CSIT (March 2013) 1(1):29–49
DOI 10.1007/s40012-012-0002-0

ORIGINAL RESEARCH

# File replication and consistency maintenance mechanism in a trusted distributed environment

**Manu Vardhan · Dharmender Singh Kushwaha**

Received: 5 July 2012 / Accepted: 13 August 2012 / Published online: 12 October 2012


**Abstract** An effort has been made to propose an adaptive, trust based, on demand and reliable file replication approach for distributed environment. The work proposes an active replication mechanism for file replication; file access and performance transparency to system, thereby ensuring replication decisions such as when to replicate, where to replicate, load status of the node selected for replication and avoidance of redundant replica on that node. The proposed model introduces File Replication Server (FRS) that replicates the file, when total number of request for it reaches the threshold value, adhering to replication decision. If all replicas of file are not synchronized with each other, this drives system into inconsistent state. To keep files consistent, changes made at one replica of file should be reflected on other replicas in minimum possible time. Unlike conventional approaches viz., Write Update and Write Invalidate, in which either all or any fixed master replica is updated respectively, the proposed approach transfers the role of master replica to the last modified replica. File replica is updated on-demand by only propagating the required partial updates. The master replica immediately computes the file content modifications but propagates them only on-demand. Once a node gets itself registered, it becomes the part of elite FRS community. This paper introduces basic trust parameters and adaptive factors in computing trustworthiness of peer FRS, namely, frequency of the request for a particular file that a FRS perform, registration type of node i.e., paid or unpaid, write operation not permitted if trust value of a FRS is less than threshold, feedback that a FRS gives about other FRS and authenticity of the session key. Simulation results show that during high file request scenario for a particular file, frequently accessed files are replicated on other FRSs dynamically and file request is redirected in transparent manner, thus reducing request completion time by about 28.78–47.24 % as compared to FTP. Experimental results shows that various factors viz., file size, size of modifications and number of replicas to be updated affects the time to propagate the changes to other replicas. Percentage reduction in time for propagating these changes varies from 31.56 to 78.1 %. Similarly, the reduction in time for updating replicas simultaneously varies from 78.17 to 85.37 %. Though the percentage increase in time required for acquiring the keys to access the services with trust based security model varies from 11.94 to 17.49 %, the same is compensated by reduction in time for updating replicas. In order to ascertain stability of proposed model, Calculus of Communicating System (CCS) for proposed model is compiled on CWB-NC tool and bisimulation equivalence is proved for proposed file replication model.

**Keywords** Consistency · Distributed systems · File replication · Update propagation · Write Update

## 1 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

M. Vardhan (✉) · D. S. Kushwaha
Department of Computer Science & Engineering, MNNIT, Allahabad, India
e-mail: vardhanmanu@gmail.com

D. S. Kushwaha
e-mail: dsk@mnnit.ac.in

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 exists, that are used to communicate the changes made in the file, to its replicas. Two major policies used for update propagation are Write Update and Write Invalidate. Both these policies have their own advantages and disadvantages.

Multiple copies of file engender the issue of confliction between file replicas; it is called inconsistency. To keep the replicated file consistent, it is mandatory that all the replica of a file strictly have the same content at a time; it means changes made in a file must be reflected on other replicas in zero time (immediately). It is practically impossible because of network delays. Conventional approaches for modification propagation maintain a master replica and any changes made in any other replica have to be propagated immediately to the master replica. This overhead is reduced in proposed approach by notifying all replicas that the latest modified replica of file (f) is (r1) and is now the new master replica or new owner for file (f).

As a result, system can handle large number of requests as several replicas 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 File Replication Servers (FRSs).
- File access frequency based, dynamic file replication on the peer FRS.
- 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.

Reputation systems [1] provide a way for building trust by utilizing community based feedback about past experiences of peers to help making recommendation and judgment on quality and reliability of the transactions. The challenge of building such a reputation based trust mechanism in distributed system is how to effectively cope with various malicious behaviors of peers such as providing fake or misleading feedback about other peers. Another challenge is how to incorporate various contexts in building trust as they vary in different communities and transactions. Further, the effectiveness of a trust system depends not only on the factors and metrics for building trust, but also on the implementation of the trust model. Most existing reputation mechanisms require a central server for storing and distributing the reputation information. It remains a challenge to build a decentralized trust management system that is efficient, scalable, and secure in both trust computation and trust data storage and dissemination.

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 discusses about the proposed trust based approach. Section 4 discusses the replication and consistency maintenance model followed by simulations and results in Section 5. Finally, Section 6 concludes the work followed by references.

## 2 Related work

Cohen and Shenker [2] 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. [3] replicate objects both on insertion and retrieval on the path from the initiator to the target, mainly for anonymity and availability purposes. Wolfson et al. [4] address 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. Similarly, locality aware file replication is proposed by Cheng and King [5] to ensure data reliability and availability through the parallel I/O system. To ensure synchronized file replication across two loosely connected file systems, a transparent service model has been developed by Rao and Skarra [6] that propagate the modification of replicated files and directories from either file system. Primary-copy (master–slave) approach for updating the replicas says that only one copy could be updated (the master), secondary copies are updated lazily. There is only one replica which always has all the updates. Consequently the load on the primary copy (master replica) is large. Domenici [7] discuss several replication and data consistency solutions, including Eager (Synchronous) and Lazy (Asynchronous) replication, Single-Master and Multi-Master Model, pull-based and push-based consistency mechanism. Author presents various replication and consistency maintenance algorithms to deal with huge scientific data. Guy et al. [8] propose a replica modification approach, a replica is designated either as 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 and Xu [9] propose 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. [10] propose the differentiated replication to improve access performance and replicas availability. They make effort on performance, availability and consistency. But consistency maintenance algorithm does not take storage capacity into account. Some replicas that are not accessed for a long time by grid users will waste the free space of storage device. Düllmann et al. [11] propose a high-level replica consistency service, called Grid Consistency Service (GCS). The GCS allows updating file and consistency maintenance. The literature proposes 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. Hu et al. [12] propose 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.

There are some recent research on reputation and trust management in distributed systems. Aberer and Despotovic [13] are one of the first in proposing a reputation based management system. However, their trust metric simply summarizes the complaints a peer receives and files and is very sensitive to the skewed distribution of the community and misbehaviors of peers. Chen and Singh [14] differentiate the ratings by the reputation of raters that is computed based the majority opinions of the rating. Adversaries who submit dishonest feedback can still gain a good reputation as a rater in their method simply by submitting a large number of feedback and becoming the majority opinion. P2PRep proposed by Cornelli et al. [15] is a protocol where servants can keep track of information about the reputation of other peers and share them with others. Their focus is to provide a protocol complementing existing protocols, as demonstrated on top of Gnutella. However, there are no formalized trust metric and no experimental results in the paper validating heir approach. Dellarocas [16] propose mechanisms to combat two types of cheating behavior when submitting feedback. The basic idea is to detect and filter out exceptions in certain scenarios using cluster-filtering techniques. The technique can be applied into feedback-based reputation systems to filter out the suspicious ratings before the aggregation. Another work is Eigen Trust proposed by Kamvar et al. [17]. Their algorithm again focuses on a Gnutella like P2P file sharing network. They based their approach on the notion of transitive trust and addressed the collusion problem by assuming there are peers in the network that can be pre-trusted. While the algorithm showed promising results against a variety of threat models, we argue that the pre-trusted peers may not be available in all cases and a more general approach is needed. Another shortcoming of their approach is that the implementation of the algorithm is very complex and requires strong coordination and synchronization of peers. A proposal specifically attempted to address the issue of quality of the feedback. A recent paper by Miller et al. [18] propose a mechanism, based on budget balanced payments in exchange for feedback, that provides strict incentives for all agents to tell the truth. This provides yet another approach to the problem of feedback trustworthiness. However, such a mechanism is vulnerable to collusion. Sen and Sajja [19] propose a word-of-mouth reputation algorithm to select service providers. Their focus is on allowing querying agent to select one of the high-performance service providers with a minimum probabilistic guarantee. Zacharia and Maes [20] propose an approach that is an approximation of game-theoretic models and studied the effects of feedback mechanisms on markets with dynamic pricing using simulation modeling. The basic idea is to generate trust values describing the trustworthiness, reliability, or competence of individual nodes, based on some monitoring schemes. Such trust information is then used for malicious node detection [21], and even time synchronization. Although there are a few works studying one or several possible vulnerabilities [22] in ecommerce and P2P applications, there is a lack of systematic treatment of this problem. There has been a great deal of confusion on the topic of trust. Many researchers recognize trust as an essential element in security solutions for distributed systems [23]. But it is still not clear what trust is and how exactly trust can benefit network security [24]. Keynote is a well-known trust management system [25] designed for various large and small-scale Internet-based applications. It provides a single, unified language for both local policies and credentials. It has several shortcomings with respect to trust negotiation [26]. Trust-Builder [27] provides a broad class of negotiation policies, as well as a policy- and language-independent negotiation protocol that ensures the interoperability of defined policies within the Trust-Builder architecture. Gwertzman and Seltzer [28] states that most wide-area replication schemes are client initiated. Decisions on when and where to replicate files are made without the benefit of the server's global knowledge of the situation. Author believe that the server should play a role in making these replication decisions, and propose a geographical push-caching as a way of bringing the server back into the loop. Hitoshi et al. [29] propose a file clustering based replication algorithm for grid file systems. The

algorithm groups files according to a relationship of simultaneous accesses between files and stores replicas of the clustered files into storage nodes, to satisfy expected most of future read access times to the clustered files and replication times for individual files being minimized under the given storage capacity limitation. Hisgen et al. [30] examines the Echo distributed file system. The primary goals of Echo are to explore issues of scaling, availability, and performance. For scaling and uniformity of access, Echo provides a global, hierarchical name space. Replication is used for availability. Performance is achieved by distributed caching on clients and by using a log on the file server to reduce disk seeks. Hurley and Yeap [31] establish that file replication and migration can be utilized simultaneously to potentially provide significant performance benefits over a system, without file migration or replication. File replication can be viewed as a natural extension to file migration, and thus, a dynamic file replication policy based on an established file migration heuristic is derived.

## 3 Proposed approach

### 3.1 Trust based security service design

This section gives an overview of Trust Management Service and discusses the main components of the system. Also identifies the functionalities and interdependency between the components. Once a node gets itself registered, it becomes the part of elite FRS community denoted as Service Provider (SP) or Service Requestor (SR). Figure 1 presents the components of the Trust Management Service. Major features of different modules are discussed below:

TNI: Trusted Node Identifier
SR: Service Requestor
SP: Service Provider
- - - -: Nodes registered with TNI
____: Shows the logical connection between nodes

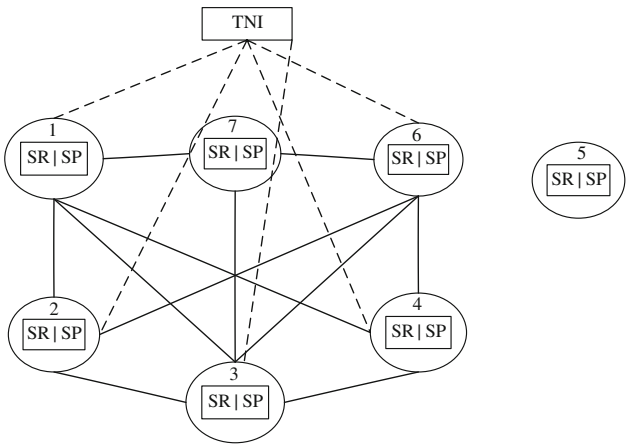



**Fig. 1** Security model for distributed

### 3.2 Message types

- $M_1$: SUKREQUEST (SeqNo, DestNode, TOS)
- $M_2$: SUKREPLY (SeqNo, SUK, TrustValue)
- $M_3$: PAIDREG ()
- $M_4$: UNPAIDREG (Referral)
- $M_5$: ACK (Type, Message)
- $M_6$: GET (E (SUK, Request))

### 3.3 Acknowledgement types

See Table 1 .

### 3.4 Data structure used

Table 2 shows the data structure maintained by TNI. Various parameters are described as below:

*Node_ID* shows the ID of the node registered with TNI. *Trust value* is the trust of a particular node. *Last update* shows the latest modification date of the trust value. *Permissions* identifies the operation (read and write) that a node can perform on a file. *Refferal_ID* identifies the ID of the node that refers an unregistered node for registration. *Paid* this field identifies the whether the node registration is of type paid or unpaid.

**Table 1** Acknowledgement types

| Type | Acknowledgement |
|---|---|
| 0 | Everything is fine |
| 1 | Node does not exist in trust manager map |
| 2 | Session key mismatch |
| 3 | Session key expire |
| 4 | Referral untrusted |
| 5 | Referral not exist |
| 6 | Session key does not exist |
| 7 | File does not exist |
| 8 | Invalid request |
| 9 | Does not have proper access right |
| 10 | Already registered |
| 11 | Proceed to paid registration |
| 12 | You did not sign SLA |
| 13 | Registration successful |
| 14 | Node already registered |
| 15 | Must wait for minimum rejoining time |

Data structure maintained in Table 3 is used to identify the frequent file request behavior. Node_ID is used to identify the node information from which repeated request for a particular file is received. Count: this field gives the number of times a file is requested in a particular time span. Filename is the name of the file for which frequent requests are received. Last request time is the last access time of a file.

### 3.5 Trust based security mechanism

TNI keeps the log of the registered nodes. As shown in Fig. 2, Service Requester (SR) node requests Session Usage Key (SUK) from TNI to access the service from the SP. TNI provides the SUK based on the current trust value of both nodes i.e. SR & SP. SUK is for limited time period as defined by the system. Information (trust value, permission, file count) about the communicating nodes is maintained at TNI. SP provides access to the services (file read and write operation) based on current trust value of the requester node. SP monitors the behavior of requester node and informs TNI to update the trust value of requester node based on past request.

Flow graph for the SR is shown in Fig. 3. After the SUK is received by the service requester, connection is established between the SP and SR.

Figure 4 shows the flow of TNI. TNI checks whether the request is made for updating the trust value of SR or revocation of SUK. Accordingly, the requester node is informed in reply to the request made. Where RN is the Requesting Node and TV is trust value of RN.

Figure 5 shows the flow diagram of SP. After receiving the request from the SR, SP checks, if the request made is for file. If not SP confirms whether registration is required or not. If the request is for a file, SP decrypts the request using the SUK and proceeds.

**Table 2** Trust value information

| Fields | Data type | Size |
|---|---|---|
| Node_ID | String | 50 |
| Trust value | Long | 8 |
| Last update | Date | 12 |
| Permissions | Char | 4 |
| Refferal_ID | String | 50 |
| Paid | Boolean | 1 |

**Table 3** Frequent file request information

| Fields | Data type | Size |
|---|---|---|
| Node_ID | String | 50 |
| Count | Integer | 50 |
| Filename | String | 100 |
| Last request time | Date | 12 |

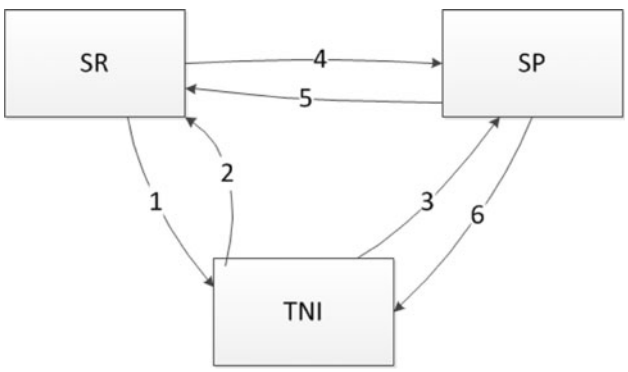



**Fig. 2** Working model of trust based security system

### 3.6 Trust based parameters

#### *3.6.1 Parameters considered for identifying node behavior*

1. Node doesn't exist.

1.1 Node is not registered.
1.2 Node debarred because of malicious behavior.

2. Session key (SUK) expires.
3. Increase trust values if request is correct (Good behavior).
4. Decrease trust value if session key (SUK) mismatch.
5. Decrease trust value if 'file not exists'.
6. Decrease trust value if repeated request is made in a particular time span.
7. Warning against violation of access permissions, to the RN.
8. In case of unpaid registration, decrease trust value of referee over wrong referral. If trust value decreases to minimum value (bad mouthing attack).
9. In case of unpaid registration, referee node must have the required trust value.
10. In case of unpaid registration, if trust value reaches 0, in that case node will be debarred.
11. In case of unpaid registration, initial trust value = -half the trust value of referee.
12. In case of paid, initial trust value = half of max trust value.
13. Node is rejoining within kicked out time period.

13.1. Rejoin by paid method.
13.2. Rejoin by unpaid method.

**Fig. 3** Flow graph of SR

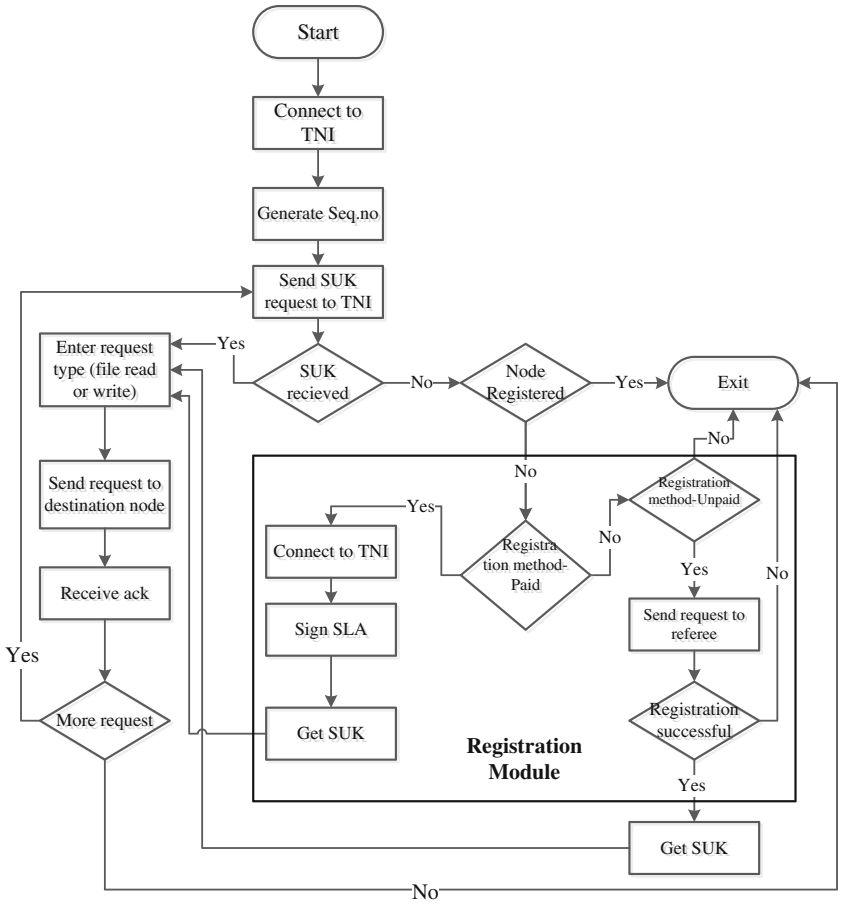
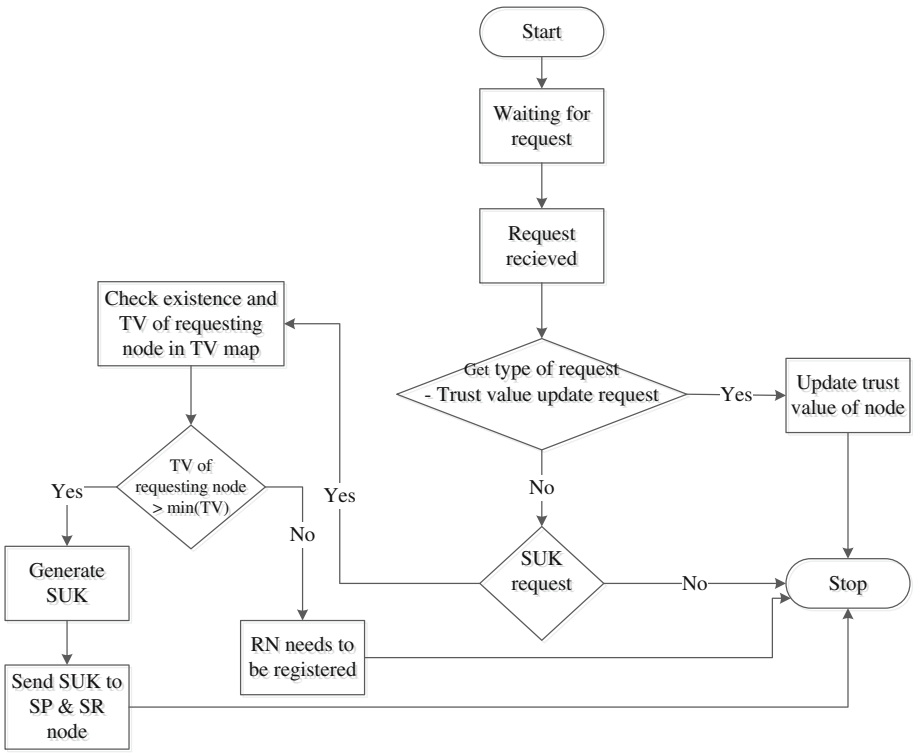



**Fig. 4** Flow graph of TNI



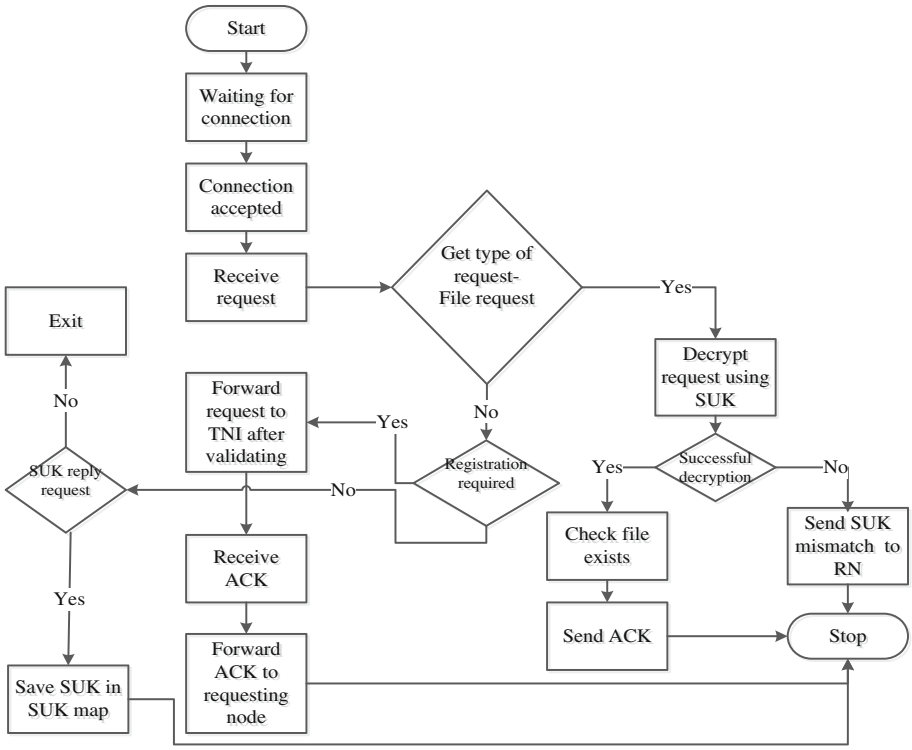



**Fig. 5** Flow graph of service provider

*3.6.2 Cases (for modifying trust value)*

1. When there is session key (SUK) mismatch, in that case decrease the trust value.
2. If RN requests the same file frequently,
   a. When first time file request is made, save the file name.
   b. Subsequent file request from same node will be identified by comparing the requested file name with saved file name.
      i. If both are same, count value is increased, if count value $\geq C_{fr}$ and $T_f - T_i < T_{fr}$ update trust value of requested node, else do nothing.
      ii. Else, update file name and save time and set count value.

(where $T_{fr}$ = frequent time period and fr = frequent request of same type).

3. This paper does not consider the case, if server is responding after a long time, since that may depend on load, communication link, size and type of requested file.

3.7 Interaction diagram

*3.7.1 Unregistered node*

Node1 ($N_1$) sends request to TNI that it wants to communicate with Node2 ($N_2$) and also requests SUK. TNI sends message to $N_1$, that $N_1$ is not a registered node (Fig. 6).

*3.7.2 Paid registration (direct)*

$N_1$ sends request to TNI that it wants to communicate with $N_2$ and also requests SUK. TNI sent message to $N_1$, that $N_1$ needs to register. $N_1$ sent registration request to TNI. Registration Successful message is send to $N_1$ by TNI (Fig. 7).

*3.7.3 Unpaid registration*

$N_1$ sends request to TNI that it wants to communicate with $N_2$ and also requests SUK. TNI sends message to $N_1$ saying that you are not a registered node so you cannot communicate with $N_2$. $N_1$ sends $N_2$ a request message saying that it wants to register. $N_2$ forwards $N_1$ request to TNI. TNI replies to $N_2$ that registration is successful. $N_2$ forwards

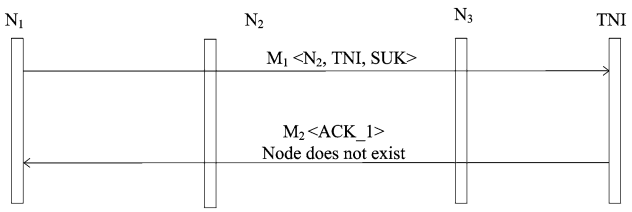



**Fig. 6** Not a registered node

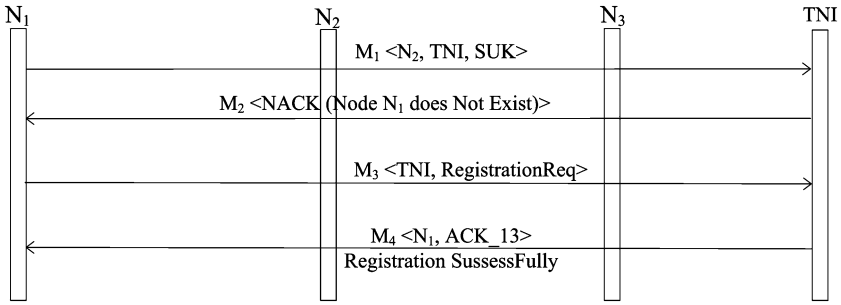



**Fig. 7** Paid registration (direct)

this message to $N_1$. Now $N_1$ can communicate with TNI and $N_2$ (Fig. 8).

### *3.7.4 Session usage key distribution*

$N_1$ sends request to TNI that it wants to communicate with $N_2$ and also requests SUK. TNI sends trust value of $N_1$ and $N_2$ to $N_2$ and $N_1$ respectively and also the requested SUK (Fig. 9).

### *3.7.5 Session usage key mismatch*

$N_1$ sends request to TNI that it wants to communicate with $N_2$ and also requests SUK. TNI sends trust value of $N_1$ and $N_2$ to $N_2$ and $N_1$ respectively. $N_1$ sends a request message to $N_2$. SUK mismatch between $N_1$ and $N_2$ is found by $N_2$. $N_1$ and $N_2$ sends message to TNI to update the trust value of $N_2$ and $N_1$ respectively (Fig. 10).

### *3.7.6 Block write access*

$N_1$ sends request to TNI that it wants to communicate with $N_2$ and also requests SUK. TNI sends message back to $N_1$ providing the SUK and trust value of $N_2$. TNI also sends message to $N_2$ providing $N_1$ SUK as well as trust value of $N_1$. $N_1$ sends message to $N_2$ requesting the file in write mode. $N_2$ replies to $N_1$ that $N_1$ does not have enough trust value to access the file in write mode. $N_1$ sends a message to TNI to update trust value of $N_2$. $N_2$ sends a message to TNI to update trust value of $N_1$ (Fig. 11).

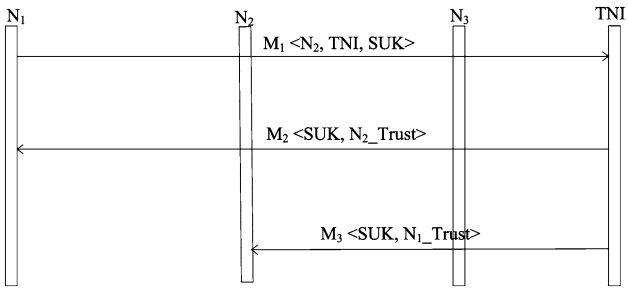



**Fig. 9** Session key distribution

### *3.7.7 File not found*

$N_1$ sends request to TNI that it wants to communicate with $N_2$ and also requests SUK. TNI sends message back to $N_1$ providing the SUK and trust value of $N_2$. TNI also sends message to $N_2$ providing $N_1$ SUK as well as trust value of $N_1$. $N_1$ sends message to $N_2$ requesting the file in write mode. $N_2$ replies to $N_1$ that $N_1$ does not have enough trust value to access the file in write mode. $N_1$ sends a message to TNI to update trust value of $N_2$. $N_2$ sends a message to TNI to update trust value of $N_1$ (Fig. 12).

### *3.7.8 Frequent request*

Frequent File Request Detection Algorithm

1. Get a request.
2. Check request type.
3. If request is a file request,

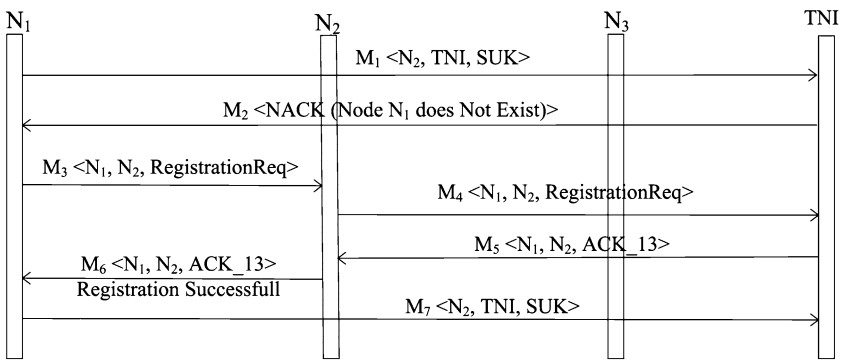



**Fig. 8** Unpaid registration

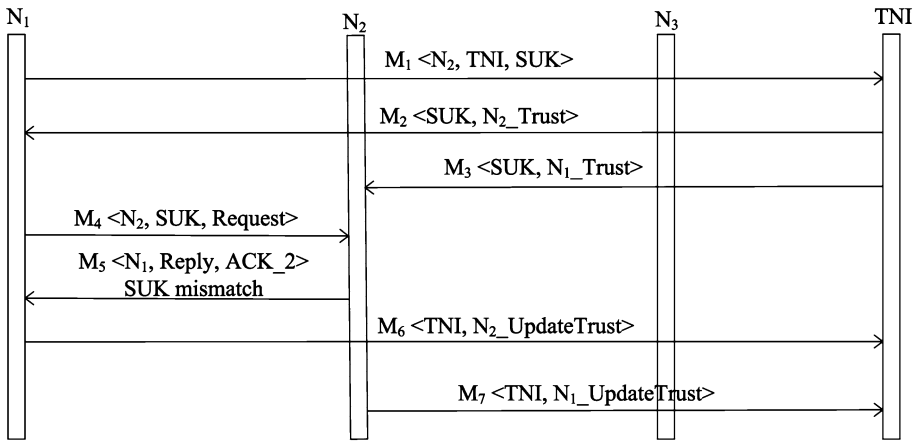



**Fig. 10** Session key mismatch

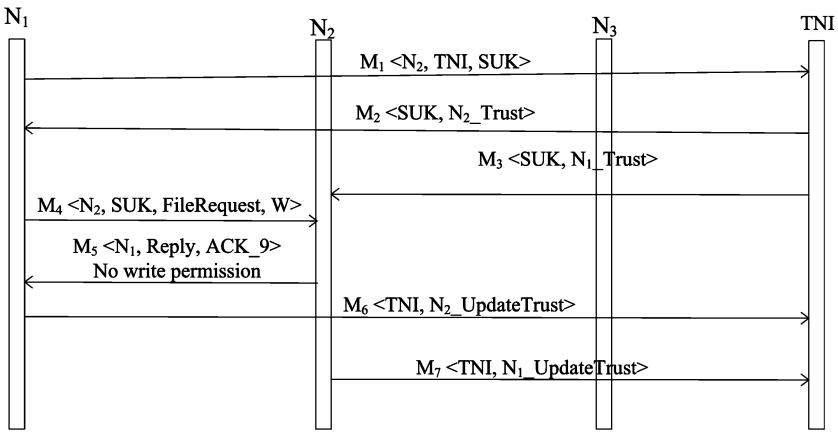



**Fig. 11** No write access

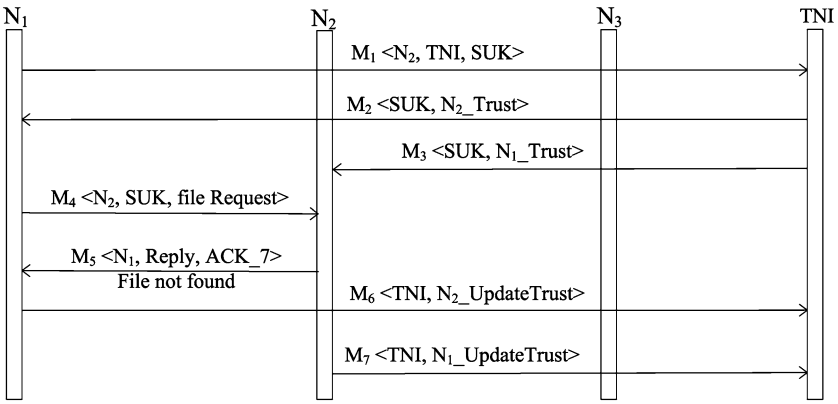



**Fig. 12** File not found

3.1 Check if previous request from same node exist.
3.2 If previous request is same as current request then check for maximum request within specified time period.
3.3 If previous request does not exist then, store the request node and request in a map which has node_id as key.

$N_1$ sends request to TNI that it wants to communicate with $N_2$ and also requests SUK. TNI sends message back to $N_1$ providing it the SUK and trust value of $N_2$. TNI also sends message to $N_2$ providing it SUK as well as trust value of $N_1$. $N_2$ identifies that $N_1$ is frequently requesting the files. $N_2$ replied with ACK_0. $N_1$ sends message to TNI to update the trust value of $N_2$. $N_2$ sends message to update trust value of $N_1$ (Fig. 13).

### *3.7.9 Session key expire*

$N_1$ sends request to TNI that it wants to communicate with $N_2$ and also requests SUK. TNI sends trust value of $N_2$ to

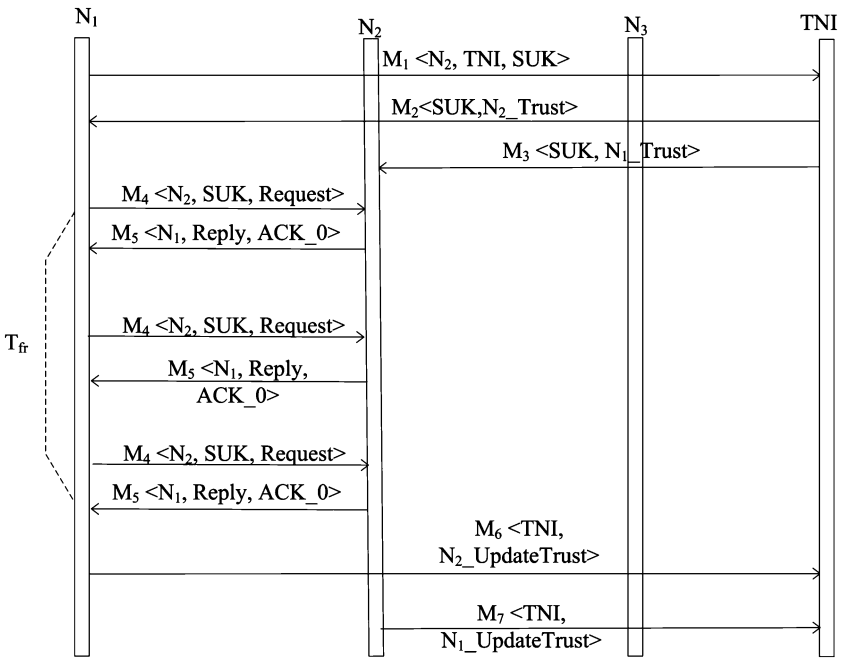



**Fig. 13** Frequent request

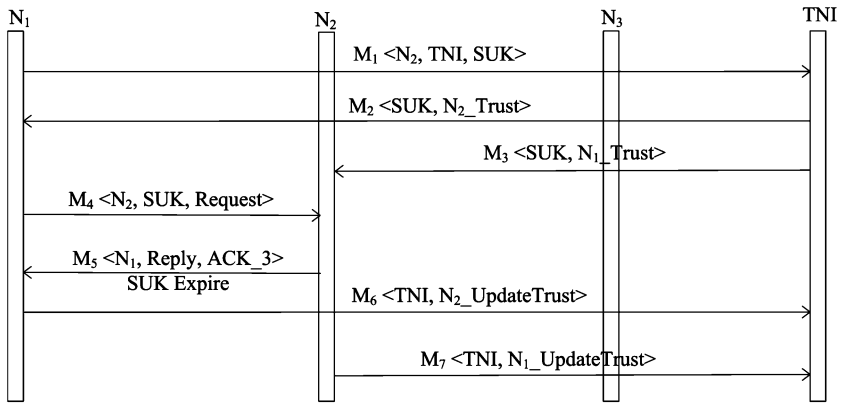



**Fig. 14** Session key expire

$N_1$. TNI sends trust value of $N_1$ to $N_2$. $N_1$ sends a request to $N_2$. SUK between $N_1$ and $N_2$ expires, hence the service cannot be accessed by $N_1$. $N_1$ sends message to TNI to update the trust value of $N_2$. $N_2$ sends message to TNI to update the trust value of $N_1$ (Fig. 14)

To validate the proposed security model, Calculus of Communicating System (CCS) is written and its Observational equivalence is proved using the Concurrency Workbench of the New Century (CWB-NC) that provides different techniques for specifying and verifying finite-state of concurrent systems.

## 3.8 Observational equivalence of trust based security mechanism

Terms used: suk = Session usage key; sukreq = SUK request; SC = Simple Client; SS = Simple Server; freq = file request; skey = Session Key; utrust = update trust request; utvalue = Update trust value; ack = Acknowledge; sreq = Session Key Request; prreq = Paid Registration Request; uprreq = UnPaid Registration Request; TNI = Trust Node Identifier; refreq = Referral Registration Request; STGS = simple Ticket Granting Server (Fig. 15).

### *3.8.1 Simple security model*

Definition of simple client

$$SC \overset{def}{=} \text{`sukreq.suk.SC} + \text{`freq.ack.SC} \quad (1)$$

Definition of Simple Ticket Granting Server

$$STGS \overset{def}{=} \text{sukreq.`suk.`suk.STGS} \quad (2)$$

Definition of simple SP (server)

$$SS \overset{def}{=} \text{freq.`ack.SS} \quad (3)$$

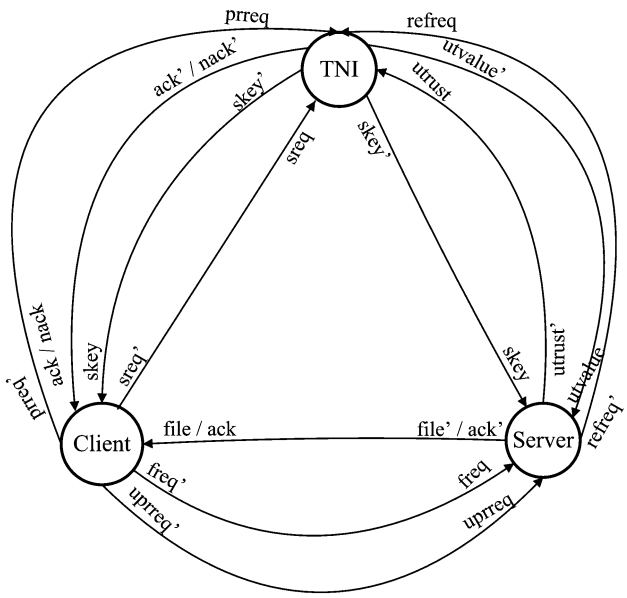



**Fig. 15** Replication security state diagram

Definition of simple security system

$$\text{SSYSTEM} \stackrel{\text{def}}{=} \text{SC} \mid \text{STGS} \mid \text{SS} \quad (4)$$

*3.8.2 Trust based model*

Definition of Client

$$\text{CLIENT} \stackrel{\text{def}}{=} \text{`sukreq.suk.CLIENT} + \text{`freq.ack.CLIENT} \quad (5)$$

Definition of Trust Node Identifier

$$\text{TNI} \stackrel{\text{def}}{=} \text{sukreq.`suk.`suk.TNI} + \text{utrust.`utvalue.TNI} \quad (6)$$

Setting internals for security modules

$$\text{RSI} \stackrel{\text{def}}{=} \{ \text{utrust, utvalue} \} \quad (7)$$

Definition of security service provider (server)

$$\text{SERVER} \stackrel{\text{def}}{=} \text{`utrust.utvalue.SERVER} + \text{freq.`ack.SERVER} \quad (8)$$

Definition of Security System

$$\text{SYSTEM} \stackrel{\text{def}}{=} (\text{CLIENT} \mid \text{TNI} \mid \text{SERVER}) \setminus \text{RSI} \quad (9)$$

3.9 MU Calculus

1. Whenever a RN makes a File request, eventually there will always be an acknowledgement to the RN
   $\text{P1} = \text{AG}((\text{not} < \text{-freq} > \text{tt}) \setminus / (\text{AF}(< \text{-ack} > \text{tt})))$
2. Whenever a RN makes a Session Key Request, eventually there will always be a Session Key or Nack to the RN
   $\text{P2} = \text{AG}((\text{not} < \text{-sreq} > \text{tt}) \setminus / (\text{AF}((< \text{-nack} > \text{tt}) \setminus / (< \text{-skey} > \text{tt}))))$
3. Whenever a node makes a Trust Value request, eventually there will always be a Updated Trust Value to the node
   $\text{P3} = \text{AG}((\text{not} < \text{-utrust} > \text{tt}) \setminus / (\text{AF}(< \text{-utvalue} > \text{tt})))$
4. Whenever a node makes a Paid Registration request, eventually there will always be an ACk or Nack to the node
   $\text{P4} = \text{AG}((\text{not} < \text{-prreq} > \text{tt}) \setminus / (\text{AF}((< \text{-ack} > \text{tt}) \setminus / (< \text{-nack} > \text{tt}))))$
5. Whenever a node makes a unpaid Registration request, eventually there will always be an Ack or Nack to the node
   $\text{P5} = \text{AG}((\text{not} < \text{-uprreq} > \text{tt}) \setminus / (\text{AF}((< \text{-ack} > \text{tt}) \setminus / (< \text{-nack} > \text{tt}))))$
6. Whenever there is a session key to the RN, there is a prior Session Key Request from the RN to TNI
   $\text{P6} = \text{A}((\text{not} < \text{-skey} > \text{tt}) \text{ W } (< \text{-sreq} > \text{tt}))$
7. Whenever there is a Referral Registration Request from Server, there is a prior Unpaid Registration request from the RN to TNI
   $\text{P7} = \text{A}((\text{not} < \text{-refreq} > \text{tt}) \text{ W } (< \text{-uprreq} > \text{tt}))$
8. Whenever there is a file to the RN, there is a prior file request from the RN to SERVER
   $\text{P8} = \text{A}((\text{not} < \text{-file} > \text{tt}) \text{ W } (< \text{-freq} > \text{tt}))$
9. Whenever there is a Updated Trust Value to the Node, there is a prior Update Trust request from the Node to TNI
   $\text{P9} = \text{A}((\text{not} < \text{-utvalue} > \text{tt}) \text{ W } (< \text{-utrust} > \text{tt}))$
10. There is a possibility of a Session Key Request from RN, followed by a session key from the TNI. This sequence of actions may also repeat infinitely
    $\text{P10} = \max \text{X} = < \text{t} > < \text{-sreq} > < \text{t} > < \text{-skey} > \text{X}$
11. There is a possibility of a file request from RN, followed by a ACK from the Server. This sequence of actions may also repeat infinitely
    $\text{P11} = \max \text{Y} = < \text{t} > < \text{-freq} > < \text{t} > < - \text{ack} > \text{Y}$

12. There is a possibility of a file request from RN, followed by a Update Trust request from server followed by Updated Trust Value from the TNI then followed by ack from Server. This sequence of actions may also repeat infinitely

$$\text{P12} = \max Z = <\text{t}><\text{-freq}><\text{t}><\text{-utrust}><\text{t}><\text{utvalue}><\text{t}><\text{-ack}> Z$$

Having established the trust model, we can safely proceed to provide services i.e. access to file read and file write to the requestor. For this a Trust based File Replication and Consistency model is proposed, with a view that communication between FRS is now secured and all malicious activities carried out by any FRS will be observed and notified to TNI, which in turn will lead to FRS deregistration based on its trust value.

## 4 Replication and consistency maintenance model

### 4.1 File replication model

Figure 16 shows a group of FRSs along with the nodes and these nodes are termed as RNs when they request for a particular file in a distributed environment. RN only has read access on the file. RNs cannot perform write operation to modify the file. File can be modified only by the FRS. The figure represents the logical connections between FRSs and RNs. FRSs will communicate/exchange information with each other as and when required.

An FRS can be 'local' or 'remote' with respect to RN. For RN, 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 one 'local' FRS and $(n-1)$ 'remote' FRSs.

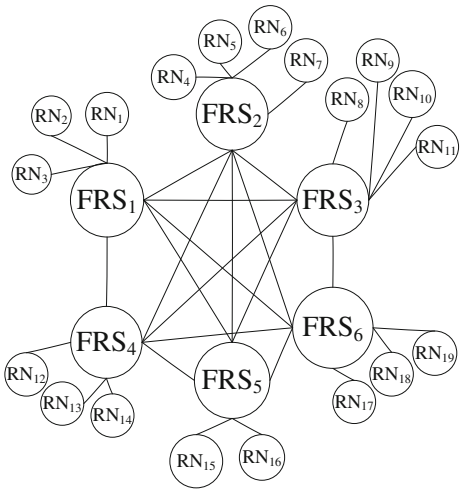



**Fig. 16** Proposed scenario

### 4.2 Data structure used

*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 FRS is getting a VALID file from some other peer FRS. 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 FRS that has the replica of the file (Table 4).

*Peer FRS ID*: ID of peer File Replication Server. *PeerFRS IP*: IP address of peer File Replication Server. *Peer FRS Port*: Port address of peer File Replication Server (Table 5).

**Table 4** 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 | Integer Array |
| Peers | Integer Array |

**Table 5** Peer FRS table

| Attribute | Type |
|---|---|
| Peer FRS ID | Integer |
| Peer FRS IP | String |
| Peer FRS Port | Integer |

### 4.3 Proposed replication mechanism

Each FRS receives a file request from the RN and based on its current load status, handles the request. Frequently accessed files are replicated on other FRS when the request count for a particular file reaches the threshold value. The various states of FRS are described below:

- Ready: File is present on the FRS and the Request Count for the file is less than the threshold value.
- Busy: File is present on the FRS and the Request Count for the file is equal to the threshold value.
- File Not Found: File is not present on the FRS.

The handling of the request takes place as shown in the flow diagram (Fig. 17).

Now, to understand the working of FRS in a much better way, few scenario's are discussed in the next section.

### 4.4 Replication scenarios

The various scenarios presented in this section explain the complete file replication model (FRM). The scenarios described below involve 3 FRSs $S_1$, $S_2$, $S_3$ and one RN $N_1$. The messages exchanged during the communication between FRSs and RN are described below:

- $M_1$: This is a request message and involves the request for file, resource_FRS_list, replication and status of other FRS. The listserver is the request for all the filenames, FRS IP and FRS Port address from the Local FRS.
- $M_2$: This is the status message of FRS. The different status is ready, busy and file not found.
- $M_3$: This message denotes the sending of the file contents to the RN or FRS, or the sending of the IP address, Port address of remote FRSs and resource_FRS_list present on the local FRS.
- $M_4$: This message involves the IP and Port address of the remote FRS from which the RN establishes the connection to receive the replicated file.
- $M_5$: Reply acknowledgement (RACK) after the file has been replicated successfully.

#### *4.4.1 Case 1: Local FRS $S_1$ cannot fulfill the request and looks for a remote FRS that can fulfill the file request*

As shown in Fig. 18, $N_1$ establishes connection with FRS $S_1$ and sends resource_FRS_list request (message $M_1$) to it. On successful connection, resource_FRS_list is received

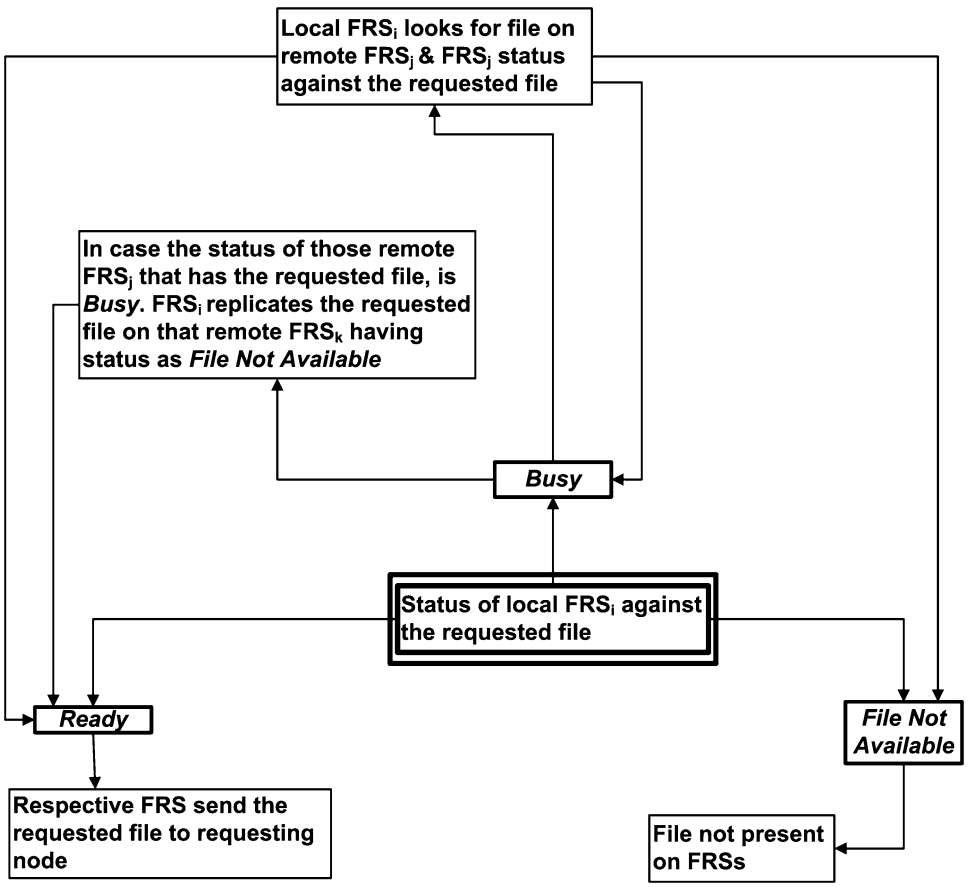



**Fig. 17** Flow diagram for replication

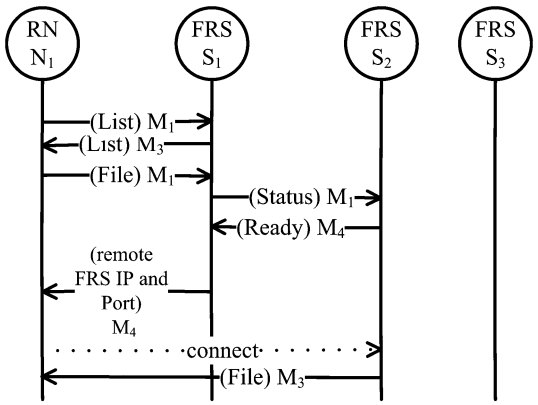



**Fig. 18** Remote FRS S2 handles the request

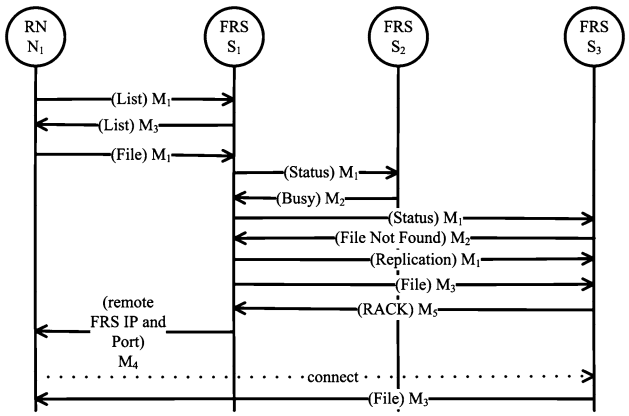



**Fig. 19** Remote FRS S3 handles the request

($M_3$) by $N_1$. $N_1$ sends file request ($M_1$) to the $S_1$. $S_1$ sends the status request message ($M_1$) to remote FRS $S_2$. $S_2$ sends status as 'ready' ($M_2$) to $S_1$. $S_1$ send IP and Port address of $S_2$ (message $M_4$) to $N_1$. $N_1$ receives the file from $S_2$.

#### *4.4.2 Case 2: Local FRS $S_1$ replicates the file on remote FRS $S_3$*

As shown in Fig. 19, $N_1$ establishes connection with FRS $S_1$ and sends resource_FRS_list request ($M_1$) to it. On successful connection, resource_FRS_list is received ($M_3$) by $N_1$. $N_1$ sends file request ($M_1$) to $S_1$. The status of S1 is busy and so it sends the status request message ($M_1$) to remote FRS $S_2$. $S_2$ sends status as 'busy' ($M_2$) to $S_1$. $S_1$ sends the status request message ($M_1$) to remote FRS $S_3$. $S_2$ sends status as 'file not available' ($M_2$) to $S_1$. $S_1$ sends the replication request message ($M_1$) to $S_3$. $S_1$ creates the file replica ($M_3$) on the $S_3$. $S_3$ sends RACK message ($M_5$) to $S_1$. $S_1$ sends the IP and Port address of the $S_2$ ($M_4$) to $N_1$. $N_1$ receives the file from $S_3$.

Now, after creating the file replica on more than one server, there arises a need to maintain consistency among all the replicas of a file. If a file is modified at any FRS, those changes need to be propagated to those FRS on which the replica is present. For this a partial update propagation mechanism for maintaining file consistency is proposed in the next section.

### 4.5 Update propagation mechanism for maintaining replica consistency

Time $T_t$ depends on the size of file to transfer. Instead of replacing stale replica of file f1 (r2, $\overline{f1}$) with updated replica (r1) of file (f1) i.e. (r1,f1) located on $FRS_1$, only the changes made to the file are extracted and propagated i.e. $\Delta$ ((r1,f1), (r1, $\overline{f1}$)), where (r1, $\overline{f1}$) is now old copy of (f1) on $FRS_1$. These changes will be stored in Diff file denoted by D(f1, sequence_no, timestamp) and are propagated to stale replica (r2) of file (f1) on $FRS_2$. After applying these changes on stale copy (r2, $\overline{f1}$) it will be updated to (r2,f1) using join operation $\Sigma$((r2, $\overline{f1}$), D(f1, sequence_no, timestamp)). Time required to extract modification is denoted by $t_\Delta$, time to join the Diff file with stale file is $t_\sum$ and time of propagation for D(f1, seq_no, timestamp) is $t_{pD}$, so total time to reflect the changes made to (r1, f1) in (r2, $\overline{f1}$) is $T_{rf}$.

$$T_{rf} = t_\Delta + t_{pD} + t_\sum$$

$T_{rf}$ = total time required to update the replica on $FRS_i$ ($ri$,$\overline{f1}$), $t_\Delta$ = time required to extract modification content and store them in Diff file (D), $t_{pD}$ = time required to propagate Diff file/s D from (r1) to (r2), $t_\sum$ = time for joining stale file ($\overline{f1}$) with Diff file/s (D), $T_{cft}$ = time required to transfer whole (complete) file

In case of one replica, proposed approach is beneficial if and only if $T_{rf} < T_{cft}$. If there are more than one replica of

**Table 6** Data structure maintained on FRS

| File name | Owner ID | Last write time stamp | Modification file | | | |
|---|---|---|---|---|---|---|
| F1 | FRS1 | F1($t_{lw}$) | D (f1, 1, timestamp) | D (f1, 2, timestamp) | D (f1, 3, timestamp) | …D (f1, n, timestamp) |
| F2 | FRS2 | F2($t_{lw}$) | D (f2, 1, timestamp) | D (f2, 2, timestamp) | D (f2, 3, timestamp) | …D (f2, n, timestamp) |
| F3 | FRS4 | F3($t_{lw}$) | D (f3, 1, timestamp) | D (f3, 2, timestamp) | D (f3, 3, timestamp) | |
| F4 | FRS5 | F4($t_{lw}$) | D (f4, 1, timestamp) | D (f4, 2, timestamp) | | |
| Fn | FRSm | Fn($t_{lw}$) | D (fn, 1, timestamp) | D (fn, 2, timestamp) | D (fn, 3, timestamp) | …D (fn, n, timestamp) |

file, than for updating each replica r*n* where ($n > 1$), will take $t_{pD} + t_{\sum}$, as the previously existing *Diff file/s* (D) can be propagated to all the replicas.

In previous approaches every file has a primary replica and other replicas are considered as the secondary replica, this primary replica is called the master replica [8]. When a replica of file is updated on secondary replica, than primary copy have to be updated immediately. With this approach there is need to wait until file write operation completes on secondary copy and then transfer this updated file to the master replica. In proposed approach as soon as a replica get request for write operation it notifies other replicas about the new master replica. As all replicas knows that new master copy is the replica on which last write operation done. So there is no need to update any other replica immediately. Every FRS maintains a data structure given below in Table 6. The entries in data structure keep track of information, when a file was last modified ($t_{lw}$) and by which FRS. Detailed working of consistency mechanism for a file replica on a FRS is given in Table 3.

Flow diagram for maintaining file consistency is given in Fig. 20.

To validate the proposed model, CCS is written and its Bisimulation equivalence is proved using the Concurrency Workbench of the New Century (CWB-NC) that provides different techniques for specifying and verifying finite-state of concurrent systems.

### 4.6 Bisimulation equivalence of replication model

Stability analysis of FRM using a process algebraic approach is carried out in this section. Transition systems [32] 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. Meaning of the symbols used in the CCS [33] is described as follows:

- SPN: Stands for Simple Provider Node. This denotes the Server Node of the No-Replication model.
- SRN: Stands for Simple Requesting Node. This denotes the Client Node of the No-Replication model.
- NR: This denotes the No-Replication Model.

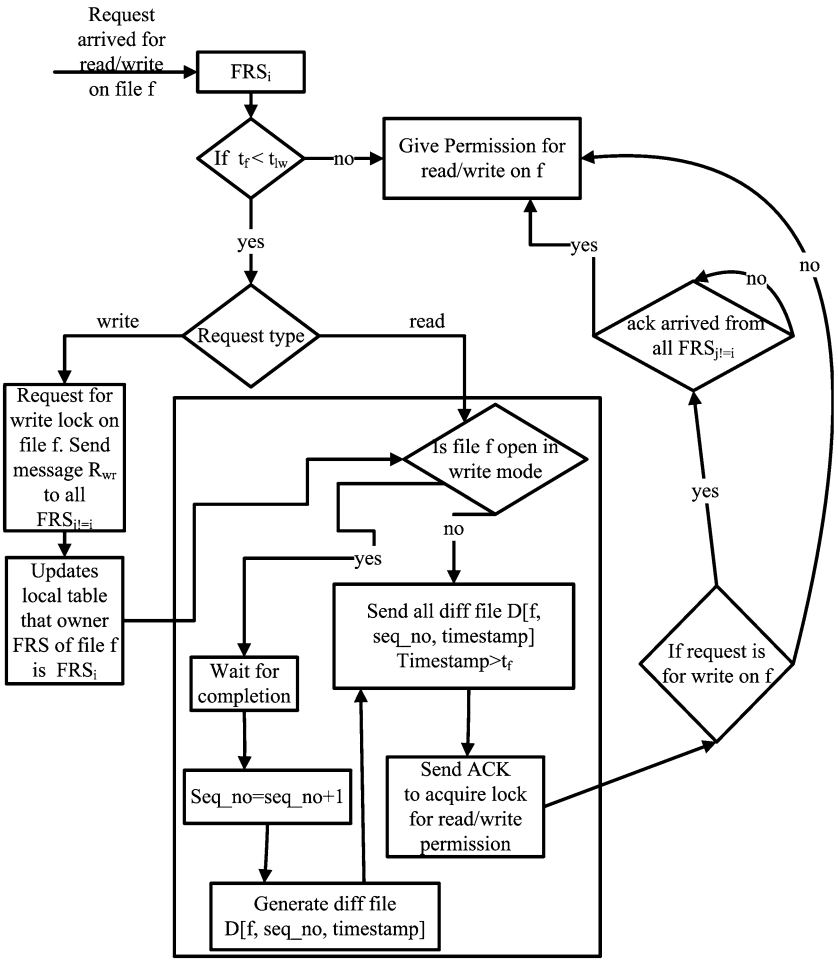



**Fig. 20** Flow graph for maintaining file consistency

- FRS: Stands for File Replication Server. This denotes the Server Node of the p model.
- RRN: Stands for Replication Requesting Node. This denotes the Client Node of the proposed replication model.
- RI: This is the set of internal actions for the proposed replication model.
- The symbol in CCS (‘) denotes the output actions and the rest of the actions denote the inputs.

#### 4.6.1 Definition of simple provider and requesting node

*Definition of simple provider node (SPN)*: provides the file to the RN, without performing any file replication and changes its state back to initial state i.e. SPN (Fig. 21).

$$\begin{aligned} \mathrm{SPN} &\overset{\text{def}}{=} \mathrm{listReq}.\text{‘}\mathrm{listSent}.\mathrm{SPN} \\ &+ \mathrm{get}.\text{‘}\mathrm{ready}.\text{‘}\mathrm{fileContent}.\mathrm{SPN} \end{aligned} \tag{10}$$

*Definition of Simple Requesting Node (SRN)*: request a file from the simple server node and changes its state back to initial state i.e. SRN (Fig. 22).

$$\begin{aligned} \mathrm{SRN} &\overset{\text{def}}{=} \text{‘}\mathrm{listReq}.\mathrm{listSent}.\mathrm{SRN} \\ &+ \text{‘}\mathrm{get}.\mathrm{ready}.\mathrm{fileContent}.\mathrm{SRN} \end{aligned} \tag{11}$$

*Model for simple server with no replication (NR)*:

$$\mathrm{NR} \overset{\text{def}}{=} (\mathrm{SPN} \mid \mathrm{SRN}) \tag{12}$$

#### 4.6.2 Definition of FRS and requesting node

*Definition of FRS*: fulfils the file requests, performs the file replication and changes its state back to initial state i.e. FRS.

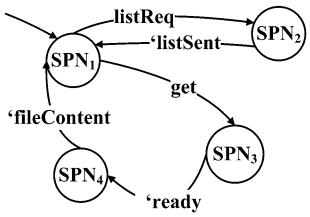



**Fig. 21** Simple provider node

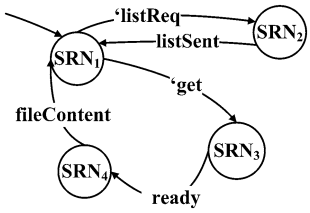



**Fig. 22** Simple requesting node

$$\begin{aligned} \mathrm{FRS} &\overset{\text{def}}{=} \mathrm{listReq}.\text{‘}\mathrm{listSent}.\mathrm{FRS} \\ &+ \mathrm{head}.\text{‘}\mathrm{no}.\mathrm{FRS} + \mathrm{put}.\mathrm{fileContent}.\mathrm{FRS} \\ &+ \mathrm{get}.(\text{‘}\mathrm{ready}.\text{‘}\mathrm{fileContent}.\mathrm{FRS} \\ &+ \text{‘}\mathrm{head}.\mathrm{no}.\text{‘}\mathrm{put}.\text{‘}\mathrm{fileContent}.\text{‘}\mathrm{newfrs}.\mathrm{FRS}) \end{aligned} \tag{13}$$

*Definition of Replicating Requesting Node (RRN)*: requests a file from FRS and changes its state back to initial state i.e. RRN

$$\begin{aligned} \mathrm{RRN} &\overset{\text{def}}{=} \text{‘}\mathrm{listReq}.\mathrm{listSent}.\mathrm{RRN} \\ &+ \text{‘}\mathrm{get}.(\mathrm{ready}.\mathrm{fileContent}.\mathrm{RRN} + \mathrm{newfrs}.\mathrm{RRN}) \end{aligned} \tag{14}$$

*Setting internals for replicating module*

$$\mathrm{RI} \overset{\text{def}}{=} \{\ \mathrm{head},\ \mathrm{put},\ \mathrm{no},\ \mathrm{newfrs}\ \} \tag{15}$$

*Definition of replicating module*

$$\mathrm{R} \overset{\text{def}}{=} (\mathrm{FRS} \mid \mathrm{RRN}) \setminus \mathrm{RI} \tag{16}$$

Above mentioned CCS is compiled on CWB-NC and bisimulation equivalence is proved between File Replication and no-replication model.

#### 4.6.3 MU Calculus of replication model

1. Whenever a RN makes a List request, a List will be sent to the RN.

$$\mathrm{P1} = \mathrm{AG}\ (\mathrm{not} < - \text{‘}\mathrm{listReq} > \mathrm{tt} \setminus/ \mathrm{AF}\ ([\mathrm{listReq}] < \mathrm{listSent} > \mathrm{tt}))$$

2. Whenever a RN makes a Get request, there exists a possibility that a ready message is received by the RN.

$$\mathrm{P2} = \mathrm{AG}\,(\mathrm{not} < - \text{‘}\mathrm{get} > \mathrm{tt} \setminus/ \mathrm{EF}\,([\text{‘}\mathrm{get}] < \mathrm{ready} > \mathrm{tt}))$$

3. Whenever a RN makes a Get request, there exists a possibility that a newfrs message is received by the RN.

$$\mathrm{P3} = \mathrm{AG}\,(\mathrm{not} < - \text{‘}\mathrm{get} > \mathrm{tt} \setminus/ \mathrm{EF}\,([\text{‘}\mathrm{get}] < \mathrm{newfrs} > \mathrm{tt}))$$

4. Whenever a RN makes a Get request, there exists a possibility that a fileNotFound message is received by the RN.

$$\mathrm{P4} = \mathrm{AG}\ (\mathrm{not} < - \text{‘}\mathrm{get} > \mathrm{tt} \setminus/ \mathrm{EF}\ ([\text{‘}\mathrm{get}] < \mathrm{fileNotFound} > \mathrm{tt}$$

5. Whenever a RN makes a Get request, there exists a possibility that a serverBusy message is received by the RN.

$$\mathrm{P5} = \mathrm{AG}\ (\mathrm{not} < - \text{‘}\mathrm{get} > \mathrm{tt} \setminus/ \mathrm{EF}\ ([\text{‘}\mathrm{get}] < \mathrm{serverBusy} > \mathrm{tt}))$$

6. Whenever a RN makes a Get request, there exists a possibility that a ready message is received by the RN and the file Content thereafter.

   P6 = AG (not < − 'get > tt \/ EF ([' get] < ready > < fileContent > tt))

7. Whenever a RN makes a Get request, there exists a possibility that a head message is received by the RN.

   P7 = AG (not < -get > tt \/ EF ([get] < 'head > tt))

8. For every head request by the server, there is a possibility that a yes message is received.

   P8 = EF ([' head] < yes > tt)

9. For every head request by the server, there is a possibility that a no message is received.

   P9 = EF ([' head] < no > tt)

10. For every head request by the server, there is a possibility that a busy message is received.

    P10 = EF ([' head] < busy > tt)

11. For every no request received by the server, there is a possibility that a put message is sent by the server.

    P11 = EF ([no] < 'put > tt)

12. For every no request received by the server, there is a possibility that a fileNotFound message is sent by the server.

    P12 = EF ([no] < 'fileNotFound > tt)

13. For every put request by the server, there is a possibility that a fileReplicate message is received.

    P13 = EF ([' put] < 'fileReplicate > tt)

14. For every put request received by the server, there is a possibility that a fileReplicate message is also received.

    P14 = EF ([put] < fileReplicate > tt)

15. For every busy request received by the server, there is a possibility that a serverBusy message is sent.

    P15 = EF ([busy] < 'serverBusy > tt)

Finally, having discussed all this, next section presents the simulation and results obtained from it.

## 5 Simulation and results

### 5.1 Simulation results

As shown in Table 7, the simulation has been conducted for three cases using two, three and four FRSs (2FRS, 3FRS and 4FRS). Each RN requests for file $F$ of size 64.1 MB from $FRS_1$. The experiment is carried out considering three scenarios viz., 2FRS, 3FRS and 4FRS. Threshold varies depending on the number of FRS, here in case of 2FRS threshold is 40, 3FRS it's 27 and for 4FRS it's 20.

Table 8 shows the request completion time in seconds and the FRS that handles the request. Table shows that the average request completion time under replication scenario is 28.78–47.24 % less when compared to FTP and 4.9 % less when compared to no-replication scenario. Figure 23 shows the comparison in terms of request completion time between FTP and proposed replication mechanism (with 2FRS, 3FRS and 4FRS). When replication is done, average completion time for a request is always less than the average completion time under no replication operation. After reaching the threshold, based on the request received by $FRS_1$, it replicates the file on $FRS_2$. So the request handled by FRS 2 takes more time since this time is inclusive of replication overhead from $FRS_1$ to $FRS_2$. If there is no replication on $FRS_2$ and all request are handled

**Table 7** Experiment Configuration Table

| | Replication | No replication |
|---|---|---|
| Number of FRSs | 2,3,4 | |
| Number of FRS utilized | 2FRS: 2 | 1 |
| | 3FRS: 3 | |
| | 4FRS: 4 | |
| Initial File request load | 80 | 80 |
| Filename | F1 | F1 |
| File size | 64.1 MB | 64.1 MB |
| Number of file replica created | 2FRS: 1 | Zero |
| | 3FRS: 2 | |
| | 4FRS: 3 | |

**Table 8** Average request completion time (s)

| RN | FTP | No replication | Replication | | |
|---|---|---|---|---|---|
| | | | 2FRS | 3FRS | 4FRS |
| 1–10 | 388.94 | 361.495 | 412.097 | 268.222 | 250.301 |
| 11–20 | 404.702 | 364.881 | 389.419 | 316.205 | 345.888 |
| 21–30 | 416.188 | 395.397 | 346.867 | 271.195 | 267.495 |
| 31–40 | 409.011 | 417.939 | 184.859 | 231.905 | 267.273 |
| 41–50 | 428.998 | 376.007 | 212.006 | 136.231 | 73.045 |
| 51–60 | 414.934 | 409.104 | 229.352 | 160.803 | 197.858 |
| 61–70 | 436.131 | 420.128 | 265.578 | 134.674 | 214.395 |
| 71–80 | 435.843 | 426.312 | 334.723 | 240.041 | 187.78 |
| Average | 416.84 | 396.40 | 296.86 | 219.90 | 225.50 |
| | | 4.90 % | 28.78 % | 47.24 % | 45.90 % |

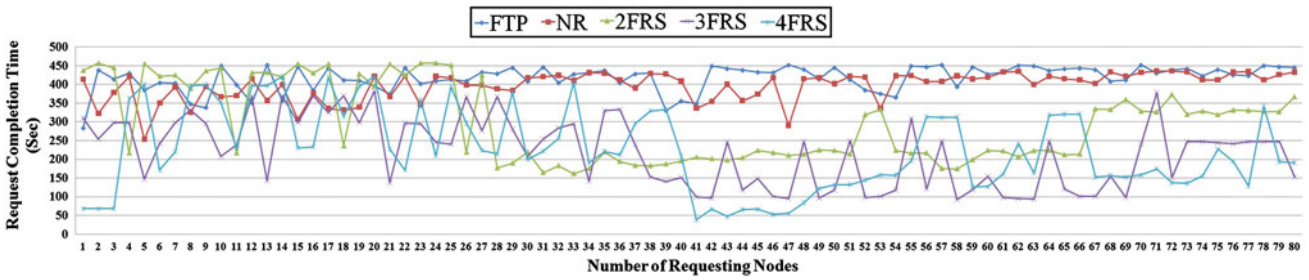



**Fig. 23** Comparison of request completion time

by $FRS_1$, the service time for each request increases significantly. This is shown in Table 8.

When the local FRS reaches its threshold value and replicates the file on some other FRS, the replication overhead is compensated by the following benefits:

- Avoid retransmission of request by the RN.
- Reduces latency in case of load above threshold.
- Ensures scalability.
- Provides fault tolerance capability.

The proposed model is simulated on Linux platform and LAN of 10.0 Mbps. Proposed approach is compared with the conventional Complete File Transfer approach (CFT). Proposed approach outperforms the CFT in terms of time and size of data transfer to replace a stale replica of file with the modified one.

**Table 9** Increasing number of replicas updated for Constant file size (5 MB) with modification size (100 kB)

| Number of replicas updated | Time with partial file transfer (in ms) | Time with CFT (in ms) | Percentage reduction in time for changes propagation |
|---|---|---|---|
| 1 | 129 | 591 | 78.17 |
| 2 | 196 | 1,088 | 81.98 |
| 3 | 263 | 1,585 | 83.40 |
| 4 | 319 | 2,071 | 84.59 |
| 5 | 375 | 2,564 | 85.37 |

Table 9 describes the details and possible cases for better understanding of this approach. It also shows that as the number of replicas to be updated increases total reduction in time for updating replicas increases. The corresponding graph is shown in Fig. 24.

Figure 25 shows that for a constant file size of 5 MB, a stale file replica can either be updated by replacing it with the latest modified file of 5 MB or by using the proposed approach. As the file content modification size increases, ratio of (file size/modification size) decreases and size of Diff file/s that needs to be propagated keeps on increasing. As a result percentage reduction in time for synchronizing the stale file replica with the latest modified file also decreases (Table 10).

Based on the various feature, a comparison matrix of the proposed model with the Kerberos [34], Pippal et al. [35] and Tao et al. [36] is shown in Table 11. Table shows that the proposed model provides few extra features in addition to those provided by the existing model.

Figure 26 shows the number of messages exchanged for node registration, SUK and finally for accessing various services.

Referral of a node by the help of Sec-SLAs, checking the access right of the file for providing the service (file read or write), checking the validity of the SUK and observing the malicious behavior on the basis of frequency of file requests. On the basis of these features average time is calculated with and without considering them. Results are tabulated in Table 12.

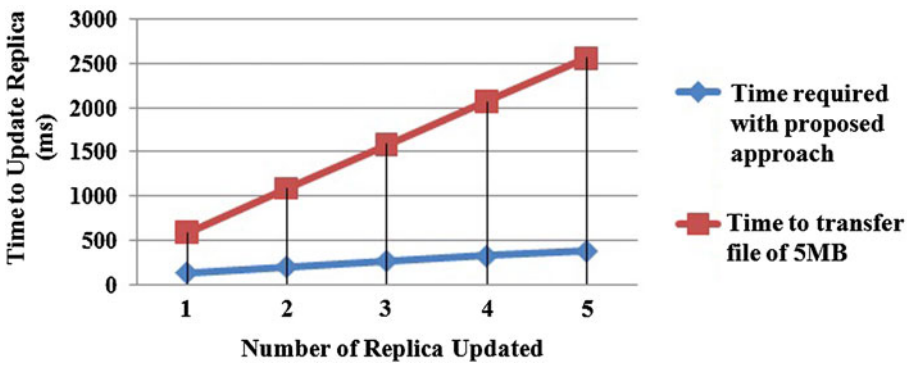



**Fig. 24** *Graph* corresponding to Table 9

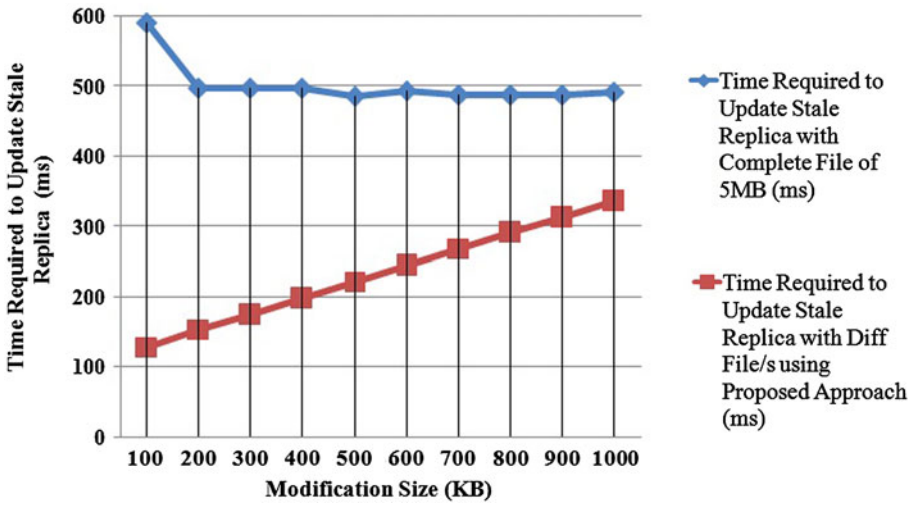



**Fig. 25** Time required to update the stale replica as file content modification size increases

**Table 10** Constant file size (5 MB) with increasing modification size

| File size (MB) | Modification content size (kB) | Size of diff file (kB) | Time to transmit complete file of 5 MB (ms) | Time to calculate diff file (ms) | Transfer time of diff file (ms) | Time for joining original file with diff file (ms) | Total time (ms) | % age reduction in time |
|---|---|---|---|---|---|---|---|---|
| 5 | 100 | 206.6 | 591 | 76 | 45 | 8 | 129 | 78.1 |
| 5 | 200 | 413.4 | 497 | 82 | 64 | 8 | 154 | 69.01 |
| 5 | 300 | 620.1 | 497 | 86 | 83 | 6 | 175 | 64.78 |
| 5 | 400 | 826.8 | 497 | 91 | 100 | 7 | 198 | 60.16 |
| 5 | 500 | 1,024 | 486 | 95 | 119 | 8 | 222 | 54.32 |
| 5 | 600 | 1,228 | 493 | 98 | 139 | 8 | 245 | 50.30 |
| 5 | 700 | 1,435 | 488 | 102 | 158 | 9 | 269 | 44.87 |
| 5 | 800 | 1,638 | 487 | 107 | 176 | 8 | 291 | 40.24 |
| 5 | 900 | 1,840 | 488 | 110 | 195 | 7 | 312 | 36.06 |
| 5 | 1,000 | 2,048 | 491 | 115 | 212 | 10 | 336 | 31.56 |

**Table 11** Feature based comparison with previously proposed models

| Feature | Kerberos [34] | Pippal et al. [35] | Tao et al. [36] | Proposed Model |
|---|---|---|---|---|
| Checking trust based security parameters | No | No | No | Yes |
| Adequate service registry maintenance | Yes | Yes | Yes | Yes |
| Use of symmetric keys | Yes | Yes | Yes | Yes |
| Checking the access rights | No | Yes | Yes | Yes |
| Handling of bad mouthing attack | No | No | No | Yes |
| Total | 4 | 5 | 4 | 7 |

## 6 Conclusion

This paper proposes a trust based file replication and update propagation model that creates a file replica when the number of request exceeds the threshold value and also maintains file consistency. This threshold is decided, based on the configuration of FRS and application requirements. It discusses the basic trust parameters and adaptive factors in computing trustworthiness of peer FRS, namely, frequency of the request for a particular file that a FRS

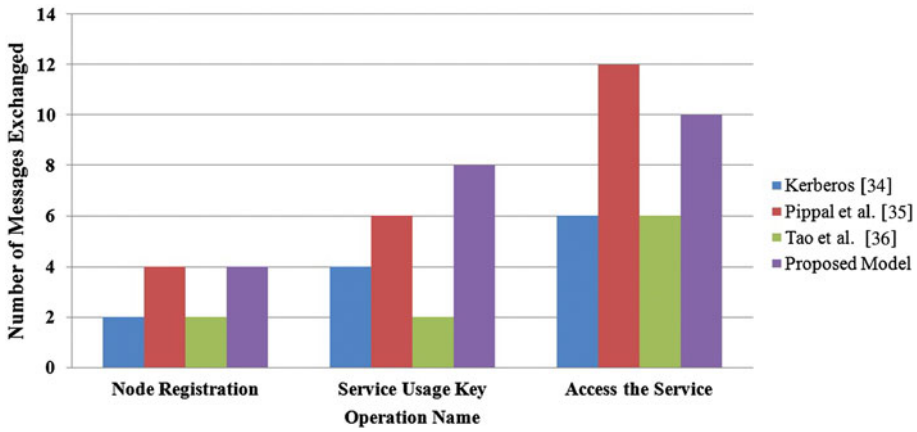



**Fig. 26** *Graph* showing messages exchanged for different models

**Table 12** Average time required for acquiring keys and service (s)

| | Node registration | SUK | Service access time |
|---|---|---|---|
| Proposed model without features | 27.002 | 63.666 | 97.667 |
| Proposed model with features | 30.666 | 77.166 | 114.833 |
| Percentage increase in time | 11.94 % | 17.49 % | 14.94 % |

performs, registration type of node i.e., paid or unpaid, blocking write operation if trust value of a FRS is less than threshold, authenticity of the session key and feedback that a FRS gives about other FRS. The proposed approach is able to resolve many of the unaddressed issues viz., file access frequency, failure handling, avoidance of unnecessary file replication, identification and finally propagation of partial updates. Instead of haphazardly creating the replica, the proposed approach autonomously determines and ascertains the location and need for file replication based on the number of requests and availability of files on FRSs. While performing any file replication operation by a FRS, if this FRS crashes, the proposed model completes the file request via one of the peer FRS thus providing fault tolerance capability to the system. Simulation results show that during high file request scenario for a particular file, frequently accessed files are replicated on other FRSs dynamically and file request is redirected in transparent manner, thus reducing request completion time by about 28.78–47.24 % as compared to FTP. It has also been observed that, when a FRS replicates a file on other FRS, the replication overhead is compensated by various factors like avoiding retransmission of request by the RN, reducing latency in case of load above threshold, ensuring scalability and providing fault tolerance capability. Paper presents a mechanism that reduces time for updating multiple replicas of a file by using modification propagation and transfers the role of master replica to the last modified replica. The master replica computes the file modifications immediately but propagates only the required partial updates on-demand. Experimental results shows that various factors viz., file size, size of modifications and number of replicas to be updated affects the time to propagate the changes to other replicas. Percentage reduction in time for propagating these changes varies from 31.56 to 78.1 %. Simulation results also shows that proposed approach gives a far better performance in terms of time and the benefits are even more if the modification size increases and the replicas to be updated are more in number. Percentage reduction in time for updating replicas varies from 78.17 to 85.37 %. Though the percentage increase in time required for acquiring the keys to access the services with trust based security model varies from 11.94 to 17.49 %, the same is compensated by reduction in time for updating replicas.

## References

1. Resnick P, Zeckhauser R, Friedman E, Kuwabara K (2000) Reputation systems. Commun ACM 43(12):45–48
2. Cohen E, Shenker S (2002) Replication strategies in unstructured peer-to-peer networks. In: ACM SIGCOMM '02 conference, August 2002
3. Clarke I, Sandberg O, Wiley B, Hong T (2000) Freenet: a distributed anonymous information storage and retrieval system. In: Proceedings ICSI workshop on design issues in anonymity and unobservability, Berkeley
4. Wolfson O, Jajodia S, Huang Y (1997) An adaptive data replication algorithm. ACM Trans Database Syst 22(2):255–314
5. Cheng HY, King CT (1999) File replication for enhancing the availability of parallel I/O systems on clusters. In: 1st IEEE Computer Society international workshop on cluster computing, pp 137–144
6. Rao H, Skarra A (1995) A transparent service for synchronized replication across loosely-connected file systems. In: 2nd international workshop on services in distributed and networked environments, 5–6 June 1995, pp 110–117
7. Domenici A, Donno F, Pucciani G, Stockinger H (2006) Relaxed data consistency with CONStanza. In: Sixth IEEE international symposium on cluster computing and the grid, pp 425–429

8. Guy L, Kunszt P, Laure E, Stockinger H, Stockinger K (2002) Replica management in data grids. Technical Report, GGF5 Working Draft, Edinburgh, July 2002
9. Sun Y, Xu Z (2004) Grid replication coherence protocol. In: The 18th international parallel and distributed processing symposium (IPDPS '04)—workshop, Santa Fe, pp 232–239, Apr 2004
10. Huang C, Xu F, Hu X (2006) Massive data oriented replication algorithms for consistency maintenance in data grids. ICCS 2006, Part I, LNCS 3991. Springer, Berlin, pp 838–841
11. Düllmann D, Hoschek W, Martinez JJ, Segal B (2001) Models for replica synchronisation and consistency in a data grid. In: Proceedings of the 10th IEEE international symposium on high performance distributed computing (HPDC-10 '01), Oct 2001, p 67
12. Hu J, Xiao N, Zhao Y, Fu W (2005) An asynchronous replica consistency model in data grid. In: Parallel and distributed processing and applications (ISPA 2005 workshops), pp 475–484
13. Aberer K, Despotovic Z (2001) Managing trust in a peer-to-peer information system. In: Proceedings of ACM conference on information and knowledge management (CIKM)
14. Chen M, Singh JP (2001) Computing and using reputations for Internet ratings. In: Proceedings of third ACM conference on electronic commerce, Tampa
15. Cornelli F, Damiani E, di Vimercati SDC, Paraboschi S, Samarati P (2002) Choosing reputable servents in a P2P network. In: Proceedings of 11th international World Wide Web conference
16. Dellarocas C (2000) Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In: Proceedings of second ACM conference on electronic commerce
17. Kamvar S, Scholsser M, Garcia-Molina H (2003) The EigenTrust algorithm for reputation management in P2P networks. In: Proceedings of 12th international World Wide Web conference
18. Miller NH, Resnick P, Zeckhauser RJ (2002) Eliciting honest feedback in electronic markets. KSG Working Paper Series RWP02-039
19. Sen S, Sajja N (2002) Robustness of reputation-based trust: Boolean case. In: Proceedings of the first international joint conference on autonomous agents and multiagent systems
20. Zacharia G, Maes P (2000) Trust management through reputation mechanisms. Appl Artif Intell 14(8):881–908
21. Buchegger S, Boudec JL (2003) Coping with false accusations in misbehavior reputation systems for mobile ad-hoc networks. EPFL Tech. Rep. IC/2003/31, EPFL-DI-ICA
22. Dellarocas C (2000) Mechanisms for coping with unfair ratings and discriminatory behavior in online reputation reporting systems. In: Proceedings of 21st international conference information systems, Brisbane, Dec 2000
23. Josang A, Ismail R, Boyd C (2005) A survey of trust and reputation systems for online service provision. Decis Support Syst 43(2):618–644
24. Langheinrich M (2003) When trust does not compute—the role of trust in ubiquitous computing. In: Proceedings of 5th international conference on ubiquitous computing, Seattle, Oct 2003
25. Blaze M, Feigenbaum J, Ioannidis J, Keromytis A (1999) The keynote trust-management system. In: RFC 2704
26. Bertino E, Ferrari E, Squicciarini A (2004) Trust negotiations: concepts, systems, and languages. Comput Sci Eng 06(4):27–34
27. Seamons KE, Chan T, Child E, Halcrow M, Hess A, Holt J, Jacobson J, Jarvis R, Patty A, Smith B, Sundelin T, Yu L (2003) TrustBuilder: negotiating trust in dynamic coalitions. In: Proceedings of DARPA information survivability conference and exposition, vol 2(2), pp 49–51
28. Gwertzman J, Seltzer M (1995) The case for geographical push-caching. In: Presented at 5th annual workshop on hot operating systems
29. Hitoshi S, Matsuoka S, Endo T (2009) File clustering based replication algorithm in a grid environment. In: 9th IEEE/ACM international symposium on cluster computing and the grid, pp 204–211
30. Hisgen A, Birrell A, Jerian C, Mann T, Schroeder M, Swart G (1990) Granularity and semantic level of replication in the Echo distributed file system. In: Workshop on the management of replicated data, 8–9 Nov 1990, pp 2–4
31. Hurley RT, Yeap SA (1996) File migration and file replication: a symbiotic relationship. IEEE Trans Parallel Distrib Syst 7(6):578–586
32. Zomaya AY (ed) (1995) Parallel and distributed handbook. McGraw Hill Professionals, New York, pp 60–68
33. Milner R (1989) Communication and concurrency. Prentice Hall, Englewood Cliffs
34. Neuman C, Yu T, Hartman S, Raeburn K (2012) Kerberos RFC 4120. http://www.rfc-editor.org/rfc/rfc4120.txt. Accessed 4 July 2012
35. Pippal SK, Kumari A, Kushwaha DS (2011) CTES based secure approach for authentication and authorization of resource and service in clouds. In: International conference on computer and communication technology (ICCCT), pp 444–449
36. Tao J, Marten H, Kramer D, Karl W (2011) An intuitive framework for accessing computing clouds. Proc Comput Sci 4:2049–2057