• **Computer and Communication Technologies** •

Research on Distributed Dynamic Replication Management Policy*

ZHOU Xu, LU Xian-liang, HOU Meng-shu, WU Jin

(School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 610054 China)

Abstract *This paper introduces replication management policies in distributed file system, and presents a novel decentralized dynamic replication management mechanism based on accessing frequency detecting named FDRM. In FDRM, in order to provide better system performance and reduce network traffic, system nodes scan their local replicas to monitor replicas' access pattern, and makes decision independently to add, delete or migrate replicas. In addition, the scanning interval of a replica is variable according to the accessing frequency to that replica, which makes FDRM more sensitive to the change of system behaviors, so that one can get better performance with less system overhead. Experiments show the efficiency and performance improvement of FSRM.*

Key words *distributed; replica management; dynamic; frequency; adaptive*

Replication of data is an essential component of distributed file systems and peer-to-peer systems. There are two major motivations for data replication: increasing data availability and improving system performance. Adding replicas for the files in the distributed file system allows the system to keep the ability of accessing file data in cases of nodes failure and network partition, and thus increases file reliability and availability. Also, if a file is replicated near the location where it is often accessed, communication costs could be reduced by reading the data nearby, thus the response time will be reduced and system performance will be enhanced. For example, FreeNet and Napster implement a replication policy to create replicas for a file based on its popularity, thus improve file's availability and accessing efficiency.

Distributed file system must provide a uniform view for multiple replicas of files, which requires special effort to maintain data consistency. With rapid increment of system scale, and increasing complexity of network environment, network costs and system overheads for data synchronization among file replicas are also increased greatly, which may impair the system performance[1, 2]. Under this circumstance, an optimal replication management mechanism for a large-scaled distributed file system is expected.

This paper presents a novel effective distributed dynamic replica management mechanism based on accessing frequency detecting named FDRM. The aim of FDRM is to improve system performance by dynamically adding or deleting file replicas to optimize the message traffic in the network. FDRM runs in a distributed manner. Every node makes its own decision independently to add, delete or move its replicas, and also adjusts replica degree of a file to balance the performance of both write and read. FDSM keeps eyes on a replica in a periodical manner, and the length of interval of each replica is variable according to the access frequency during lifetime of replicas. By this means, FDRM treats replicas of a file individually, and achieves better performance with low system overhead.

1 Related Work

In a distributed file system, there are two essential factors for replica management policy: replica degree and allocation of the replicas. Replica degree is the number of replicas of a given file, and allocation of the replicas decides the nodes of the system in which replicas must be physically stored. Replica degree and the allocation of the replicas together determine the replication scheme for a file, i.e., a subset of the nodes in the system which hold a copy of that file.

Generally, for a read-intensive file, a widely distributed replication scheme is mandated in order to increase the number of local reads and to decrease the

Received 2004-06-28

* Supported by the Electronic Industry Development Fund of MII "Multi-Function Network Server"

load on a center node. On the other hand, an update of an object is usually written to all replicas. So, for a write-intensive file, a wide distributed scheme slows down each write, and increases its communication cost. Therefore, in this case, a narrowly distributed replication scheme is mandated.

According to the moment at which decisions are taken, there are two major kinds of replication management policies: static replication policy and dynamic replication policy. A static replication policy decides the replica degree and the allocation of the replicas at the moment when the file enters the system, and this decision will not be changed during the lifetime of the file. This decision is usually based on probable access patterns, and has been extensively studied[3]. The static scheme performs well if the access patterns are known before, but this is not always possible, especially in a large-scaled distributed system environment. A dynamic replication policy takes its decision at the moment of file creation and this decision keeps changing during the lifetime of the file. For instance, both the replica degree and the allocation of the replicas are likely to change from time to time. The change of replication scheme is based on the changing pattern of file accesses as well as the load of system. By adapting the replication degree and the allocation of the replicas, dynamic replication policy can reduce the network traffic, and provide better system performance.

According to the method that dynamic replication policy uses to make decision for replica scheme, we can classify dynamic policy into two kinds: centralized solution and distributed solution. A centralized solution has to have a global view on the access pattern of all the replicas of a file, and makes its decision based on the information which is gathered in a centeral point. The benefit of centralized solution is simplicity. However, a centralized solution is likely to become a bottleneck for a large distributed file system and it is less resilient[4]. Instead, in a distributed solution, a node makes its decision of replica scheme individually based on its local information of this replica, and needs not to know the information of replicas on other nodes. Distributed solutions have two advantages over centralized ones. They respond to changes of read-write pattern in a timelier manner, since they avoid the delay involved in the collection of statistics, computation, and their overhead is lower because they eliminate the extra messages required in centralized cases[5].

Bartal presented an algorithm: each node maintains a counter for each file it accessed, and increases the counter by 1 with every read on the file, when the counter reaches a certain value *D*, node keeps the file as a new replica, then counter begins to decrease by 1 with every write on the file, when the counter drops to zero, the node delete its replica[4]. It's simple to realize Bartal's algorithm, but it is difficult to have a proper value for *D*. And any node accessing a file has to maintain a counter for the file no matter it has the replica or not, which obviously enlarges the range of managed nodes. Giacomo had another solution: a node in the system scans its replicas periodically, when it finds that the read/write ratio of a replica reaches a certain level, it invokes the dynamic replication algorithm to add or delete a replica[6]. This solution only involves the nodes which keeps the replica, and is easy to implement. But further test proves that the system performance is sensitive to the interval of periodicity[5]: if the file is not accessed frequently, keeping short periodical interval is wasteful and would lead an unnecessary load on the node, and if the file is intensively accessed, a long interval of periodicity could not response to the changes of system behaviors timely. In the algorithm of Swarup, a node maintains a deterministic finite state automaton (DFSA) per every neighbor per file to record each neighbor's access pattern for each file[7]. Node uses DFSA to predict coming access pattern of its replica, and determines the replication scheme in advance. This algorithm is adaptive to the system well and has good performance, but it is complicated to construct the DFSA and needs to maintain a large amount of information for both replicas and neighbors. The computation complicity will increase greatly when system scale enlarges.

2 Basic Model of FDRM

In this paper, we addresses on distributed dynamic replication policy. We have some assumptions: FDRM works in the read-one-write-all context, which means a file read involves only one replica and the write update is propagated to all replicas; the read request is satisfied locally or by the nearest replica; the read-write pattern during a period is predictable based on the pattern in the immediately preceding time period.

2.1 Rules for Replica Adding and Deleting

We assume that there are n replicas of a given file f in the system, $node_i$ (i=1, 2, $\cdots$, n) denotes n nodes which store these replicas, α denotes the system cost to satisfy a read request at the closest replica, and β denotes the cost for system to update a replica. Working in a distributed manner, FDRM uses local information of a replica to determine replication scheme. During a time interval t, a replica of f on $node_i$ maintains some counters to record requests to f both locally and remotely: r_{in} is the number of local read requests, $r_{\text{out},j}$ is the number of read requests from $node_j$, w_{in} is the number of local write requests, w_{out} is the number of update requests that $node_i$ receives.

By the third assumption above, we can suggest that the access pattern in next interval t' is the same as the pattern during current period. So the precondition for node$_i$ to add a new replica to another $node_j$ is that the overall costs after adding is less than the costs before adding:

$$r_{\text{out},j}\alpha + nw_{\text{out}}\beta + (n-1)w_{\text{in}}\beta > (n+1)w_{\text{out}}\beta + nw_{\text{in}}\beta \quad (1)$$

The precondition for $node_i$ to delete its replica of f is the overall cost after deleting is less than the costs before deleting:

$$nw_{\text{out}}\beta + (n-1)w_{\text{in}}\beta > r_{\text{in}}\alpha + (n-1)w_{\text{out}}\beta + (n-1)w_{\text{in}}\beta \quad (2)$$

In a real system, we can suggest $\alpha \approx \beta$, so Eqs.(1) and (2) can be simplified as

$$r_{\text{out},j} > w_{\text{out}} + w_{\text{in}} \quad (3)$$

$$r_{\text{in}} < w_{\text{out}} \quad (4)$$

2.2 Interval of Periodicity

After a period of time t, a node will make its decision of adding or deleting a replica. But as we mentioned above, the length of t is important to system's performance. In order to solve this problem, the intervals of periodicity in FDRM are variable and sensitive to the frequency of file access. The interval will be short when access is not frequent and the interval will become longer when frequency of file accessing increases. By this means, different parts of the replication scheme may have different scan frequencies.

We assume that there is a sequence of interval period $t_1, t_2 \cdots, t_n$ for a replica, A_i is the total number of access during the period t_i. If the current interval of periodicity is t_n, then the next interval t_{n+1} is

$$t_{n+1} = t_n \frac{A_{n-1}t_n}{A_n t_{n-1}} \quad (5)$$

3 Implementation of FDRM

We denote the current number of replicas of a file f as *replica_degree*. However, the value of *replica_degree* should be within a certain range. If it is too small or shrinks to zero, the availability of f could not be guaranteed. If it is too large, too much system resource will be consumed. So we set a lower limit *MIN* and an upper limit *MAX* for *replica_degree*[8]:

$$MIN \leqslant replica\_degree \leqslant MAX \quad (6)$$

Each $node_i$ that owns a replica of f maintains three counters r_{in}, w_{in} and w_{out} for f. $node_i$ also keeps a record *outside_read_list* of information of all outside reading during current period, including counters $r_{\text{out},j}$ recording the number of read requests f received from node$_j$. When node$_j$ reads the replica on $node_i$, *outside_read_list* of node$_i$ will be scanned to find out whether the replica has been accessed before. If there is a record of previous operation from node$_j$, node$_i$ increases corresponding $r_{\text{out},j}$, otherwise, node$_i$ adds a new record to *outside_read_list*, and sets $r_{\text{out},j}$ to 1.

In order to adjust the interval of periodicity t, $node_i$ maintains a record for the replica of f. In this record, we have current interval length t_n, previous interval length t_{n-1}, and the number A_{n-1} of all access that replica had received during last period. Then we can use Eq.(5) to determine the length of next interval t_{n+1}.

There are some special cases of interval of periodicity we should concern about. If there is no access during this period, then no matter how many accesses the replica had during last period, the length of next period should be extended to double length of current interval. If there is no access during the last period, then no matter how many accesses during current period, the length of next period should be shrinked to half length of current interval. To avoid intervals being too short or too long, we set a lower and upper limit for it.

$$MIN\_PERIOD \leqslant t \leqslant MAX\_PERIOD \quad (7)$$

At the end of each period of replica, $node_i$ decides what operation will be taken to the replica.

First, $node_i$ tries to add a replica to current

replication scheme. It scans the *outside_read_list* to find the last accessing node which satisfies Eq.(3) and copies a new replica to it. Because the node that had read access most recently may have further access, FDRM ensures that file will be copied to the node which most probably needs the file.

Then, $node_i$ will decide to delete its replica or not. Even when counters satisfy Eq.(4), $node_i$ could not delete the replica immediately. For there may be several replica in the system that satisfy the condition of Eq.(4), deleting without necessary consultation may cause *replica_degree* drops to below *MIN* or even to zero. To avoid this, when the condition of Eq.(4) is satisfied, the replica just changes its state to deleting other than deletes immediately, then the replica sends messages to other replicas of *f* to request permission to delete itself. On receiving a request message from other replicas, if a replica is also in the state of deleting, it ignores the request, otherwise it sends a permission message back. If the replica in deleting state receives *MIN* permissions, it deletes itself. The state of deleting will last till the end of next period. If replica could not receive enough messages by then, it returns to normal state. A replica in deleting state can provide service for other node as it is in normal state.

A replica always tries to add a replica first then tries to delete its own replica, which may cause a replica to add a new replica to another node at first then to delete itself. It has the same effect as file migration.

If actions of both adding and deleting a replica fail, and the *replica_degree* reaches *MAX*, FDRM will applies real migration process. *outside_read_list* will be scanned to find the first node *j* that satisfies the condition of Eq.(8), and replica will be moved to that node.

$$r_{\text{out},j} > r_{\text{in}} \tag{8}$$

There is another special cause: a file may be accessed intensively during a period of time, but after that period, it becomes less interesting and has no further accesses. At this time, *replica_degree* would have increased to a certain level or reached *MAX*, but there is no need to keep so many replicas for that file. To solve this problem, a replica checks its interval of periodicity before adding or deleting. If interval reaches *MAX_PERIOD* and has been kept for two continuous periods, replica turns its state to *deleting*, tries to delete itself.

At last, $node_i$ sets next interval of periodicity for the replica of file *f* by Eq.(5), and resets all the counters of the replica.

4 Experiment Result

4.1 Experiment Environment and Parameters

We constructed an experimental FDRM system with 15 machines distributed in campus. In order to make comparative test, we have applied three different replication management policies. One is FDRM policy, another is the policy developed in Ref.[6], which has a fixed interval of periodicity (we call it non-FDRM), and the last one is a static policy. The parameters of FDRM are: *MIN*=2, *MAX*=8, *MIN_PERIOD*=20 s, *MAX_PERIOD*=30 000 s. The *replica_degree* of static policy is 3.

We continuously monitored the operations on a certain file *f*, to find out the performance of each replication management policy. And because the performance of the system using non-FDRM policy is sensitive to the length of static interval, so we choose different interval to have our tests.

4.2 Fixed Interval of Non-FDRM is 20 s

In this test, the fixed interval of non-FDRM equals to the shortest interval of FDRM. We recorded the numbers of operations transmitted in the network in a period of time. These operations include all remote reads and write updates. The results are shown in the table below.

Tab.1 Result of test 1

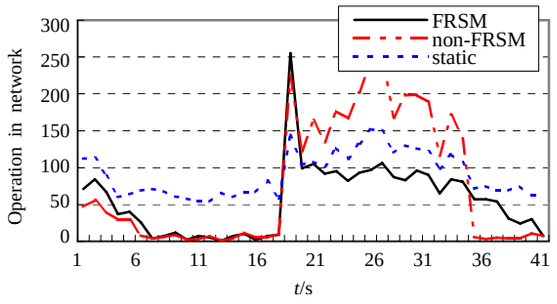
	FSRM	Non-FSR *M*	Static
Number of Operations	26 827	39 472	44 663

The number of operations in network of FDRM is only 60% of that of static policy, and is only 67% of that of non-FDRM, which shows an obvious improvement. Non-FDRM only reduced the operations in network by 5 191 less than that of static policy; performance enhancement is not very notable.

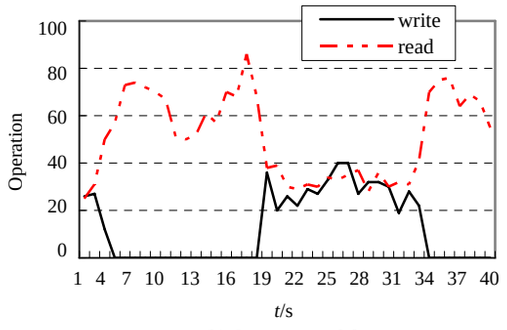
Fig.1 shows curves of operation numbers and *replica_degree* during a period of our test. The three curves in Fig.1a are operations in network of three replication policies respectively, the two curves in Fig.1b are read and write requests in the system respectively, and the two curves in Fig.1c show the changes of *replica_degree* of FDRM and non-FDRM.

In Fig.1a, we can find that there are some troughs

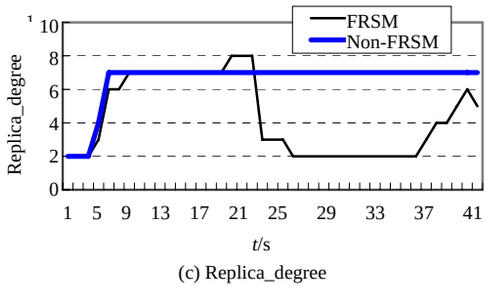
in the curves of FDRM and non-FDRM (for example, interval between 5~18), and these troughs in the two curves are almost in the same positions. According to Fig.1b, we learn that these intervals of troughs are the periods that system has intensive read accesses on *f*. And from Fig.1c, we find that during these intervals both FDRM and non-FDRM have a large number of replicas. It shows that FDRM and non-FDRM both realize the increasing of read accesses in the system, and raise their *replica_degree* accordingly to try to satisfy the requests locally, as a result, reduce the operations transmitted in the network.



(a) Operation in network



(b) Operation on life



(c) Replica_degree

Fig.1 Results of tests

In Fig.1a, we found some crests in curve of FDRM (for example, interval between 19~37), from Fig.1b we learne that these intervals of crests are the periods that system had intensive write accesses on *f*. From Fig.1c we learn that FDRM decrease its *replica_degree* automatically to avoid large number of update operations consuming the network resource. The crests of write operations in Fig.1b and the troughs of *replica_degree* of FDRM in Fig.1c are almost at the same position on the time axis. Investigating curve of FDRM in Fig.1a more carefully, we find there is a sharp peak at the beginning of each crests (for example, at the point of 19 on time axis). They are caused by sudden increase of write accesses in system, and at that moment FDRM still has a large number of replicas. But in Fig.2c, we can find FDRM adjust its *replica_degree* timely, which reduce the operations in network rapidly and form the sharp peak. All these evidences prove that FDRM's effectiveness on adapting to the changes of system behaviors and effectualness in improving the system performance.

Investigating the curve of non-FDRM in Fig.1a, we find that there are also some crests, but they do not drop to a proper level timely, instead, keep in a relatively high level. From Fig.1b, we know that is because the number of replicas does not decrease timely. The reason that causes the problem is mainly because the intervals of periodicity of all replicas are the same, so all the periods end almost at the same time. At that moment, all replicas find the increase of write accesses and send out their messages to ask for deleting permission. Because almost all replicas is in the state of deleting, they all ignore the permission request messages from other node, so none of them can get enough permissions, which cause the failure of replica deleting process. But for FDRM, because replicas have different period lengthes, which reduces the feasibility of deleting conflict, and makes the change of *replica_degree* more sensitive.

During the period of test, *f* in FDRM had 4.26 replicas averagely; while *f* in non-FDRM has 6.22 replicas, more than FDRM. Although the average replica number is bigger than the replica number 3 of static policy, FDRM still has much less operations in the network than static policy, which proves that FDRM can improve the system performance and reduce the network traffic effectively.

By recording the changes of intervals of one node in our system, we know that during the period of test, that node scanned its local replica of *f* 57 times, the longest interval is 2 233 s, the shortest interval is 20 s, and the average interval length is 176 s. But in non-FRSM, that node scanned the replica 500 times, much higher than FDRM. It shows that FDRM can provide better system performance with lower system overhead.

4.3 Fixed Interval of Non-FDRM is 40 s

We set the fixed scan interval of non-FDRM to 40 s. It is easy to predict that because of the increasing of interval length, non-FDRM will become less sensitive than when interval is 20 s. The table below is the result of the second test, we can find that FDRM keep good performance, but non-FDRM dose worse than last time and almost has the same performance as static policy.

Tab.2 Result of test 2

	FSRM	Non-FSR *M*	Static
Number of Operations	29 061	41 687	41 273

And during the period of the second test, FDRM and non-FDRM both keep 4 replicas of *f* averagely, that is to say FDRM has much better performance than non-FDRM on the condition of consuming the same amount of storage resources.

During the test, our monitored node scan the replica of file *f* 77 times, the longest interval is 2 507 s, the shortest interval is 20 s, and the average interval length is 131 s. non-FDRM scan 250 times, still much higher than that of FDRM.

5 Conclusions

In this paper, we have described a novel decentralized dynamic replication management mechanism based on accessing frequency detecting named FDRM. In FDRM, system node scans local replicas to detect their access pattern, and makes decision independently to add, delete or migrate a replica. In addition, the scanning interval of a replica is variable according to the accessing frequency to that replica, which makes FDRM more sensitive to the change of system behavior.

Experimental results show that FDRM can improve the system performance greatly and reduce the network traffic effectively without extra system overhead.

References

[1] Han H. Studies on Internet Oriented Distributed Massive File Storage[D]. Ph.D. Dissertation of Peking University, 2002

[2] Ranganathan K, Foster I. Identifying Dynamic Replication Strategies for a High Performance Data Grid[A]. International Workshop on Grid Computing[C]. Denver: Springer-Verlag, 2001. 75-86

[3] Dowdy D, Foster D. Comparetive models of the file assignment problem[J]. Computing Surveys, 1982: 14 (2): 287-313

[4] Bartal Y, Fiat A, Rabani Y. Competitive algorithms for distributed data management[A]. In Proc. 24th ACM Symp. On Theory of Computing[C]. NY: ACM Press, 1992. 39-50

[6] Cabri G, Corradi A, Zambonelli F. Experience of Adaptive Replication in Distributed File Sytems[Z]. Proc. of EUROMICRO-22, Prague, 1996. 459-466

[7] Acharya S, Zdonik S B. An Efficient Scheme for Dynamic Data Replication[R]. Brown University CS-93- 43, 1993

[5] Wolfson O, Jajodia S, Huang Y. An adaptive data replication algorithm[J]. ACM Transactions on Database Systems, 1997, 22: 255-314

[8] Ranganathan K, Iamnitchi A, Foster I. Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities[A]. Proceedings of the Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems[C]. Berlin: IEEE Computer Society, 2002

Brief Introduction to Author(s)

ZHOU Xu (周 旭) was bron in 1976. He is a doctoral postgraduate student at UESTC. His current interests include distributed computing, peer-to-peer technology and operation system. Email: xzhou@uestc.edu.cn

LU Xian-liang (卢显良) was born in 1943. He is now a professor at UESTC. His research interests include operating system, computer network. Email: xlu@uestc.edu.cn

HOUMeng-shu (侯孟书) was born in 1971. He is now pursuing Ph.D. degree at UESTC. His research interests include peer-to-peer network, network storage. Email:mshou@uestc.edu.cn

WU Jin (吴 劲) is now pursuing Ph.D. degree at UESTC. Her research interests include distributed computing, computer network. Email: wj@uestc.edu.cn