ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
DOI: 10.17559/TV-20130507221245

A DYNAMIC FILE REPLICATION BASED ON CPU LOAD AND CONSISTENCY MECHANISM IN A TRUSTED DISTRIBUTED ENVIRONMENT

Manu Vardhan, Dharmender Singh Kushwaha

Original scientific paper
An effort has been made to propose a CPU load based dynamic, cooperative, trust based, and secure file replication approach based along with consistency among file replicas for distributed environment. Simulation results consisting of 100 requesting nodes, three file servers and file size ranging from 677 KB to 11 MB establishes that, when the CPU load is taken into consideration, the average decrease in file request completion time is about 22,04 ÷ 24,81 % thus optimizing the CPU load and minimizing the file request completion time. The CPU load decreases by 4,25 ÷ 5,58 %. Results show that, the average write latency with proposed mechanism decreases by 6,12 % as compared to Spinnaker writes and the average read latency is 3 times better than Cassandra Quorum Read (CQR). The proposed partial update propagation for maintaining file consistency stands to gain up to 69,67 % in terms of time required to update stale replicas. Thus the integrity of files and behaviour of the requesting nodes and file servers is guaranteed within even lesser time. Finally, a relationship between the formal aspects of simple security model and secure reliable CPU load based file replication model is established through process algebra.

***Keywords**: CPU load balancing; distributed systems; file consistency; file encryption; file replication; role based credentials; trust management; update propagation and write invalidate*

Dinamička replikacija datoteke zasnovana na mehanizmu opterećenosti i konzistencije CPU u pouzdanom distribuiranom okruženju

Izvorni znanstveni članak
Pokušalo se predložiti dinamički, kooperativni, pouzdani i sigurni pristup replikaciji datoteke utemeljen na opterećenosti CPU uz konzistenciju među replikama datoteke za distribuirano okruženje. Rezultati simulacije koja se sastoji od 100 potrebnih čvorova, tri servera datoteke i datoteke veličine od 677 KB to 11 MB pokazuju da kada se uzme u obzir opterećenje CPU, prosječno smanjenje vremena potrebnog za popunjavanje datoteke je oko 22,04 ÷ 24,81 %. Tako se optimiziralo opterećenje CPU i smanjilo traženo vrijeme popunjavanja datoteke. Opterećenje CPU smanjuje se za 4,25 ÷ 5,58 %. Rezultati pokazuju da se prosječno kašnjenje upisa (write latency) s predloženim mehanizmom smanjuje za 6,12 % u usporedbi sa Spinnakerovim, a prosječno vrijeme čekanja čitanja (read latency) je 3 puta bolje od Cassandra Quorum Read (CQR). Predložena parcijalna propagacija ažuriranja za održavanje konzistencije datoteke povećava se do 69,67 % u odnosu na vrijeme potrebno za ažuriranje zastarjelih replika. Tako je integritet datoteka i ponašanje zahtijevanih čvorova i servera datoteke zagarantirano za čak manje vremena. Konačno, kroz algebra postupak uspostavljen je odnos između formalnih aspekata jednostavnog modela sigurnosti i sigurnog pouzdanog modela replikacije datoteke zasnovanog na sigurnom pouzdanom opterećenju datoteke.

***Ključne riječi**: balansiranje CPU opterećenja; distribuirani sustavi; kodiranje datoteke; konzistencija datoteke; kvalifikacije utemeljene na ulozi; pouzdano upravljanje; propagacija ažuriranja i poništavanje zapisa; replikacija datoteke*

1 Introduction

To achieve high availability of files in distributed environment, a secure and efficient replication mechanism is required. The communicating nodes in the environment should be trustworthy so as to provide high level of security against various attacks viz., compromised key attack, identity spoofing and masquerading. This should be coupled with least amount of latency and faster response time. For this an efficient CPU load based approach is imperative. All this is needed in order to ascertain the credibility of the participating nodes working together to achieve the goal of Computer Supported Cooperative Working (CSCW).

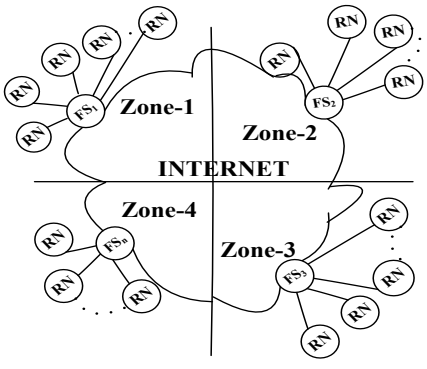

Figure 1 Proposed scenario

Load balancing is one of the important aspects of CSCW. It is achieved by replicating the requested file from the heavily loaded node to lighter one and subsequently redirecting the file request to the lightly loaded node in case any node enters the overloaded region. Along with this, an efficient consistency mechanism should be in place to confirm the integrity of the files. For addressing these issues, paper proposes the scenario as shown in Fig. 1.

Fig. 1 represents two types of nodes viz., File Server (FS) and Requesting Node (RN). It shows a group of File Servers (FSs), along with Requesting Nodes (RNs) that sends the request for a file in a distributed environment. It can be observed from Fig.1 that the connections are scaled on Internet between FSs. FS and RN will communicate/exchange information with each other as and when required. Zones are logically divided depending on the proximity between FSs, based on the addressing scheme (IP address). FS to which a RN is connected is termed as local FS and for this RN all other FS are termed as 'remote' FS.

Properties of FS are as under:

- When a node agrees to share files, it executes FS-server process to assume the role of FS.
- Each FS has a shared directory that contains the files.
- RN requests the file shared by the FS, so as to fulfil its requirement.

- Shared files are replicated based on CPU load from FS_i to FS_J, as and when required.
- When a RN requests a file in write mode, it should commit the changes before timeout period. Timeout period is the time duration in which the RN has to release the lock on the file open in write mode.
- Once the changes are committed, the changes are propagated only to the local FS.
- As soon as the local FS updates its file, it invalidates the replica of that file on other FS, so as to avoid accessing the stale replicas.

Those nodes that agree to share its files with other nodes in the distributed environment are designated as File Server (FS). Initially FS_i connects to FS_J. On successful connection between FS_i and FS_J a message is multicast to all other FS by FS_i. This message contains FS_i IP address and list of files shared by FS_i (where $J \neq i$; $i \leq n$; $J \leq n$; n is the total number of available FS at that time instance). On receiving the multicast messages, all these FS's update its table. Now, all available FS's have the IP address and list of files shared by these FS's. These set of FS are capable of receiving and fulfilling the file requests. But the number of requests a FS can handle is limited by the CPU load. Requesting Node (RN) will send the read/write file request to the local FS. This Local FS, on receiving the file request either fulfils the request locally, or, looks for any remote FS that has the handle of requested file. Now it forwards the IP address of this remote FS to RN. As soon as the connection is established with the remote FS, this FS acts as local FS for RN. All this is carried out ensuring CPU load based file replication, consistency and security issues as proposed in this paper.

To achieve the above mentioned issues for this scenario, the paper aims to increase the availability of files for various nodes in a secure distributed environment. The roadmap to meet this objective is modularized as under:

i A node sends its request for a particular file to a FS.

ii Before CPU load based file replication is done,

- Authenticity of nodes is established based on the trust value, by utilizing the services of Trust Monitor (TM).

iii In order to provide higher level of security, the files are replicated only after encrypting it using Advance Encryption Standard (AES) [26] and finally

iv An efficient partial update consistency mechanism is proposed to maintain the integrity among the replicated files.

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 describes the proposed trust based security mechanism. Section 4 discusses the proposed CPU load based file replication and consistency maintenance mechanism approach in a secured environment and its bisimulation equivalence. Section 5 shows the simulation results followed by conclusion in section 6.

2 Related work

Everyone looks for trusted partners in order to send or receive the data. Popular reputation systems [1] include Eigen Trust [8], Peer Trust [9], Power Trust [10] etc. Eigen Trust is one of the most cited and compared trust models. It assigns each peer a unique global trust value based on the peer's history. However, it introduces the concept of pre-trusted peers, which is very useful in the model, but there is not always a set of peers that can be trusted by default, prior to the establishment of the community. Dou [12] presents a novel recommendation-based trust model. Author identifies that the present trust model could not promise the convergence of iterations for trust computation and model does not consider security problems against Sybil attack and Slandering. Moreover, another assumption of Eigen Trust is that the peers who are honest about the resources they provide are also likely to be honest in reporting their local trust value is arguable. Another work on Eigen Trust proposed by Kamvar et al. [8], focuses on a Gnutella like file sharing network. Shortcoming of the approach is that its implementation is very complex and requires strong coordination and synchronization of peers. Peer Trust [9] is a reputation-based trust supporting framework, which includes a coherent adaptive trust model for quantifying and comparing the trustworthiness of peers based on a transaction-based feedback system. On one hand, it introduces three basic trust parameters and two adaptive factors in computing trustworthiness of peers, namely, feedback a peer receives from other peers, the total number of transactions a peer performs, the credibility of the feedback sources, transaction context factor and the community context factor. On the other hand, it defines a general trust metric to combine these parameters. However, the way it measures the credibility of a peer does not distinguish between the confidence placed on a peer when supplying a service or carrying out a task, and when giving recommendations about other peers. Cuboid Trust [14] is a global reputation-based trust model for peer to peer networks which builds four relations among three trust factors including contribution of the peer to the system, peer's trustworthiness (in reporting feedbacks) and quality of resource. It applies power iteration in order to compute the global trust value of each peer. In this system, direct trust or direct experiences are not given a differentiated treatment, which cannot be well interpreted. In addition, like Eigen trust model, Cuboid trust introduces the concept of pre-trusted peers. It builds several relations among three factors including contribution, trustworthiness and quality of resource to create a more general trust based model. One such trust based system is Gossip Trust proposed by Zhou et al. [15] that enables lightweight aggregation and fast dissemination of global scores. It does not require any secure hashing or fast lookup mechanism. Thus, it is applicable to both unstructured and structured networks. In GroupRep [16], a peer evaluates the credibility of a given peer by its local trust information or the reference from the group it belongs to. An improved computing method to calculate the global trust value is proposed by Fajiang Yu [17]. However, most models do not suit highly dynamic and personalized trust environment. Although

the reputation is operated on limited number of feedbacks rather than aggregating all the ratings, it provides good performance in a variety of situations. There is some recent research on reputation and trust management in distributed systems. Aberer and Despotovic [18] are one of the first in proposing a reputation based management system. However, their trust metric simply summarizes the complaints a peer receives and is very sensitive to the skewed distribution of the community and misbehaving peers. Chen and Singh [19] differentiate the ratings by the reputation of ratters that is computed based on the majority opinions of the rating. Adversaries who submit dishonest feedback can still gain a good reputation as a ratter in their method simply by submitting a large number of feedbacks and becoming the majority opinion. Dellarocas [20] proposes mechanisms to combat two types of cheating behaviour when submitting feedback. The basic idea is to detect and filter out exceptions in certain scenarios using cluster-filtering techniques. This can be applied to feedback-based reputation systems to filter out the suspicious ratings before the aggregation. Sen and Sajja [21] 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. The basic idea is to generate trust values describing the trustworthiness, reliability, or competence of individual nodes, based on some monitoring parameters. Buchegger and Boudec [22] use such trust information for malicious node detection. Josang et al. [23] gives an overview of existing systems that can be used to derive measures of trust and reputation. Langheinrich [24] argues for a renewed evaluation of the benefits from the concept of trust but leaves the calculation of trust assessment up to humans. Keynote is a well-known trust management system proposed by Blaze et al. [25], designed for various large and small-scale Internet-based applications. It provides a single, unified language for both local policies and credentials. For providing higher level of security, Advance Encryption Standard (AES) [26] is used for encryption and decryption file, while replicating the file. AES is a symmetric secret key algorithm used for encryption and decryption of data. The key size is 64-bits. This mechanism derives a 64-bit key value for use by this cipher.

Having discussed the security mechanism for providing the high level of security and once the trust is established between the communicating nodes, some leading proposals of CPU load balancing, replication and consistency mechanism are discussed next.

The issue of load balancing emerges when distributed computing systems and multiprocessing systems began to gain popularity. Baumgartner and Wah [50] and Casavant and Kuhl [2] propose algorithm related to the problem in load balancing in clusters. Lan et al. [3] and Bahi et al. [4] propose distributed load balancing policy, in which every node executes this policy autonomously. Moreover, the load balancing policy can be static or dynamic. In a static load balancing policy, the decisions are predetermined, while in a dynamic load balancing policy, the decisions are made at runtime. Dhakal et al. [5] proposes that a dynamic load balancing policy can be made adaptive to the changes in system parameters, such as the traffic in the channel and the unknown characteristics of the incoming loads. Cortes et al. [6] and Trehel et al. [7] propose that dynamic load balancing can be performed based on either local information (pertaining to neighbouring nodes) or global information, where complete knowledge of the entire distributed system is needed before a load balancing action is executed.

Payli et al. [34] proposes that Dynamic Load Balancing (DLB) provides application level load balancing for parallel jobs using system agents and DLB agent. The approach requires a copy of system agents on all the systems so that DLB agent may collect load information from these systems and perform load balancing. Yagoubi and Slimani [35] puts forward a dynamic tree based model to represent grid architecture and proposes Intra-site, Intracluster and Intra-grid load balancing. Nehraet. al. [36] addresses issues to balance the load by splitting processes into separate jobs and then distributing them to nodes. The authors propose a pool of agents to perform this task. Both approaches modify the dynamic load-balancing step of an adaptive solution.

Tang et al. [31] and Cao et al. [32] address that load balancing plays a critical role in achieving high utilization of resources in Data Grids. Yan et al. [33] proposes a dispatcher and agent based hybrid load balancing policy underlying grid computing environment. The dispatcher performs maintenance, status monitoring, node selection and assignment and adjustment task for each node. The author's consideration of load balancing restricts the system to the ''join and leave'' decision of nodes.

When replication is involved in a distributed file system, there is a need to address many questions. Should the file be replicated on server side only or client side or both? Should we replicate the whole file or a chunk of it? Should we replicate the file content or the file attributes too?

A high-level overview of Network File System (NFS) is presented by Walsh et al [29]. Details of its design and implementation are given by Sandberg et al [30]. Sun NFS uses a TTL (time to live) based approach at the client-side to invalidate replicas. As far as file consistency is concerned, it is not always guaranteed. In case a client modifies a file and subsequently updates this file present on the server, the latest data will still not be available to another client sharing the file until the TTL period is over. The design of NFS involves simplicity and hence they did not take into consideration any of the complex concurrent read/write issues. Dharma et al. [38] propose a data replication algorithm that not only has a provable theoretical performance guarantee, but also can be implemented in a distributed and practical manner. Specifically, authors have designed a polynomial time centralized replication algorithm that reduces the total data file access delay by at least half of that reduced by the optimal replication solution. Google File System [47] introduces an atomic append operation so that multiple clients can append concurrently to a file without extra synchronization between them. GFS has a relaxed consistency model that supports highly distributed applications and remains relatively simple and efficient to implement. File mutations are atomic and are handled exclusively by the master. When an update/mutation

succeeds without interference from concurrent writers (means no overlapping in time), the state is defined as consistent. If interference occurs, then state is undefined i.e. the order is not known but consistent, by maintaining the order of operations on all the replicas. By default GFS creates three replicas. GFS also uses a 2-phase write protocol to achieve consistency among replicas. GFS's consistency is not strict, as it may read from a stale replica before the information is refreshed.

To ensure synchronized file replication across two loosely connected file systems, a transparent service model has been developed by Rao and Skarra [39] that propagates 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 [40] discusses 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. It deals with huge scientific data. Guy [41] proposes a replica modification approach wherein 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 [42] proposes two coherence protocols viz. lazy-copy and aggressive-copy. In lazy-copy protocol, while accessing a modified replica, first the metadata of the modified replica is accessed to get the timestamps of the original and the modified replica. By comparing the timestamps of these two replicas, it is decided if the replica is up-to-date or not. In aggressive copy protocol, no update delay between the original and modified replicas exists. Once the original replica is altered, all other remaining replicas are immediately updated. Dirk et al. [44] and Huang et al. [43] 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 and discusses how they can be included into a replica consistency service. The next section discusses the security mechanism based on node behaviour for distributed environment.

3 Proposed security mechanisms based on node behaviour

For performing secure file replication in distributed environment, a mechanism is required to identify the malicious node activity and for ascertaining the integrity of files.

Reputation systems [18] provide a way for building trust by utilizing community based feedback about past experiences of nodes to help making recommendation and judgment on quality and reliability of the transactions and messages exchanged between communicating nodes. The challenge of building such a reputation based trust mechanism in distributed system is "How to effectively cope with various malicious behaviours 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?" This section proposes Trust Management Service based on the feedback of nodes.

Fig. 2 presents the components of the Trust Management Service. It identifies and elaborates the functionalities and interdependency between the components. Different nodes are denoted as Trust Monitor (TM)/File Server (FS) and Requesting Node (RN). Each trusted FS also assumes the role of TM. Each RN needs to get registered with TM. TM maintains the log of the registered RN. Log consists of <Node_ID, Trust Value, Service Usage Key (SUK)> as discussed below in sub-section 3.2 data structure. Depending on the application requirement, the role of TM can be distributed or centralized. For the given scenario Node 1 2, 3, 4 and 5 are FS, that also performs the role of TM.

TM: Trust Monitor, RN: Requesting Node, FS: File, Server SUK: Service Usage Key, ------: Shows the logical connection between nodes.

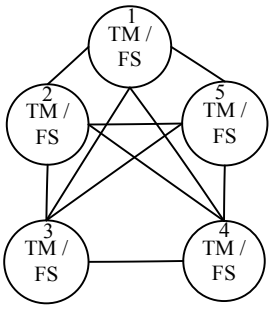

Figure 2 Scenario of trusted file replication model

3.1 Data structure used by TM and FS

Node_ID: shows the IP address of the node (FS and RN) registered with TM and stores TV against each Node_ID. *Trust Value (TV)*: keeps the trust value of a particular node (FS and RN) and also the threshold limit of TV. *Service Usage Key* (*SUK)*: this file identifies the Service Usage Key assigned to a node (FS or RN). *Last file request time*: is the last request time of a file to identify frequent file access behaviour of RN. *Frequent File Count*: for each RN this field furnishes the count of total number of files requested in a specified time span. *Filename*: Name of file. *FileSize*: Size of file. *Request Count*: Number of requests a FS handles depending on the CPU load. *Replication Threshold*: Maximum number of requests a FS can handle, depending on the CPU load, after that file will be replicated on other FS. Once the request gets fulfilled, value in this field is decremented by one. *Valid*: It is a Boolean variable that signifies whether the file is stale or updated. *Lock*: It is an integer variable that signifies that a node has acquired lock on the file and the file is being updated. *Primary FS ID*: It is an integer variable. This specifies the ID of the primary FS (FS that has the latest updated file) of the file. *Last Write Timestamp* (t_{lw}): It is an integer variable. It stores the timestamp at which the particular file is last updated. *Diff*

files: this field is used to store the time stamp (t_{lw}) of Diff files that are created after a replica is modified. *Peers*: It is an array of integer variables and stores the IP address of the FS that has the replica of the file.

Peer FS table is maintained by all FS containing the following fields: *Peer FS ID*: ID of peer File Server. *Peer FS IP*: IP address of peer File Server. *Peer FS Port*: Port address of peer File Server.

3.2 Design of security service mechanism

On receiving the file request from RN, TM checks RN's TV from its TV field of data structure, to ensure that TV(RN)>min(TV). If TV(RN)< min(TV) the request will be discarded. TM authenticates the integrity of SUK (against SUK validity and tampering) i.e. TM matches the SUK received from RN with the SUK present in its SUK field of the data structure. SUK provides a time period within which file access or other operations have to be carried out. This enhances the security and minimizes the risk of security breach by RN, because the SUK is valid only for a limited time period as defined by the TM. If SUK of RN has expired, it will request TM for revocation of SUK. FS provides access to the services (files read and write operation) based on current trust value of the RN and also checks for frequent file request behaviour for the following scenarios:

- If TV of RN, TV(RN)<Threshold(TV), only file read permission shall be granted to RN. File write permission in this case is not allowed to be granted to RN.
- If a RN makes several file requests within a particular time period, file request count will be detected from the <count field and last request time field> of the data structure.
- The request will be fulfilled, if the request count does not exceed the count limit within specified time span. But if the request count exceeds the count limit, the request is rejected. As the behavior of this node is treated as malicious the TV of RN is decreased by 0.1. The specified time span = (current_request_time – last_request_time). TM defines the limit for file request count and the duration of time span. For this local system clock is used.
- FS sends the encrypted file to RN.
- Based on the behavior of RN, its trust value will be updated by the TM.

A nonce is generated by TM on receiving the SUK request from RN. This nonce is known as SUK. SUK is provided by TM to the individual RN only on request. Trust Monitor (TM) keeps the log of the RNs registered with this TM and the same information is maintained by each TM. RN requests Service Usage Key (SUK) from TM to access the service of the FS. TM provides the SUK based on the current TV of the RN. TV of RN should be ≥min(TV). If the TV of RN >min(TV), RN will receive the SUK, else the request will be discarded. Minimum TV is the lowest trust value assigned to RN by the TM. SUK is for specific time period as defined by TM. FS on receiving the file request validates the SUK (against tampering of SUK and its validity). TM matches the SUK received from RN with the SUK present in its SUK field of the data structure. TM also checks for frequent file request behaviour as discussed above. After validating the SUK, TM provides file read or write permission based on the TV of RN as discussed above. TM observes the behavior of RN and updates its trust value. Threshold value of trust lies in between min(TV)<Threshold(TV)<max(TV). Upon subsequent interaction between FS and RN, the TV of RN gradually increases and once threshold limit is reached, the interactions for getting the updated TV of RN nullifies. TV of RN increases or decreases by a multiple of 0.1 as defined by TM. All this is carried out by the following method:

- Node registration with TM.
- RN request for SUK from TM.
- Generation and Distribution of SUK by TM to RN.
- Authentication of SUK by TM, on receiving the request from RN.

Fig. 3 shows two nodes RN, TM/FS and interaction between them. RN sends the registration request to TM and after successful registration RN receives the "ack" message from TM. RN requests for SUK from TM and receives the same. A generic flow and the interaction between different entities (RN, TM/FS) can be observed from Fig. 3.

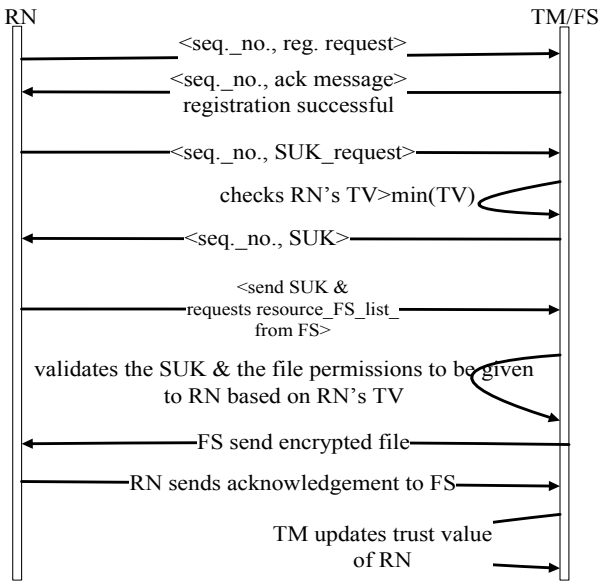

Figure 3 Interaction between RN, FS and TM

3.2.1 Working of TM

TM checks whether the request is made for node registration, granting of Service Usage Key (SUK) or updating Trust Value (TV). RN that gets registered with TM is assigned the min(TV) as defined by TM.

- If RN requests for SUK, TM checks TV(RN), in case TV(RN)<min(TV), SUK is not provided to RN. Accordingly, RN is informed in reply to the request made. Otherwise, if TV(RN)>min(TV), TM send the <SUK> to RN.
- This SUK is assigned by TM to that node RN against their IP address. TM provides SUK to RN on demand.
- TM revokes the SUK of RN if their

TV(RN)>min(TV). Otherwise, the request is discarded.

- In case TV of RN falls below the threshold, TM will update the TV of this RN. If the TV of RN falls below threshold limit, that RN is eligible only for file read permissions and the IP address of that RN is marked by TM to identify these nodes.

3.3 Ensuring file security by using file encryption technique

To enhance the security of file replication mechanism, symmetric key cryptography for encryption and decryption with Advance Encryption Standard (AES) is utilized. AES takes the file as input and creates a cipher of the same length. AES uses a symmetric key which means the same key is used to convert cipher back into the original file. Its block size is of 128 bits. The key size is also of 128bits. Fig. 4 illustrates the communication between FS and RN. On receiving the file request, FS validates the credentials of RN. Once the credentials are validated successfully, FS encrypts the file using AES and transmits it to the RN. RN on receiving the encrypted file decrypts it using the same key as used for encryption. Once the file is successfully decrypted, RN acknowledges the receipt of file to FS.

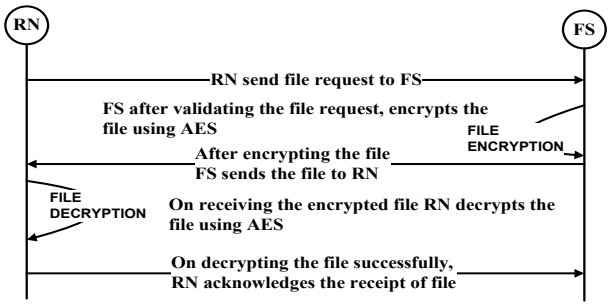

Figure 4 Secure file transfer using AES technique

4 CPU load based file replication and consistency maintenance mechanism

Fig. 5 shows the File Server (FS) and Requesting Nodes (RN). FS is responsible for providing the replication service in the distributed environment. The number of processes a CPU is currently executing decides the load on the CPU i.e. overloaded or average loaded. In case the CPU is overloaded and it keeps fulfilling the request, the file request completion time will increase. But in case the CPU load of the FS is 100 %, FS will start dropping the file request. So, to avoid such situation, a CPU load based file replication mechanism is proposed. Based on the CPU load the requested file is replicated from an overloaded node (FS) to an average loaded node (FS) and the file request is redirected to average loaded node. In Fig. 5 different types of messages labelled as M6, M7, and M8 are elaborated here. M6: updates the load status and other required parameters in the data structure. M7: this message replicates the file and redirects the request from an overloaded node to an average loaded node. M8: carries the file request as sent by the requesting node.

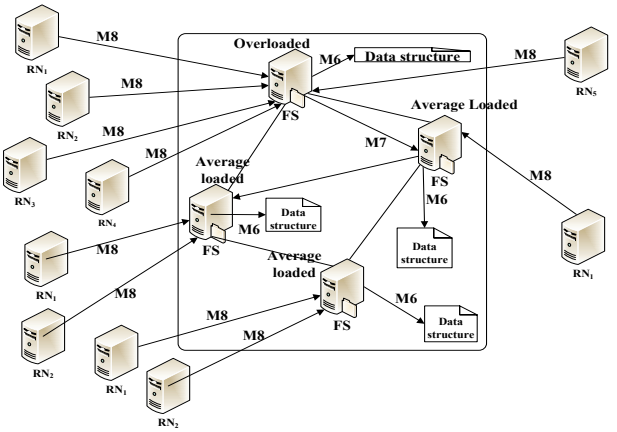

Figure 5 CPU load based file replication mechanism

Fig. 5 shows four File Servers (FS) that are logically connected to each other as scaled on internet. Each FS is assumed as the trusted node. In the proposed File Replication Model as shown in Fig. 5, an average loaded FS can fulfil the file request of the requesting node whereas an overloaded FS looks for an average loaded FS on which the file request can be redirected. Overloaded FS are those on which CPU load is equal to or above 75 % and the CPU load of the average loaded FS is below 75 %. In order to reduce the overhead of polling and broadcasting periodically, FS does not enquire about the load status of other FSs on periodic basis. Instead each FS sends its load status information to other FS when it changes its state from overloaded to average loaded. The algorithm for CPU load based file replication is as follows:

Each FS receives a file request from the Requesting Node (RN) and based on its current CPU load status, handles the request. Requested file is replicated on other FS's when the CPU gets overloaded. The various states of FS are described below:

- Average loaded: File is present on the FS and the CPU load is below 75 %, marked as *ready*.
- Overloaded: File is present on the FS and the CPU load is equal to or above 75 %, marked as *busy*.

The handling of the request takes place as shown in the flow diagram in Fig. 6. It can be observed from the figure if the status of local FS_i is overloaded. In this case, FS_i checks its *peers* field as discussed in data structure section 3.2. Peers field identifies the IP address of only those FS's that have the replica of the requested file. FS_i sends a message to say FS_J. FS_J is one of the peers having the replica of the requested file. This message requests for the status of FS_J. FS_J checks its status against the requested file and replies back to FS_i depending on the following conditions:

- If the status of FS_J is average loaded and requested file is present on FS_J, it will fulfil the request. IP address and port number of this FS_J is sent to the RN. RN connects to this FS_J and receives the file.
- If the status of FS_J is overloaded, FS_i sends a message to FS_k from the *peers* field. This message requests for the status of FS_k. FS_k checks its CPU load status and replies back its status to FS_i (where $J \leq n$; $k \leq n$; n is the total number of available FS at that time instance). Thus, only selected FS from the *peers* field,

in an ordered way, will be requested for their status against the requested file.

- As soon as FS_i finds a peer FS (FS_J or FS_k) with its status as Average loaded, IP address of that peer FS (FS_J or FS_k) is sent to the RN by FS_i and the RN connects to that peer FS (FS_J or FS_k) and receives the file. And no more request messages for CPU load status will be sent to peer FSs by FS_i.
- *Peer* field identifies the IP address of only those FS's that have the replica of the requested file. If those FS's present in the *peer* field that has the replica of the requested file are overloaded, and the remaining FS's do not contain the replica of the requested file, in this case, FS_i replicates the file on FS_J that has the status as Average loaded. IP address of this peer FS_J is sent to the RN and RN connects to this FS_J and receives the file. Thus, the overhead of broadcasting the status request message is avoided. In case the number of FS's is more, the proposed replication approach significantly reduces the number of messages exchanged.

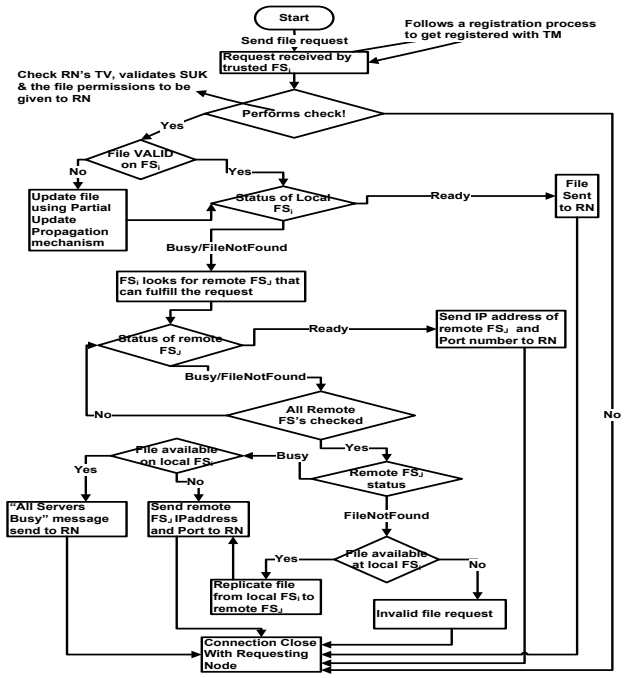

Figure 6 Flow diagram for CPU load based file replication

The functioning of File Server (FS) under various scenarios is discussed in the next section.

4.1 Replication Scenarios

The various scenarios presented in this section explain the complete File Replication model. The scenarios described below involve three FS's viz., S_1, S_2, S_3 and one Requesting Node (RN) N_1. The messages exchanged during the communication between FSs and RN are described as follows: M_1: This is a request message that consists of either resource_FS_list message or file request or replication request or status of other FS. The resource_FS_list message is the request message for the list of file names, FS IP address and FS Port number from the Local FS. M_2: This is the status message of FS. The two statuses are Average loaded, and overloaded. M_3: This message denotes the sending of the file contents to the RN or FS, or the sending of the IP address, Port address of remote FSs and the resource_FS_list present on the local FS to RN. M_4: This message involves the IP and Port address of the remote FS from which the requesting node establishes the connection to receive the replicated file. M_5: Reply acknowledgement (RACK) from FS_J to FS_i is sent, after the file has been replicated successfully on FS_J.

4.1.1 Case 1: Local FS S_1 cannot fulfil the request and looks for a remote FS S_2 that can fulfil the file request

Requesting Node (RN) N_1 requests SUK from TM and once the SUK is received by RN, it sends resource_FS_list a request (message M_1) to FS(S_1). TM ensures that TV(RN)>min(TV). After ensuring the TV of RN and validating the SUK, FS(S_1) sends the resource_FS_list (message M_3) to N_1. N_1 sends file request (M_1) to the S_1. S_1 checks the file availability on FSs, file validity on S_1 (i.e. locally) and S_1 status based on CPU load. S_1 observed that the request cannot be fulfilled locally because S_1 status is overloaded. Now, S_1 sends the status request message (M_1) to remote FS(S_2). S_2 replies its status as *Average loaded* (M_2) to S_1. S_1 sends IP and Port address of S_2 (message M_4) to N_1. N_1 receives the file in encrypted form from S_2. After the communication gets over, both S_1 update the trust value of RN.

4.1.2 Case 2: Local FS S_1 replicates the file on remote FS S_3

As discussed earlier, Requesting Node (RN) N_1 requests SUK from TM and once the SUK is received by RN, it sends resource_FS_list request (message M_1) to FS(S_1). TM ensures that TV(RN)>min(TV). After ensuring the TV of RN and validating the SUK, FS(S_1) sends the resource_FS_list (message M_3) to N_1. N_1 sends file request (message M_1) to S_1. S_1 checks the file availability on FSs, file validity on S_1 (i.e. locally) and S_1 status based on the CPU load. S_1 observed that the request cannot be fulfilled locally because S_1 status is *overloaded*. The status of S_1 is *overloaded*, so, FS(S_1) sends the status request message (M_1) to remote FS(S_2). S_2 replies its status as *overloaded* (M_2) to S_1. After the communication gets over, both S_1 and S_2 update the trust value of each other on TM. Now, S_1 sends the status request message (M_1) to remote FS(S_3). S_3 replies to S_1, its status as Average loaded and also informs that the requested file is not present (M_2) on S_3. S_1 sends the replication request message (M_1) to S_3. S_1 encrypts the requested file and creates the replica of requested file (message M_3) on the S_3. Once the file is successfully decrypted, S_3 sends RACK message (M_5) to S_1. After the communication gets over both S_1 and S_3 update the TV of each other to TM. S_1 sent the IP address and Port number of the S_3 (message 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 FS, those changes need to be propagated to those FS on which the replica is present. For this a partial update propagation and write invalidate mechanism for maintaining file consistency is proposed in the next section.

4.2 Proposed partial update propagation mechanism for maintaining replica consistency

It is assumed that the clocks of all FS's are synchronized with each other and all RN's synchronize their clocks with local FS. A partial update propagation and write invalidate mechanism is proposed. Most of the existing approaches propose that every file has a primary replica and other replicas are considered to be secondary. This primary replica is called the master replica [39]. In most of the existing approaches, if a secondary replica of the file on node N_x is modified, the master replica on node N_y has to be updated immediately. With this approach there is need to wait until file write operation on secondary replica on node N_x gets completed. After the secondary replica has been updated on node N_x, this updated replica on node N_x, needs to be propagated from node N_x to the master replica on node N_y. But, with the proposed approach, FS that has last modified the file replica will become the primary FS for that file. FS maintains the following entries in data structure that keep track of information like file name, file's last modification time (t_{lw}), IP address of the FS that has latest valid file and *Diff file/s* created at different time stamps <File_Name (f_i), Last Write Time Stamp (t_{lw}), Primary FS ID, *Diff File* ($D(f_i(t_{lw}))$) > i.e. f_1, FS_x, $f_1(t_{lw})$, $D(f_1(t_1))$ $D(f_1(t_2))$ …$D(f_1(t_n))$. As soon as the FS_i gets request for write operation on file f_1, FS_i checks whether file f_1 is VALID or INVALID.

If file f_1 is valid on FS_i, it acquires the lock on file f_1. Now, FS_i identifies those remote FS_J that have the replica of file f_1, from its Peers field of the data structure as discussed in section 3.2. FS_i sends a message only to these remote FS_J, that the new primary FS of file f_1 is FS_i. On receiving this message, FS_J invalidates its replica f_1 and makes an entry in the Primary FS ID field of the data structure that the primary FS of file (f_1) is FS_i, on which last write operation has been done. So there is no need to update any other replica immediately. But if the file is invalid, it is updated using Partial Update Propagation.

Partial Update Propagation: If file f_1 is invalid, FS_i checks the Primary FS ID field of the data structure as discussed in section 3.2. This field gives the IP address of the FS that has the latest replica of file f_1. FS_i sends a request message to primary FS_J of file f_1. FS_i requests for the updates of file f_1, done after time stamp t_{lw} i.e. FS_i $[f_1(t_{lw})]$. FS_J on receiving the request message for updates from FS_i, FS_J checks the Time Stamp of file f_1 i.e. FS_J $[f_1(t_{lw})]$. If the Time Stamp (t_{lw}) of file f_1 on FS_J is subsequent to the Time Stamp (t_{lw}) of file f_1 on FS_i, in this case FS_J will send only those *Diff file/s* i.e. $D(f_1(t_{lw}))$, that are created after the Time Stamp of file f_1 on FS_i i.e. FS_i $[f_1(t_{lw})]$. After receiving the *Diff file/s* from FS_J, FS_i performs the join operation ($\sum$) to update its stale file replica. Before applying the join operation on file f_1, FS_i ensures that file f_1 is not locked by any RN_i associated with FS_i. After applying the join operation, file f_1 turns into an updated one. Now, FS_i has the valid file f_1.

To validate the proposed model, Calculus of Communicating System (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.

4.3 Bisimulation equivalence of secure CPU load based file replication and consistency mechanism

Stability analysis of Secure File Replication and Consistency Mechanism, using a process algebraic approach is carried out in this section. Transition systems [49] 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 observe. Meaning of the symbols used in the CCS [46] 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. *FS*: Stands for File Server. This denotes the Server Node of the R model. *RN*: Stands for Requesting Node. This denotes the Client Node for the proposed replication model. *RI*: This is the set of internal actions for the proposed replication model. The symbol in CCS (') denotes the action of sending message and the rest of the actions denote the inputs/receiving message.

4.3.1 Definition of Simple Provider and Requesting Node

Definition of Simple Provider Node (SPN): Provides the file to the requesting node, without performing any file replication and changes its state back to initial state i.e. SPN_i. SPN in state SPN_i on receiving the file request message (requestFile) from SRN, changes its state from SPN_i to state SPN_1. In state SPN_1, after acknowledging the existence of file ('fileExists), SPN changes its state from SPN_1 to SPN_2. SPN in the state SPN_2, sends its status ('fsStatusAverageloaded) to SRN and switches to state SPN_3. Finally, after sending the file ('fileContent) to SRN, SPN switches its state from SPN_3 to initial state i.e. SPN_i.

$$SPN \stackrel{def}{=} requestFile.'fileExists.'fsStatusAverageloaded.'fileContent.SPN \quad (1)$$

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

$$SRN \stackrel{def}{=} 'requestFile.fileExists.fsStatusAverageloaded.fileContent.SRN \quad (2)$$

Setting internals for simple module

$$SI \stackrel{def}{=} \{fileExists\} \quad (3)$$

Model for Simple Server with No Replication (NR)

$$NR \stackrel{def}{=} (SPN \mid SRN) \setminus SI \quad (4)$$

4.3.2 Definition of File Server (FS) and Requesting Node (RN)

Definition of File Server (FS): FS fulfils the file requests received from RN, performs the file replication from FS_i to FS_J and changes its state back to initial state i.e. FS_i. FS in initial state i.e. FS_i on receiving the file request (requestFile) changes its state to FS_1. After acknowledging the existence of file ('fileExists) FS now changes its state from $FS_1$to FS_2. FS from state FS_2 can change its state either to FS_3 or FS_4. In case FS switches from state FS_2 to state FS_3 (i) FS in state FS_2 sends its status as Average loaded ('fsStatusAverageloaded) to the RN and switches its state from FS_2 to FS_3. In this state (FS_3) FS sent the encrypted file content ('AESencFileContent) to the RN. After successfully transmitting the file to RN, FS changes its state from FS_3 to initial state FS_i. OR If FS switches from state FS_2 to state FS_4 (ii) FS in state FS_2 sends a request message to remote FS_J for their status ('fsStatus) and FS switches its state from FS_2 to FS_4. After receiving the status from remote FS_J as Average loaded and file not present on FS_J, FS changes its state from FS_4 to state FS_5. FS in this state i.e. FS_5 sends a replication request ('put) to remote FS_J and changes its state from FS_5 to state FS_6. In this state (FS_6) FS replicates the encrypted file ('AESencFileContent) on remote FS_J. After successfully replicating the file from FS_i to remote FS_J, FS reaches state FS_7. Now, FS in this state (FS_7) sends the IP address and port number of remote FS_J to the RN and changes its state from FS_7 to initial state i.e. FS_i.

FS $\stackrel{def}{=}$ fsStatus.'no.FS + put.AESencFileContent.FS + requestFile.('fileExists.('fsStatusAverageloaded.'AESencFileContent.FS + 'fsStatus.no.'put.'AESencFileContent.'newfs.FS)) (5)

Definition of Requesting Node (RN): requests a file from FS and changes its state back to initial state i.e. RN.

RN $\stackrel{def}{=}$ 'requestFile.(fileExists.(fsStatusAverageloaded. AESencFileContent.RN + newfs.RN)) (6)

Setting internals for replicating module

RI $\stackrel{def}{=}$ { fsStatus, put, no, newfs, fileExists } (7)

Definition of replicating module

R $\stackrel{def}{=}$ (FS | RN) \ RI (8)

Above mentioned CCS is compiled on CWB-NC and bisimulation equivalence is proved between dynamic File Replication model (R) and simple server with no replication (NR) model i.e. R ≈ NR. The output of the CWB-NC compiler is shown below:

cwb-nc> load FS.ccs
Execution time
(user,system,gc,real):(0.000,0.000,0.000,0.002)
cwb-nc>eq -S bisim R NR
Building automaton...
States: 34
Transitions: 62
Done building automaton.
TRUE
Execution time
(user,system,gc,real):(0.004,0.000,0.000,0.005)

This output shows the bisimulation equivalence of the proposed Replicating (R) model with the standard non-replicating (NR) model.

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

5 Simulation and results

5.1 CPU load based file replication mechanism

The simulation has been conducted for CPU load based File replication, using one, two, and three FSs. The simulation is carried out with 100 RN's and each RN requests for file F of size 677KB; 3,1 MB or 11 MB from FS_i. The proposed model is simulated on Linux platform with the network transfer speed of 300 kb/s.

The comparison in terms of request completion time for varying file size using one, two, and three FS is shown in Figs.7, 9, and 11. When CPU load based file replication mechanism is devoid of any security mechanism, average completion time for a request is always less than the average completion time with trust and security. Initially when the files are not available (replicated) on other FS's, the time required to fulfil the request of RN is higher. After sufficient replicas are created, the service time for each request decreases significantly. When any FS_i receives file request for file *fi* and this request moves the CPU load to 75 %, it replicates the file on FS_J. This replication overhead is compensated by the benefits like avoiding re-sending of request (in case the FS is not able to service the request, it forwards to other available FS).

5.1.1 One file server

A scenario with 100 requesting nodes and only one FS is shown in Fig. 7. It shows the request completion time taken by one FS for varying file size viz., 677 KB; 3,1 MB and 11 MB. It can be observed from the figure that, with 1-FS the file request completion time increases as the number of requesting nodes increases. This is due to the reason that the system keeps fulfilling the request even when the CPU is 95 % loaded. For file size of 677 KB, the request completion time of requesting node 1÷60 is 562,63 ms and for requesting node 61÷100 it is 645,975 ms, i.e. increase in request completion time by 14,81% and the corresponding CPU load increases from 4,94 units to 4,97 units i.e. by 0,51 %. For file size of 3,1 MB, the request completion time increases from 1338,55ms to 1539 ms, i.e. by 14,97 % and the corresponding CPU load increases from 4,94 units to 5,01 units i.e. by 1,34 %. For file size of 11 MB, the request completion time increases from 3775,91 ms and to 4337,7ms, i.e. by 14,87 % and the corresponding CPU load increases from 4,56 units to 4,97 units i.e. by 8,90 %. The average increase in request completion time using 1FS is 14,88 % and the average increase in CPU load is by 3,58 %. The request completion time decreases for requesting node 82÷100, because the load on FS decreases as most of the requests

are completed and this is in accordance with the CPU load of FS.

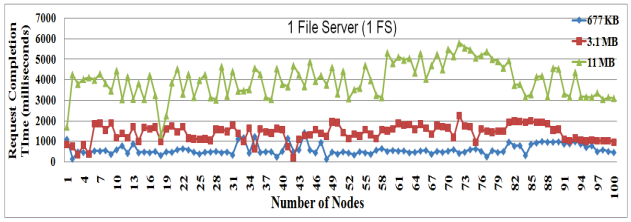

Figure 7 Request completion time based on CPU load using 1-FS

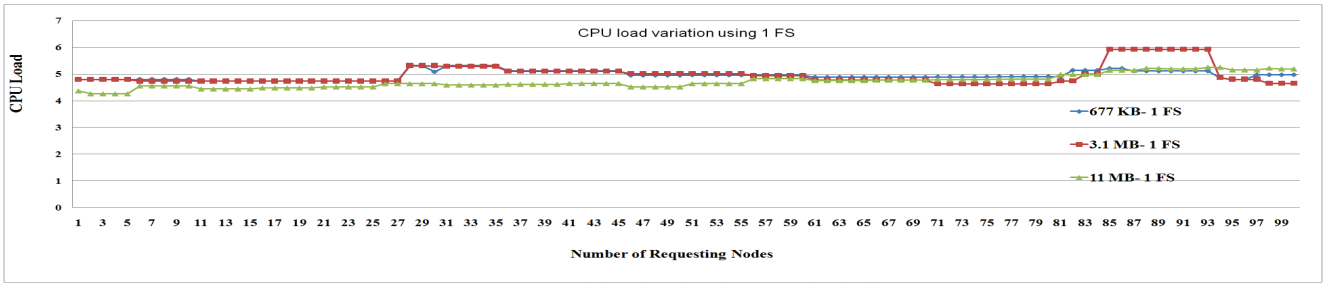

Figure 8 Request completion time based on CPU load using 1-FS

5.1.2 Two file servers

A scenario with 100 requesting nodes and two FSs is shown in Fig. 9.

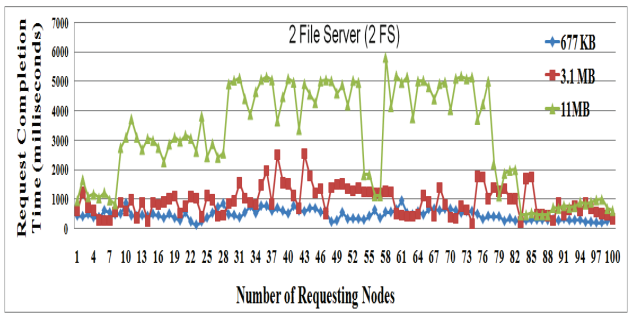

Figure 9 Request completion time based on CPU load using2-FS

It shows the request completion time in seconds for 2FS's. With two FS, the file request can be fulfilled from two servers at different locations (FS_1, and FS_2). In case of 2FS, when the CPU load is greater than or equal to 75 %, the requested file is replicated from FS_1 to FS_2. Now the request is fulfilled from both the FS, i.e. FS_1, & FS_2. In case CPU load of both the FS is greater than or equal to 75 %, the request will be dropped until the CPU load is less than 75 %. For the file size of 677kB, the request completion time for requesting node 1 ÷ 60 is 485,46ms and for requesting node 61 ÷ 100 is 396,95ms, i.e. request completion time decreases by 18,23 %, because the corresponding CPU load decreases from 2,68 units to 2,59 units i.e. by 3,13 %. For the file size of 3,1MB, the request completion time decreases from 1025,03ms to 785,8ms, i.e. by 23,33 %, because the corresponding CPU load decreases from 2,25 units to 2,16 units i.e. by 3,91 %. For the file size of 11MB, the request completion time decreases from 3430,78 ms to 2550,92 ms, i.e. by 25,64 %, because the corresponding CPU load decreases from 2,70 units to 2,55 units i.e. by 5,70 %. The average decrease in request completion time using 2FS is 22,04 % and the average decrease in CPU load is by 4,25 %.

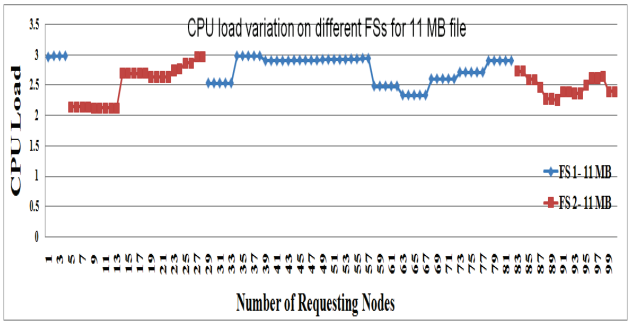

Figure 10 CPU load variation on FS-1 and FS-2 for 11 MB file

5.1.3 Three File Servers

A scenario with 100 requesting nodes and three FSs is shown in Fig. 11. It shows the request completion time in seconds for 3FS's. With three FS, the file request can be fulfilled from any of the three locations (FS_1, FS_2 or FS_3). In case of 3FS, when the CPU load is greater than or equal to 75 %, the requested file is replicated from FS_1 to FS_2 or FS_1 to FS_3, depending on the CPU load status of FS_2 and FS_3. In case both FS_2 and FS_3 are average loaded, FS is selected in an ordered way. Now the request is fulfilled from all the FS, i.e. FS_1, FS_2 & FS_3. In case CPU load of all the FS is greater than or equal to 75 %, the request will be dropped until the CPU load is less than 75 %. For the file size of 677kB, the request completion time for requesting node 1 ÷ 60 is 570,6 ms and for requesting node 61 ÷ 100 is 450,67 ms, i.e. request completion time decreases by 21,01 %, because the corresponding CPU load decreases from 1,66 units to 1,60 units i.e. by 3,44 %. For the file size of 3,1 MB, the request completion time decreases from 869,21 ms to 650,82 ms, i.e. by 25,12 %, because the corresponding CPU load decreases from 1,96 units to 1,82 units i.e. by 6,89 %. For the file size of 11 MB, the request completion time decreases from 2925,05 ms to 2097,6ms, i.e. by 28,28 %, because the corresponding CPU load decreases from 1,97 units to 1,83 units i.e. by 7,19 %. The average decrease in request completion time using 3FS is 24,81 % and the average decrease in CPU load is by 5,58 %.

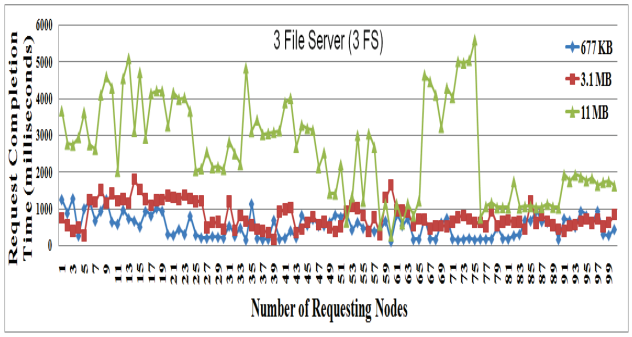

Figure 11 Request completion time based on CPU load using 3-FS

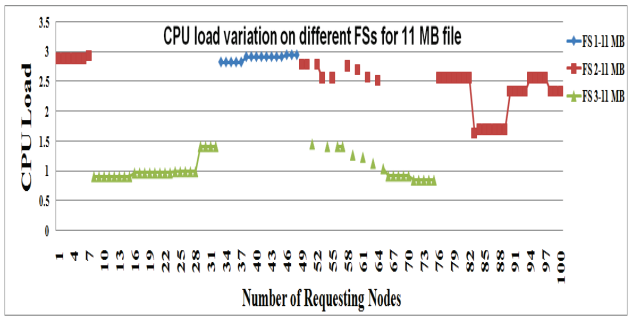

Figure12 CPU load variation on FS-1, FS-2 and FS-3 for 11 MB file

Table 1 Average request completion time based on CPU load (ms)

Number of File Serve (FS)	Requesting Node (RN)	677 KB	3,1 MB	11 MB
1FS	1 - 20	526,2	1307	3496,7
	21 - 40	582,65	1375,75	3788,15
	41 - 60	579,05	1332,9	4042,9
	61 - 80	502,85	1623,05	5047,2
	81 - 100	789,1	1454,95	3628,2
2FS	1 - 20	449,8	692,25	2212,9
	21 - 40	521,5	1107,1	3979,65
	41 - 60	485,1	1275,75	4099,8
	61 - 80	529,4	863,2	4290,9
	81 - 100	264,5	708,4	810,95
3FS	1 - 20	836,05	1091,35	3605,15
	21 - 40	351,75	774,9	2992,8
	41 - 60	524	741,4	2177,2
	61 - 80	341,1	669,5	2755,6
	81 - 100	560,25	632,15	1439,6

Tab. 1 shows the average request completion time for various scenarios.

5.1.4 Partial update propagation

For a file of size 677 kb, Fig. 13 shows the comparison between the proposed partial update consistency mechanism and the write update mechanism. With the proposed partial update consistency mechanism, the average time required for updating the stale replicas decreases from 554,35 ms to 165,7 ms. The average decrease in time for updating the stale replicas using partial updates is 69,67 %.

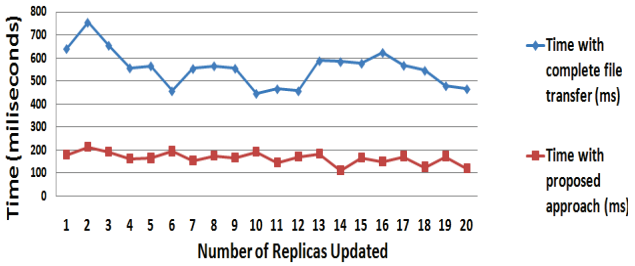

Figure 13 Partial Update Consistency Mechanisms

5.2 Comparison with GFS, Spinnaker and Cassandra

Load balancing can be used to distribute incoming requests to two or more instances of an application, dividing the work load between the instances. In GFS [47], the load balancer is a software or hardware application that distributes the requests of different types to the appropriate applications. Spinnaker [27] is a consistent and highly available data store that is designed to run on a large cluster of commodity servers in a single data centre. Spinnaker is derived from Cassandra [37] codebase that is eventually a consistent data store.

The graphs of our results show the average latency of a read or write operation (on the Y axis) for a given system "load" (on the X axis). System load is the average number of read or write requests per second generated by a requesting node. Results are shown for the scenario of 100 requesting nodes and two file servers.

Table 2 System Configuration

	Spinnaker and Cassandra	Proposed
Processsor	Two Quad-core 2,1 GHz AMD	3,6 GHz P IV
Memory	16 GB	1 GB
Hard Disk	5 SATA disks	80 GB
Ethernet Connection	1 Gb/s	100 Mb/s
Switch	1 Gb/s	100 Mb/s
Bandwidth	1 Gb/s	300 Kb/s

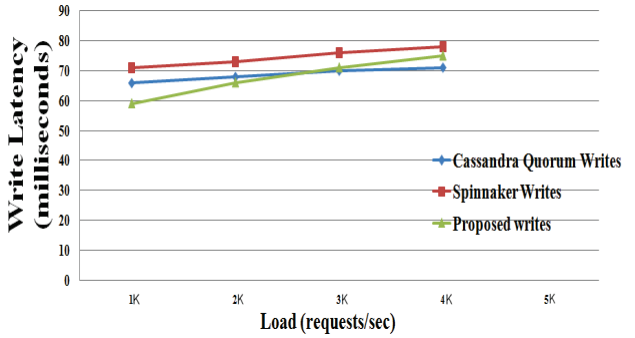

Figure 14 Average write latency

Fig. 14 shows the average latency of a write as the load increases. It is observed that, the average write latency with proposed mechanism decreases by 6,12 % as compared with Spinnaker writes, because the file request can be fulfilled from any of the two file servers. But this latency increases by 2,06 % as compared to Cassandra quorum write, because sometimes, the file server gets overloaded which increases the write latency of proposed write mechanism and also due to lower system configuration and bandwidth as discussed in Tab. 2.

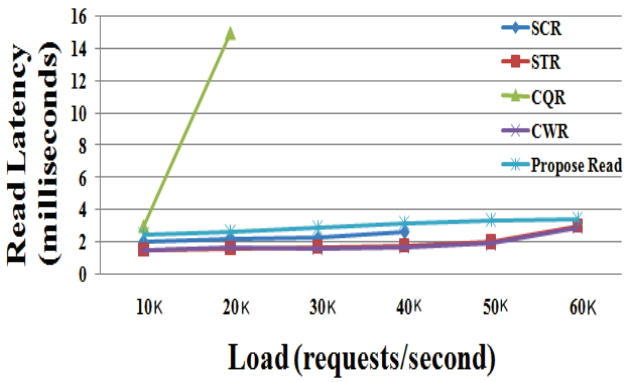

Figure 15 Average read latency

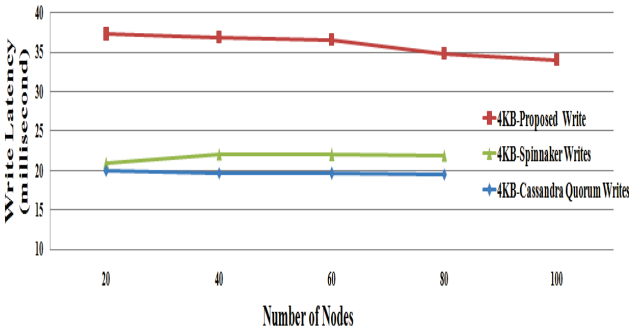

Figure 16 Average write latency with increasing number of nodes

Table 3 Comparison of load balancing approaches

Parameters	GFS	Spinnaker	Cassandra	Propose Mechanism
Load Balancer Type	software and hardware	Software	Software	Software
Levels of Load Balancing	Two	One	One	One
Dedicated Load Balancer	Yes	No	No	No
Load Balancing Technique	Use Sticky Session	key-based range partitioning	key-based range partitioning	Based on CPU load
Connection type	Physical	Logical	Logical	Logical

Fig. 15 shows the average latency of a read as the load increases. It shows the latency of Spinnaker and Cassandra for 4KB read against the proposed scheme for read across the board. It is observed from the table that the average read latency with the proposed mechanism is 3times better than Cassandra Quorum Read (CQR). This is because a quorum read in Cassandra has to access two replicas and check for conflicts, whereas a read with the proposed mechanism has to access the replica from any of the two file servers. The average read latency increases by 32,08 % as compared to Spinnaker Consistent Reads (SCR), Spinnaker Timeline Reads (STR) and Cassandra Weak Reads (CWR), because the consistent read in spinnaker only has to access the leader replica and also due to the system configuration as discussed in the Tab.2.

Fig. 16 shows that the average write latency for 4 KB Spinnaker writes is 21,68 ms, Cassandra Quorum writes is 19,7 ms and for the proposed write mechanism the average write latency is 35,86 ms. Fig. 16 shows that for both Spinnaker and Cassandra, the write latency remained roughly constant with increasing number of nodes. Whereas in the proposed approach, it is done on all the nodes. This is because a write is performed only on three nodes, regardless of the number of nodes. As compared to the proposed write mechanism, when the number of requesting nodes increases by 20 times, the average write latency increases by 1,67 ÷ 1,84 times. This shows that write latency does not increase proportionally with respect to the increasing number of nodes.

5.3 How the proposed approach is robust: A Comparison of Load Balancing Approaches

- Spinnaker and Cassandra perform write operation only at three nodes, whereas the proposed mechanism writes the file on-demand to n number of nodes.
- In Google's Bigtable when a node goes down, all the data on that node becomes unavailable until the node is restarted and its log in GFS is replayed. But with the proposed replication mechanism the data can be accessed from another file server, on which the replica is present.
- In the proposed mechanism all read and writes are carried out in a secure manner based on the trust of the requesting nodes and Advance Encryption Scheme (AES) is used while sending the file over the channel.

6 Conclusion

An optimal CPU load based approach for a trusted, distributed and dynamic file replication mechanism is proposed. An incoming request received by a file server is either serviced based on its own CPU load or redirected to the file server whose CPU is average loaded. Thus the proposed dynamic CPU load based file replication mechanism adapts to the changing CPU load. We have shown experimentally that the proposed CPU load based file replication mechanism minimizes the average file request completion time by replicating the requested file on an average loaded file server and subsequently redirecting the file request to this file server. Thus improves the system utilization rate. All this is achieved even after trust maintenance and security overhead. Basic trust parameters and adaptive factors in computing trustworthiness of peers based on Trust Value (TV) of RN, frequency of the requesting a file by RN and integrity of the SUK (service usage key) is proposed. Trust Monitor (TM) gauges the TV of requesting node based on its activities to be used by FS. To accomplish this objective, there was a need to address issues like:

- Ascertaining trustworthiness of RN.
- Establishing secure communication among various parties.
- Secure file replication from FS_i to FS_J or RN in a CPU load based trusted distributed environment and
- Finally, an efficient consistency mechanism that reaffirms the integrity of the files.

Once the trust has been established between communicating nodes i.e. FS and RN, file replication is carried out in a secure manner using AES. Initially, when

the file is present only on one file server, CPU gets overloaded which may lead to dropping of file request by the file server and subsequently increases the file request completion time. Later when the file gets replicated on most of the FS‘s, the average file request completion time decreases and the overhead of security gets negligible. In particular, when the CPU load is taken into consideration, the average decrease in the file request completion time achieved is about 22,04 ÷ 24,81 %, thus optimizing the CPU load and minimizing the file request completion time. The CPU load itself decreases by 4,25 ÷ 5,58 % and the overhead of trust maintenance and security is significantly minimized. This is attributable to the fact that the CPU load based file replication mechanism achieves a better spread of requests, and reduces the likelihood of FS’s being idle during peak traffic scenario. Results show that the average write latency with proposed mechanism decreases by 6,12 % as compared to Spinnaker writes and the average read latency is 3 times better than Cassandra Quorum Read (CQR).

The proposed partial update propagation for maintaining file consistency stands to gain up to 69,67 % in terms of time required to update stale replicas. Finally, a relationship between the formal aspects of the simple security model and secure reliable file replication model is established through process algebra. The stability and reliability analysis ensures that the system will run in the finite sequence of interaction and transitions. On the basis of these properties, we have been able to build a secure and reliable file replication model. This work is one of the few that attempt to investigate the file access time with security implications and carefully design a file replication model that absorbs the overhead of security measures while replicating a file.

7 References

[1] Resnick, P. et al. Reputation systems. // Commun. ACM. 43, (2000), pp. 45-48. DOI: 10.1145/355112.355122

[2] Casavant, T. L.; Kuhl, J. G. A Taxonomy of Scheduling in General Purpose Distributed Computing Systems. // In T.L. Casavantand M. Singhal, ed., Readings In Distributed Computing Systems, IEEE Computer Society Press, 1994.

[3] Lan, Z.; Taylor, V. E.; Bryan, G. Dynamic Load Balancing for Adaptive Mesh Refinement Application. // Proc. Int'l Conf. Parallel Processing (ICPP), 2001.

[4] Bahi, J. M.;Contassot-Vivier, C.; Couturier,R. Dynamic Load Balancing and Efficient Load Estimators for Asynchronous Iterative Algorithms. // IEEE Trans. Parallel and Distributed Systems, 16, 4(2005), pp.289-299. DOI: 10.1109/TPDS.2005.45

[5] Dhakal, S.; Hayat, M. M.; Pezoa, J. E.; Yang, C.; Bader,D A. Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration-Theory Approach. // IEEE Trans. Parallel Distrib. Syst. 18, 4(2007), pp. 485-497. DOI: 10.1109/TPDS.2007.1009

[6] Cortes, A.; Ripoll, A.; Senar, M.; Luque, E. Performance Comparison of Dynamic Load-Balancing Strategies for Distributed Computing. // Proc. 32nd Hawaii Conf. System Sciences, 8, (1999), p. 8041. DOI: 10.1109/hicss.1999.773073

[7] Trehel, M.; Balayer, C.; Alloui, A. Modeling Load Balancing Inside Groups Using Queuing Theory. // Proc. 10th Int'l Conf. Parallel and Distributed Computing System, Oct. 1997.

[8] Kamvar, S.; Schlosser, M.; Garcia-Molina, H. The Eigen Trust algorithm for reputation management in p2p networks. // Proc. ACM World Wide Web Conf. (WWW '03), Budapest, Hungary, May 2003, pp. 640-651.

[9] Wang, W.; Zeng, G.; Yuan, L. Ant-based reputation evidence distribution in P2P networks. // GCC, pp. 129-132. Fifth International Conference on Grid and Cooperative Computing, IEEE Computer Society, Changsha, Hunan, China (2006).

[10] Zhou, R.; Hwang, K. PowerTrust: a robust and scalable reputation system for trusted peer-to-peer computing. // IEEE Trans. Parallel Distrib. Syst., 18, 4(2007), pp. 460-473. DOI: 10.1109/TPDS.2007.1021

[11] Bruhadeshwar, Bezawada; Kulkarni, S. S.; Liu, A. X. Symmetric Key Approaches to Securing BGP—A Little Bit Trust Is Enough. // In IEEE Transactions on Parallel and Distributed Systems. 22, 9(2011), pp. 1536-1549. DOI: 10.1109/TPDS.2011.19

[12] Dou, W.; Wang, H.-M.; Jia, Y.; Zou, P. A recommendation-based peer to peer trust model. // J. Softw. 15, 4(2004), pp. 571-583.

[13] Kyoung-Don, Kang; Can, Basaran. Adaptive Data Replication for Load Sharing in a Sensor Data Center. // Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW '09). IEEE Computer Society, Washington, DC, USA, 20-25.

[14] Chen, R.; Chao, X.; Tang, L.; Hu, J.; Chen, Z. CuboidTrust: a global reputation-based trust model in peer-to-peer networks. // Fourth Int. Conf. on Autonomic and Trusted Computing, ATC 2007, (LNCS, 4610), 2007, pp. 203-215.

[15] Zhou, R.; Hwang, K.; Cai, M. GossipTrust for fast reputation aggregation in peer-to-peer networks. // IEEE Trans. Knowl. Data Eng. 20, 9(2008), pp. 1282-1295. DOI: 10.1109/TKDE.2008.48

[16] Tian, H.; Zou, S.; Wang, W.; Cheng, S A group based reputation system for P2P networks. // Third Int. Conf. on Autonomic and Trusted Computing, ATC 2006, (LNCS, 4158), 2006, pp. 342-351.

[17] Yu, F., Zhang, H., Yan, F., Gao, S. An improved global trust value computing method in P2P system. // Third Int. Conf. on Autonomic and Trusted Computing, ATC 2006, (LNCS, 4158), 2006, pp. 258-267. DOI: 10.1007/11839569_25

[18] Aberer, K.; Despotovic, Z. Managing trust in a peer-2-peer information system. // presented at the Proceedings of the tenth international conference on Information and knowledge management, Atlanta, Georgia, USA, 2001. DOI: 10.1145/502585.502638

[19] Chen, M.; Singh, J. P. Computing and using reputations for internet ratings. // presented at the Proceedings of the 3rd ACM conference on Electronic Commerce, Tampa, Florida, USA, 2001. DOI: 10.1145/501158.501175

[20] Dellarocas, C. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. // presented at the Proceedings of the 2nd ACM conference on Electronic commerce, Minneapolis, Minnesota, United States, 2000. DOI: 10.1145/352871.352889

[21] Sen, S.; Sajja, N. Robustness of reputation-based trust: boolean case. // presented at the Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1, Bologna, Italy, 2002. DOI: 10.1145/544741.544808

[22] Buchegger, S.; Boudec, J. L. Coping with False Accusations in Misbehavior Reputation Systems for Mobile Ad-Hoc Networks. // EPFL tech. rep. IC/2003/31, EPFL-DI-ICA, 2003.

[23] Josang, A. et al. A survey of trust and reputation systems for online service provision. // Decis. Support Syst. 43, (2007), pp. 618-644. DOI: 10.1016/j.dss.2005.05.019

[24] Langheinrich, M. When trust does not compute - the role of trust in ubiquitous computing. // presented at the Proc. 5th Int'l. Conf. Ubiquitous Comp, Seattle, WA, 2003.
[25] Blaze, M. et al. The keynote trust-management system V2, RFC 2704 1999.
[26] Advance Encryption Standard: http://www.ietf.org/rfc/rfc3962.txt, accessed on 28 Aug 2012.
[27] Rao, J.; Shekita, E. J.; Tata, S. Using Paxos to build a scalable, consistent, and highly available datastore. // Proc. VLDB Endow. 4, 4 (January 2011), pp. 243-254. DOI: 10.14778/1938545.1938549
[28] Hurley, R. T.; Soon Aun, Y. File migration and file replication: a symbiotic relationship. // Parallel and Distributed Systems, IEEE Transactions on. 7, (1996), pp. 578-586. DOI: 10.1109/71.506696
[29] Walsh, D.; Lyon, B.; Sager, G.; Chang, J. M.; Goldberg, D.; Kleiman, S.; Lyon, T.; Sandberg, R.; Weiss, P. Overview of the Sun Network Filesystem. // In Winter Usenix Conference Proceedings, Dallas. 1985.
[30] Sandberg, R.; Goldberg, D.; Kleiman, S.; Walsh, D.; Lyon, B. Design and Implementation of the Sun Network Filesystem. // In Summer Usenix Conference Proceedings, Portland. 1985.
[31] Tang, M.; Lee, B.-S.; Tang, X.; Yeo, C.-K. The impact of data replication on job scheduling performance in the Data Grid. // Future generation Computer Systems. 22, 3(2006), pp. 254-268. DOI: 10.1016/j.future.2005.08.004
[32] Cao, J.; Spooner, D. P.; Jarvis, S. A.; Nudd, G. R. Grid load balancing using intelligent agents. // Future Generation Computer Systems. 21, 1(2005), pp. 135-149. DOI: 10.1016/j.future.2004.09.032
[33] Yan, K. Q. et al. A hybrid load balancing policy underlying grid computing environment. // Journal of Computer Standards & Interfaces. (2007), pp. 161-173.
[34] Payli, R. U. et al. DLB—a dynamic load balancing tool for grid computing. // Scientific International Journal for Parallel and Distributed Computing. 07, 02(2004).
[35] Yagoubi, Y. Slimani. Task load balancing for grid computing. // Journal of Computer Science. 3, 3(2007), pp. 186-194. DOI: 10.3844/jcssp.2007.186.194
[36] Nehra, N.; Patel, R. B.; Bhatt, V. K. A framework for distributed dynamic load balancing in heterogeneous cluster. // Journal of Computer Science. (2007). DOI: 10.3844/jcssp.2007.14.24
[37] Cassandra. http://cassandra.apache.org.
[38] Nukarapu, Dharma; Tang, Bin; Wang, Liqiang; Lu, Shiyong. Data Replication in Data Intensive Scientific Applications with Performance Guarantee. // IEEE Trans. Parallel Distrib. Syst. 22, 8(2011), pp. 1299-1306. DOI: 10.1109/TPDS.2010.207
[39] Rao, H.; Skarra, A. A transparent service for synchronized replication across loosely-connected file systems. // in Services in Distributed and Networked Environments, 1995., Second International Workshop on, 1995, pp. 110-117.
[40] Domenici, A. et al. Relaxed Data Consistency with CONStanza. // presented at the Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. DOI: 10.1109/CCGRID.2006.84
[41] Guy, L. et al. Replica Management in Data Grids. // Technical report, GGF5 Working Draft, Edinburgh, Scotland, 2002.
[42] Yuzhong, S.; Zhiwei, X. Grid replication coherence protocol," in Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, 2004, pp. 232-239. DOI: 10.1109/ipdps.2004.1303278
[43] Huang, C. et al. Massive Data Oriented Replication Algorithms for Consistency Maintenance in Data Grids. // Computational Science – ICCS 2006. vol. 3991, V. Alexandrov, et al., Eds., ed: Springer Berlin / Heidelberg, 2006, pp. 838-841.
[44] Dullmann, D.; Segal, B. Models for Replica Synchronisation and Consistency in a Data Grid. // presented at the Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing, 2001. DOI: 10.1109/HPDC.2001.945177
[45] Zomaya, A. Y. Parallel and distributed computing handbook: McGraw-Hill Professional, 1996.
[46] Milner, R. Communication and Concurrency: Prentice Hall, 1989.
[47] Ghemawat, S.; Gobio, H.; Leung, S.-T. The google file system. // SIGOPS Oper. Syst. Rev. 37, 5(2003), pp. 29-43. DOI: 10.1145/1165389.945450
[48] Satyanarayanan, M. A Survey of Distributed File Systems. In Annual Review of Computer Science, Annual Reviews, Inc., Palo Alto, CA, 1989.
[49] Mishra, S.; Kushwaha, D. S.; Misra, A. K. Hybrid reliable load balancing with mosix as middleware and its formal verification using process algebra. // Future Gener. Comput. Syst. 27, 5(2011), pp. 506-526. DOI: 10.1016/j.future.2010.12.007
[50] Baumgartner, K. M.; Wah, B. W. Computer Scheduling Algorithms: Past, Present, and Future. // Information Science. 57-58, (1991), pp. 319-345. DOI: 10.1016/0020-0255(91)90085-9

Authors' addresses

Manu Vardhan
Department of Computer Science and Engineering,
Motilal Nehru National Institute of Technology Allahabad
Allahabad - 211004, India
+918853038545
vardhanmanu@gmail.com

Dharmender Singh Kushwaha
Department of Computer Science and Engineering,
Motilal Nehru National Institute of Technology Allahabad
Allahabad - 211004, India
dsk@mnnit.ac.in