that any of the replicas is available, then it transfers the request to this peer server. If no replica is available, then it immediately contacts another peer server.

C. *How Replication Works?*

In order to keep the unused replicas into bin and migrating or replicating the services on to peer servers whenever a burst of requests come, following parameters are used (cf. Table 3).:

Table 3: Parameters Proposed for Replication Algorithm

Parameters	Explanation
Req_Count (r1)	The current number of requests for replica of file f1
Min(r1)	Minimum number of replicas of Service S1 that can reside on server Si
Max(r1)	Maximum number of replicas of Service S1 that can reside on server Si
Avail_Rep(r1)	Available number of replicas of Service S1 that are currently residing on server Si
loadbound	Total load on the server Si computed as Total number of requests for all services on Si
loadmax	Maximum load on the server Si that can be handled

Case 1: For sharing the load of servers

The pseudo code for finding out when to replicate/migrate the service on the peer server is as follow:
If number of incoming requests on server for R1exceeds (Avail_Rep (r1) - Min (r1)), then an alarm is raised by server Si to handle sudden outburst of such requests in near future. To cope up with this kind of situation, server Si asks its peer server (Si+1) to share its local file table. If the number of available replicas for R1 i.e. Avail_Rep(r1) lies somewhere in between maximum(Max(r1)) and minimum (Min(r1)) number, server Si knows where to route those requests. However, if this condition fails then before migrating replica, it checks for one more condition. In this case, the total load (in terms of number of requests that are currently being processed) of the system is measured against maximum number of request a server can handle. If the peer server is found below loaded, the

replica migration takes place from Si to Si+1. If this condition does not satisfy, the request is sent to next peer server (S_i+2). Lets, explains the same with an example where request for service s1_4 comes to Server Si. It is assumed in all the cases that all the machines are within load limit, means no server is running on maximum load condition. The Avail_rep for service s1_4 is only 2, the Req_Count =1, and Avail_Rep – Min (i.e. 2 -1 =1), hence request to share local file table is sent to peer server Si+1. Although the first client simply allows downloading the service but at that moment, the Avail_rep becomes 1 and the corresponding update is propagated to Local service table. Now if one more request comes the Req_Count reaches to 2 that is greater than Avail_Rep – Min (i.e. 1 -1 =0), hence immediately one available replica gets transferred to new client, but simultaneously one of the service replica gets migrated to new peer server. Since Si+1currently holds one replica, any new request is entertained with peer server and in the meanwhile time, more replicas can be created on peer server or on the originating server.

Case 2: To update the Service File version

Whenever a look up table is exchanged among peer servers, all servers check that if a new version of the service is available. If any of the servers accommodates a new version of any service, then all other servers pull the updated replica from server and update the file version. This has to be noted down that like traditional databases only changes are pulled. Here, an updating procedure using an example where a service (s1_3) has three versions on three different servers. However, during periodic update when server gets to know that a latest version is available on peer server, the request to pull these changes is sent to peer server. The servers correspondingly acknowledge the request and send a latest version of service file to requestor server.

D. CPU Load based File Replication Mechanism

We consider a network of four Replicating Server (RS) which are connected to each other via intercommunication network. Each RS is assumed as the trusted node. In the proposed File Replication Model as shown in Fig. 3, an underloaded RS can fulfill the file request of the client whereas an overloaded RS looks for an underloaded RS on which the file request can be redirected. Average loaded RS are the ones which neither redirect the file request nor serve the new file request, so as to avoid getting overloaded. In order to reduce the overhead of polling and broadcasting periodically, RS does not enquire about the load status of all RS. Instead each RS sends its load status information to other RS when it changes. RS only monitors other RS status by its state table. Hence, RS never takes the responsibility of various RS status. The Hybrid load balancing algorithm works as follows:

```
If (CPU_Status(RS) changes then for $RS_i$
        {do (for all $RS_{i+1}$; i=1, 2, 3…, n)
                Send $RS_i$ status to all peer $RS_{i+1}$ to update
        their node status table}
If (CPU_Status (RS) (overloaded) ||
(file_request_count>=fileThreshold)
{Look for underloaded RS in node status table and Replicate
the requested file on an underloaded RS and redirect the
request to this underloaded RS}
```

The hybrid load balancing based file replication mechanism is illustrated in Fig. 3.

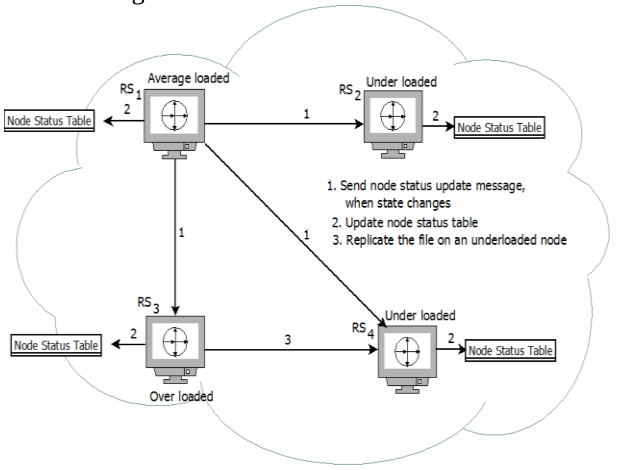

Figure 3. File replication based on CPU load balancing

File replication strategy: Replicating Server (RS) will replicate the file as follow:
1. The RS will check the CPU load, if it is overloaded, RS will replicate the file on an underloaded node.
2. In case RS CPU load is average, but the file threshold has been reached, RS will replicate the file on an underloaded node.
Inter-cluster/Destination node selection: Proposed Selection Strategy for choosing an idle workstation is based on the following aspects: *Least Busy RS First (LBRSF)*: On the basis of the information provided by RS, overloaded RS may choose least busy RS for file replication. This policy may restrict replicating the file because of unavailability of underloaded RS. *Random Selection*: An overloaded RS will replicate the file randomly to any idle RS. The selection criteria may be the availability of resources, memory availability, network bandwidth, compatibility among system and many other factors.

E. State Transition Diagram for Service Replication Service

The transition systems are considered to perform external and internal actions. The State Transition Diagram for DBSR Model has been shown in the Figure 4. The states are Start, Connect, Receive, Analyze, Replicate, Send and the virtual state Time Out. The meanings of states are as shown in Table 4.

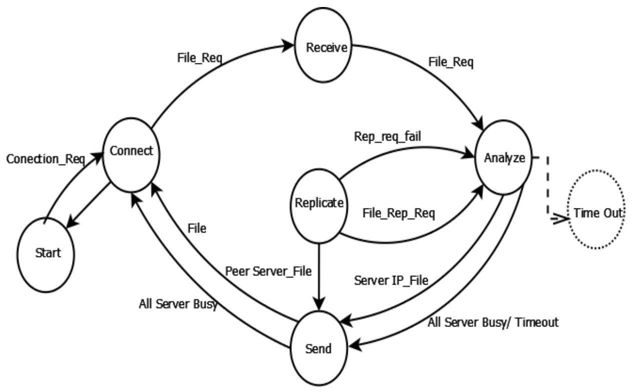

Figure 4: State Transition Diagram for DBSR Model

Table 4: States in Transition Diagram and their meanings

State	Meaning
Start	Represents the initial state from where client attempts to establish a connection with Server
Connect	When a successful connection gets establishes between server and client.
Receive	All incoming requests come to this state
Analyze	It analyzes the incoming requests, responsible for replica migration and dispatches the result to send port. It also takes care of node crash and timeout, upon receiving which it sends appropriate message to send.
Replicate	This state or extracts the replicates the service on server as requested by Analyze state and simultaneously also checks for node crash or failure.
Send	All outgoing results and messages are sent through Send.
Time out	A virtual state that keeps track of session failure or time out and communicates with Analyze.

State Transition Definition: Let A be the finite alphabet of observable actions whose element are denoted by a, b, c. let τ be the symbol of unobservable or internal action, not belonging to A. Then Act can be defined as A U {τ} and the elements u, v…   A U {τ} and w is the element of (A U {τ})*.

Transition system is quadruple A = (A, S, , S_0), where,

(a) A is an alphabet.

(b) S possibly infinite set of states; S {Start, Connect, Receive, Analyze, Replicate, Timeout, Send}

(c) $\rightarrow \epsilon SX(A \cup \{\tau\})XS$

(d) S_0= {Start}is initial state.

Let state S_1, S_2 range over S, then (S_1, u, S_2) will also be denoted by $S_1 \overset{u}{\rightarrow} S_2$, $\rightarrow^*$ denotes transitive closure of $\rightarrow$. Let w = u1, u2… u_n (A € {τ})*, then $S \xrightarrow{w} S'$ if S_1, S_2, S_3 …, S_n, S_{n+1} € S such that,

$$S = S_1 \overset{u_1}{\rightarrow} S_2 \overset{u_2}{\rightarrow} S_3 \overset{u_3}{\rightarrow} \ldots \ldots S_n \overset{u_n}{\rightarrow} S_{n+1} = S'$$

State Transition Equations:

The start state initiates the communication by sending the connection request to connect.

$$Start \xrightarrow{Connection\ \_Req} Connect \quad (1)$$

$$Connect \xrightarrow{Service\ \_Req} Receive \quad (2)$$

Once the connection gets established, the client request for a particular service reaches to receive via connect. Upon receiving this request, the request is put to analyze. The state gives the list of server where this file is currently available. If the service file is available with server, the analyze sends the IP address of server to send from where file gets transferred to client. If service file is not available then, a service replication request is sent to Replicate, and a new replica is created/migrated on peer server. The peer server ip from replicate is sent to send. While replication, if one of the node crashes or replication fails due to timeout, all server busy message is sent to the client.

$$Receive \xrightarrow{Service\ \_Req} Analyze \quad (3)$$

$$\text{Analyze} \xrightarrow{Service\ \_Rep\ \_Req} Replicate \quad (4)$$

$$Replicate \xrightarrow{Rep\ \_Req\ \_fail} Analyze \quad (5)$$

$$\text{Analyze} \xrightarrow{timeout\ ,ServerBusy} Send \quad (6)$$

$$Replicate \xrightarrow{PeerServer\ \_service} Send \xrightarrow{service} Connect \quad (7)$$

$$Analyze \xrightarrow{Server\ \_service} Send \xrightarrow{service\ \ File} Connect \quad (8)$$

$$Send \xrightarrow{AllServerBusy} Connect \xrightarrow{AllServerBusy} Start \quad (9)$$

IV. Simulation and Results

The proposed model is simulated on JAVA platform. DBSR approach is compared with Request Reply Acknowledgement (RRA) [12] and Request Reply (RR) protocol [12]. It outperforms RRA and RR protocol in terms of replication. Given below are the details and possible cases for better understanding of DBSR approach.

Table 5: Number of messages exchanged per request

Service File Replication		Total number of Messages		
		DBSR	RR	RRA
Case 1	Replica available on peer server	2n+4	2n+6	3n+9
Case 2	New replica created	2n+6	2n+10	3n+15

Table 5 shows the comparison of the DBSR approach in terms of messages exchanged per request with the existing RR and RRA protocol.

Case 1: Replica available on peer server and Case 2: New replica created, for DBSR, RR and RRA protocol. In RR and RRA protocol, there is no routine mechanism for getting the IP address of the peer server on which the file is replicated, as compared to DBSR approach. In case of RR and RRA protocol, new FRS and ip_Req message, will provide the IP address of peer node having the copy of replicated resource.

Table 6 shows Case1 where the communication is established between only those peer server on which the file is present. Here 0≤n≤i. It shows the data for n=1.

Case 1 Is treated as the special case of replication scenario, if the replica for the requested file is available on a peer node. So the number of messages exchanged will get reduced to eight and twelve respectively for RR and RRA protocol.

Case 2 In this case the communication is established between only those peer server on which the file is not present. Here 1 ≤n≤i. It shows the data for n=1.

Based on the total number of messages exchanged for successfully completing the request for file list, transfer and replication request, it is established that DBSR approach runs well for file list and transfer request. It outperforms the other two protocols, when used for file replication. In terms of total number of messages exchanged, DBSR approach shows significant performance improvement, for file replication request, as compared to file list and transfer request.

V. CONCLUSION

Cloud computing has been used as an extension of parallel processing. Coordinating various computing resources to achieve bigger task is the key of cloud computing. In this paper, we propose a demand and load balancing based file replication model that makes an attempt to create a replica only when the number of requests exceeds than a pre-defined threshold. The threshold can be taken on the basis of server configuration. Of the emerging technologies cloud computing has a lot of substance. The huge set of challenges it has brought with it has to be captured and tamed to produce more benefits. The proposed approach is able to resolve many of the unaddressed issues viz., selection criteria of node for replica placement, failure handling, file popularity based replication and avoidance of unnecessary file replication. Instead of haphazardly creating the replica, Demand Based Service replication approach (DBSR), autonomously determine the need for service file replication based on the number of requests, maximum number of replicas a server may possess and availability of files on the peer nodes. The proposed replication approach ensures accurate decision making while locating the resources and fetches them to fulfill the request in a transparent manner. While performing some service replication operation, if the node crashes, the DBSR model makes an attempt to complete the file request via peer node thus providing fault tolerance capability to the system. DBSR approach provides service replication, access and performance transparency to the system, thereby ensuring the replication decisions about the services. Results indicate that, threshold based services replication approach reduces the number of messages exchanged for service replication by 25-55%. Also in case of CPU load based file replication, it is observed that file access time reduces by 5.56%-7.65%. It establishes that DBSR approach runs well for file list and transfer request. It outperforms the other two protocols, when used for file replication. In terms of total number of messages exchanged, DBSR approach shows significant performance improvement, for service file replication request, as compared to file list and transfer request in comparison to request reply and request reply acknowledgement protocol, thus minimizing the network resource utilization. We believe it will shift the operational paradigm of the collaborative business process.

REFERENCES

[1] G. Cabri, A. Corradi, F. Zambonelli, "Experience of Adaptive Replication in Distributed File Systems", IEEE Proc. of 22nd EUROMICRO Conf. on Beyond 2000, Hardware and Software Design Strategies, 1996, pp. 459-466.

[2] H.Y Cheng, C.T. King, " File Replication for Enhancing the Availability of Parallel I/O Systems on Clusters", 1st IEEE Computer Society Int. Workshop on Cluster Computing, 1999, pp. 137-144.

[3] I. Clarke, O. Sandberg, B. Wiley, T. Hong, "Freenet: A distributed anonymous information storage and retrieval system",2000.

[4] E. Cohen, S. Shenker, "Replication strategies in unstructured peer-to-peer networks", In The ACM SIGCOMM'02 Conference, August 2002.

[5] J. Gwertzman, M. Seltzer, "The case for geographical push-caching", 5th Annual Workshop on Hot Operating Systems, 1995.

[6] S. Helen, IRM: Integrated file replication and consistency maintainence in P2P Systems, IEEE Trans. on Parallel and Distributed Systems, Vol. 21, No. 1, January 2010, pp. 100-113.

[7] A. Hisgen, et al. "Granularity and semantic level of replication in the Echo distributed file system", Workshop on the Management of Replicated Data, 8-9 Nov 1990, pp.2-4.

[8] S. Hitoshi, S. Matsuoka, T. Endo, "File Clustering Based Replication Algorithm in a Grid Environment", 9th IEEE/ACM Int. Sym. on Cluster Computing and the Grid, 2009, pp. 204-211.

[9] R.T. Hurley, S.A. Yeap, " File migration and file replication: a symbiotic relationship", IEEE Trans. on Parallel and Distributed Systems, Vol. 7, No. 6, June 1996, pp. 578-586.

[10] Q Lv et al., "Search and replication in unstructured peer-to-peer networks", In Proceedings of the 16th ACM International Conference on Supercomputing, New York, USA, June 2002.

[11] H. Rao, A. Skarra, "A transparent service for synchronized replication across loosely-connected file systems", 2nd International Workshop on Services in Distributed and Networked Environments, 5-6 Jun 1995, pp.110-117.

[12] A. Z. Spector, " Performing remote operation efficiently on a local computer Network", Communications of the ACM, Vol. 25, No. 4, 1982, pp. 246-259.

[13] X. Tang, C. Huicheng, S.T. Chanson, "Optimal Replica Placement under TTL-Based Consistency", Parallel and Distributed Systems, IEEE Transactions on , vol.18, no.3, March 2007, pp.351-363.

[14] O. Wolfson, S. Jajodia, Y. Huang,,"An adaptive data replication algorithm", ACM Transactions on Database Systems, 22(2):255–314.

[15] Q. Zhang, N. Mi, A. Riska, E. Smirni, Load unbalancing to improve performance under autocorrelated traffic, in: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, 2006.

[16] K. Zhang, S. Pande, Efficient application migration under compiler guidance, in: Proceedings of the 2005 ACM SIGPLAN Conferences on Languages, Compilers, and Tools for Embedded Systems, 2005.

[17] R.S.C. Ho, Cho-Li Wang, F.C. Lau, Lightweight process migration and memory prefetching in openMosix, in: IEEE International Symposium on Parallel and Distributed Processing IPDPS, 2008.

[18] K.Q. Yan, et al., A hybrid load balancing policy underlying grid computing environment, Journal of Computer Standards & Interfaces (2007) 161–173.

[19] R.U. Payli, et al., DLB—a dynamic load balancing tool for grid computing, Scientific International Journal for Parallel and Distributed Computing 07 (02) (2004).

[20] Junwei Cao, et al., Grid load balancing using intelligent agents, Future Generation Computer Systems 21 (1) (2005) 135–149.

[21] Yagoubi, Y. Slimani, Task load balancing for grid computing, Journal of Computer Science 3 (3) (2007) 186–194.

[22] A. Silberchatz, P.B. Galvin, G. Gagne, Operating System Concepts, 8th ed., Wiley Sons, 2008.