2012 Third International Conference on Computer and Communication Technology

Demand Based File Replication and Consistency Mechanism

Manu Vardhan, Akhil Goel, Abhinav Verma, Dharmender Singh Kushwaha
Computer Science and Engineering Department
MNNIT Allahabad
Allahabad, INDIA
{rcs1002, cs084084, cs084076, dsk}@mnnit.ac.in

***Abstract*—File replication and consistency are well known techniques in distributed systems to address the key issues such as scalability, reliability and fault tolerance. For many years, file replication and consistency in distributed environment has been researched to enhance and optimize the availability and reliability of the entire system. An effort has been made in the present work to propose a file popularity based, adaptive, on demand, reliable and comprehensive file replication and consistency mechanism. This paper introduces a group of File Replication Servers (FRS) that is responsible for ensuring the file availability, in order to fulfill the file request. The present work proposes Hybrid file update propagation mechanism so as to minimize the number of updates. Proposed model replicates the file, from one node to the other node, when the total number of request for a particular file reaches a threshold value. Proposed approach is implemented on JAVA platform. Based on the file popularity, model creates and updates the file replica on the server. The proposed Hybrid approach is compared with write invalidate and write update propagation policies. Comparison results show that for balanced load hybrid file update propagation model updates the optimum number of replicas. Whereas, for unbalanced load, hybrid approach outperforms other two approaches by handling all the file requests with minimum number of updates.**

Keywords- Consistency; File Replication; Update Propagation;

I. INTRODUCTION

In large-scale distributed environments, there is huge requirement of providing reliability. One such method to achieve reliability is replication of the critical resources. The method of replication enhances the availability of the critical resources in the system, and also ensures fault tolerance. However, the biggest concern in replication mechanism is the issue of maintaining consistency among the replicated components of the same resource. There have been many attempts in the past to address this issue. Various update propagation policies have been used to communicate the changes made in the resource to its replicas. Two of the major policies of update propagation are write-update and write-invalidate. Both these policies have their own advantages and disadvantages.

This paper proposes a consistency mechanism that uses an update propagation policy that combines the advantages of both the above the mentioned update propagation policies. The proposed approach has been named the Hybrid Consistency Mechanism. This mechanism is capable of deciding whether to use write-update or write-invalidate as the update propagation policy depending on the dynamics of the whole system. As a result, system can handle large number of requests as several copies of the file exist. Model proposed in this paper avoids unnecessary file replication and tries to resolve the following issues: Prevents the creation of file, if a copy of the requested file is available on a peer node, file popularity-based, dynamic file replication on the peer servers, handling the file request in case of node failure without user intervention.

Model uses asynchronous communication ensuring that the system will keep accepting the requests without blocking its state. It provides fault tolerance to the system by automatically connecting the user to other FRS in case one FRS fails. The Hybrid Consistency Mechanism tries to resolve the number of replicas being updated depending on the popularity of the replicated file.

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 discuss about the proposed approach. Section 4 shows the simulations and their results. Finally, section 5 concludes the work followed by references.

II. RELATED WORK

Cohen and Shenker [1] consider static replication in combination with a variant of Gnutella searching. Static strategies are applied for replication when there is a little gain from using dynamic strategies. Dynamic strategies are able to recover from failures such as network partitioning and easily adapt to changes in demand, bandwidth and storage availability. Clark et al. [2] replicates objects both on insertion and retrieval on the path from the initiator to the target, mainly for anonymity and availability purposes. Wolfson [3] 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. Gwertzman and Seltzer [4] proposed a dynamic replication strategy 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. File clustering-based replication algorithm in a grid environment is proposed by Hitoshi et al. [5] 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 on the file access behavior. Similarly, locality aware file replication is

978-0-7695-4872-2/12 $26.00 © 2012 IEEE
DOI 10.1109/ICCCT.2012.74

proposed by Cheng and King [6] to ensure data reliability and availability through the parallel I/O system. Hisgen et al. [7] discussed granularity of replication which means that a unit of data that may be replicated independently of other units of data. To ensure synchronized file replication across two loosely connected file systems, a transparent service model has been developed by Rao and Skarra [8] that propagate the modification of replicated files and directories from either file system. Hurley and Yeap [9] proposed a file replication and migration policy by which the total mean response time for a requested file at a particular site can be reduced. Similarly, Cabri et al. [10] proposed an adaptive file replication policy which is capable of reacting to changes by dynamically creating or deleting replicas.

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 of the primary copy is large. Domenici [11] discusses several replication and data consistency solutions, including Eager (Synchronous) replication and Lazy (Asynchronous) replication, Single-Master and Multi-Master Model, and pull-based and push-based. Author presented various replication and consistency maintenance algorithms to deal with huge scientific data. Jaechun [12] proposed two kinds of data replication techniques, called owner-initiated replication and client-initiated replication. Proposed replication techniques do not need to use file system-level locking functions, so that they can easily be ported to any of file systems. Guy [13] proposed a replica modification approach, a replica is designated either a master or a secondary replica. 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 [14] proposes two coherence protocols viz., lazy-copy and aggressive-copy. Replicas are only updated as needed, if someone accesses it in the lazy-copy based protocol. Huang et al. [15] proposed the differentiated replication to improve accessing performance and replicas availability. They make effort on performance, availability and consistency. But maintenance consistency algorithm does not take storage capacity into account. Some replicas are not to be accessed for a long time by grid users, which will waste the free space of storage device. Dirk et al. [16] proposed a high-level replica consistency service, called Grid Consistency Service (GCS). The GCS allows updating file synchronization and consistency maintenance. The literature proposed several different consistency levels ranging from entirely synchronized data to loosely synchronized data. Gird users can choose different consistency services dynamically adjusting replicas consistency degree. Jiafu Hu et al. [17] proposed an asynchronous model, despite the system failure or network traffic congestion, this model avoids the replicas inconsistency in grid environment. The consistency problem mentioned in the literature could be classified into two kinds, one was the metadata replica consistency and the other one the data content consistency. Chang [18] proposes architecture called Adaptable Replica Consistency Service (ARCS) with the capability of dealing with the replica consistency problem. It has the better performance and load balance for file replication in Data Grids.

III. PROPOSED APPROACH

A. Architecture

Fig 1 shows a group of File Replication Servers (FRSs) along with the Requesting Nodes (RNs). The figure represents the logical connections between FRSs and RNs. FRSs will communicate/exchange information with each other as and when required.

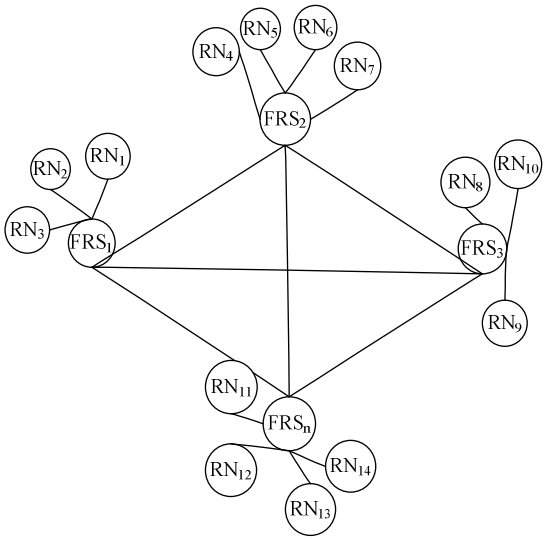

Fig. 1. FRS architecture.

An FRS can be 'local' or 'remote' with respect to a RN. For a RN, A FRS is said to be 'local' if RN is directly connected to FRS, and all the other FRSs are said to be 'remote'. So, in a group of n FRSs, each RN has 1 'local' FRS and (n-1) 'remote' FRSs.

B. Data Structure

1) File Details Table

TABLE I. FILE DETAILS TABLE

Attribute	Type
Filename	String
Filesize	Long Integer
Request Count	Integer
Replication Threshold	Integer
Valid	Boolean
Lock	Integer
Primary Server ID	Integer
Replicating	Boolean
Getting	Boolean
Timestamp Array	Integr Array
Peers	Integer Array

- Filename: Name of file.
- Filesize: Size of file.
- Request Count: Number of requests for a particular file a server handles.
- Replication Threshold: Maximum number of requests for a particular file a server can handle, at any time after that file will be replicated on other server.
- Valid: It is a Boolean variable that signifies whether the file content is valid or not.
- Lock: It is an integer variable that signifies that some server node is updating a file and hence has acquired lock on the file. Lock is significant only to the primary server node of the file.
- Primary Server ID: It is an integer variable. This specifies the ID of the primary server node (or the parent node) of the file.
- Replicating: It is a Boolean variable that signifies that the server node is replicating the file on some other node.
- Getting: It is a Boolean variable that signifies that the server node is getting a VALID file from some other peer node. This variable gets set only when the server node has an INVALID file.
- Timestamp Array: It is an array of integer variables. It stores the timestamp at which the clients requested for the file from this particular server node.
- Peers: It is an array of integer variables and stores the ID of the servers which has the replica of the file.

2) Peer Server Table

TABLE II. PEER SERVER TABLE

Attribute	Type
Peer ID	Integer
Peer IP	String
Peer Port	Integer

- Peer ID: ID of peer file replicating server
- Peer IP: IP of peer file replicating server
- Peer Port: Port number of peer server

C. Replication Mechanism

1) Description: Replication

Replication of a file is done by a server node on one of its peer server nodes when the number of client requests currently being handled on the server node crosses a threshold value, for a particular file. This condition is checked through Request Count and Replication Threshold fields of the File Details Table. When the request count crosses the replication threshold, the server replicates the file on one of its peer server nodes.

2) Description: Check file validity

A server node has an INVALID file. It receives a client request for that file and founds that all the servers having the VALID file are busy.

- In such a scenario, the server node having the INVALID file (here, INVALID file is checked using the Valid field of the File Details Table) explicitly requests the VALID file from one of the busy server nodes.

D. Hybrid Consistency Mechanism

When a node updates a file replica, it multicasts an "Update_Notification" message to all the nodes having the replica of the file that has been updated. All the recipient nodes then perform a local calculation (i.e. the calculation is performed on the node) independent of the other nodes. This calculation is done to decide whether the replica on the node needs to be immediately updated or invalidated. Since both the actions (immediate-update and invalidate) are possible on the available replicas of the updated file, the presented approach is hybrid in nature. The logic and details of the calculation has been described in the next section.

a) Calculation of Updation-Threshold on each node

When a node '*s*' receives an "Update_Notification" message for a file f, it calculates the Updation-Threshold of the file '*f*' (UT^s_f). If the number of client requests for the file f that the node has received in some units of time (known as "monitoring-time") exceeds UT^s_f, the replica of the file f on the node is a candidate of immediate-update. Otherwise, the replica is invalidated.

UT^s_f = *[Replication-Threshold * (Monitoring-Time / Avg. Transfer Time of file f)] / x*

Where, *Replication Threshold:* is the maximum number of file requests that the node 's' can handle at any time. *Monitoring Time:* is the time period during which the number of file requests handled by node 's' has been monitored. *Average Transfer Time:* is the average time that the node 's' takes to transfer file f. 'x' is any constant depending on the admin of the server nodes. If 'x' is large then Hybrid approach is more inclined to write update and if it is low then hybrid approach is inclined to invalidate.

Since the decision to immediately-update or invalidate a file replica is made on the basis of number of past requests for the file, the hybrid approach is retrospective in nature. The logic behind being retrospective is that the same request pattern can be anticipated in the future.

IV. SIMULATION AND RESULTS

Comparison of Hybrid Approach with the Write-Update and Write-Invalidate Approaches

The comparison of the hybrid approach with the write-update and write-invalidate has been discussed with the help of a scenario under different conditions of request load from clients.

Scenario Description:

There are 4 nodes each having a replica of a file f. Let these nodes be S1, S2, S3, and S4. S1 updates the file f. Let the replication threshold (Th_{rep}) of each of the nodes be 150.

Now, we will discuss the interaction of these nodes with each other depending on following factors:

- Load on each node (of the requests received from the clients).
- Update policy used by S1 (the node that updates the file f) to propagate the updates to its peer nodes.

We shall compare the three update policies (Write-Invalidate, Write-Update, and Hybrid Policy) depending on the nature of the load on each node (i.e., the requests received from the clients).

The load in the cloud can be classified broadly as:

- Balanced Load: All the nodes receive almost the same number of client requests for a file.
- Unbalanced Load: The distribution of client requests received by the nodes differs significantly from each other.

Case 1: Balanced Load

Case 1.1: Less number of file requests

Each node receives 15 client requests for the file f. Number of requests received on each node is significantly lower than the replication threshold Th_{rep} i.e., we have a balanced but very light load.

Write-Invalidate Policy

After updating the file f, node S1 invalidates all the replicas of file f in the cloud. So, there is only one replica that is valid and that is with the node S1. So, all other nodes will redirect their client requests to node S1 for processing.

Requests handled by each node: The node S1 happens to handle 60 requests (its own 15 requests and 45 other requests redirected by nodes S2, S3, and S4). *No. of replicas updated*: No transfer of file f is needed from node S1 to any other node.

Write-Update Policy

After updating the file f, the node S1 immediately updates all the replicas of the file f in the cloud. So, we now have a valid replica of the file f with each of the four nodes. In this approach, none of the nodes needs to redirect its client requests to the node S1 as all the nodes have valid replicas.

Requests handled by each node: Each node handles 15 requests. *No. of replica updated*: 3 (From node S1 to all other nodes). Here, exactly N transfers take place, if there are N replicas of the file f. In this case, Write-Invalidate is better approach than Write-Update.

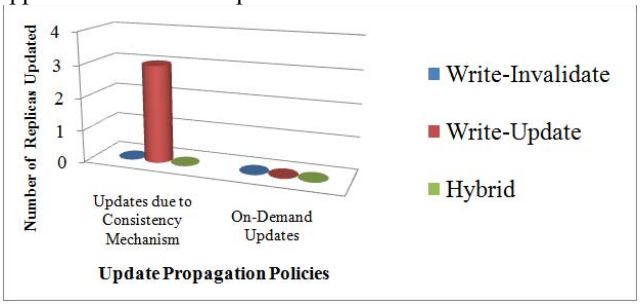

Fig. 2. Comparison of Three Policies when Number of Request is Less

Case 1.2: large number of file requests

Each node receives 130 client requests for the file f. Number of requests received on each node is near about the replication threshold Th_{rep} i.e., we have a balanced but heavy load.

Write-Invalidate Policy

After updating the file f, node S1 invalidates all the replicas of file f in the cloud. So, there is only one replica that is valid and that is with the node S1. So, all other nodes will redirect their client requests to node S1 for processing. Let node S3 redirects its 20 requests to node S1 after which the node S1 cannot accept any more requests (as the request count for file f will cross the threshold Th_{rep}). Thus, all of the nodes S2, S3, and S4 are forced to request a valid copy of file f from the node S1. Hence three updates need to be performed.

Requests handled by each node: S1 handles 150 requests and becomes overloaded. Nodes S2 & S4 handle 130 requests each, while node S3 handles 110 requests. *No. of replica updated*: 3 (From node S1 to all other nodes). Here, exactly N transfers take place if there are N replicas of the file f.

Write-Update Policy

After updating the file f, the node S1 immediately updates all the replicas of the file f in the cloud. So, we now have a valid replica of the file f with each of the four nodes. In this approach, none of the nodes needs to redirect its client requests to node S1 as all the nodes have valid replicas.

Requests handled by each node: Each node handles 130 requests. *No. of replicas updated*: 3 (From node S1 to all other nodes). Here, exactly N transfers take place if there are N replicas of the file f. In this case, Write-Update proves to be a better approach than Write-Invalidate.

In the two cases discussed above (Cases 1.1 and 1.2), the Hybrid Policy behaves like Write-Invalidate in Case 1.1, and Write-Update in Case 1.2. Thus for balanced load scenarios, the behavior of the Hybrid policy is optimal.

Case 2: Unbalanced Load

In the unbalanced load scenario, two nodes (let nodes S1 and S3) receive heavy load of client requests and are loaded with 130 requests each. The other two nodes (nodes S2 and S4) receive light load of client requests and are loaded with 15 requests each.

V. CONCLUSION

The paper presents a demand based file replication and hybrid consistency mechanism. Once the file has been replicated on 'n' File Replication Server (FRS), the issue is to maintain file consistency. Paper compares the proposed Hybrid Update Propagation policy with Write Invalidate and Write Update propagation policies. Comparison results shows that based on number of request received hybrid approach selects either write invalidate or write update propagation policy. When the request received for a particular file is low i.e. light load, in this case hybrid policy behaves in same manner as write invalidate does. Whereas, under high load scenario hybrid policy will behave like write update policy. Also in case of uneven request received by a server hybrid policy outperforms write invalidate and write update propagation policies in terms of number of replicas updated to fulfill the file request. After the consistency

mechanism has been executed and has updated the replica based on its update propagation policy, the policy is considered better than other policies, if lesser number of updates are performed. Results show that, depending on the load viz., light or heavy, hybrid policy updates or invalidates optimum number of replicas, so as to fulfill the file requests with minimum number of updates. Whereas for both the situation either light load or heavy load write invalidate will invalid all replica and write update will update all replicas, which is not desirable.

REFERENCES

[1] Cohen, E. and Shenker, S. 2002. Replication strategies in unstructured peer-to-peer networks. In ACM SIGCOMM'02 Conference, Aug. 2002.

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

[3] Wolfson, O., Jajodia, S. and Huang, Y. 1997. An adaptive data replication algorithm. ACM Transactions on Database Systems, 22(2):255–314.

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

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

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

[7] Hisgen, A., Birrell, A., Jerian, C., Mann, T., Schroeder, M. and Swart, G. 1990. 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] Rao, H. and Skarra, A. 1995. A transparent service for synchronized replication across loosely-connected file systems, 2nd International Workshop on Services in Distributed and Networked Environments, 5-6 June 1995, pp.110-117.

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

[11] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, April 1955, pp. 529–551.

[12] J. No, et al. “Data Replication Techniques for Data-Intensive Applications”, International Conference on Computational Science (4) 2006, pp. 1063-1070.

[13] 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.

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

[15] Changqin Huang, Fuyin Xu, and Xiaoyong Hu, “Massive Data Oriented Replication Algorithms for Consistency Maintenance in Data Grids”, ICCS 2006, Part I, LNCS 3991, pp. 838-841, 2006.

[16] Dirk Düllmann, Wolfgang Hoschek, Javier Jaen Martinez, Ben Segal, “Models for Replica Synchronisation and Consistency in a Data Grid”, Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10’01), Oct. 2001, pp. 67,

[17] Jiafu Hu, Nong Xiao, Yingjie Zhao, and Wei Fu, “An Asynchronous Replica Consistency Model in Data Grid”, Parallel and distributed processing and applications (ISPA 2005 Workshops), 2005, pp. 475-484.

[18] Ruay-Shiung Chang and Jih-Sheng Chang, “Adaptable Replica Consistency Service for Data Grids”, Third International Conference on Information Technology: New Generations (ITNG’06), 2006, pp. 646-651.