Journal of Computing and Information Technology - CIT 23, 2015, 3, 191–210
doi:10.2498/cit.1002490

An Integrated Distributed Storage Design Offering Data Retrievability and Recoverability Using Soft Decision Decoding of Block Codes

C. K. Shyamala and T. R. Padmanabhan

Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham (University), Coimbatore, India

Active distributed storages need to assure both consistency and dynamic data support, in addition to availability, confidentiality and resiliency. Further, since storage durability suffers in untrusted and unreliable environments, it becomes crucial to (a) select the most reliable set of servers to assure data retrievability and (b) dynamically identify errant servers and restore the data to ensure data recoverability. We address the issues of concurrency, consistency, dynamic data support, data share repair, and trust management in providing persistent storage and access. The paper focuses primarily on erasure coded distributed storages (storages employing erasure coding for data dispersal). Integration of Quorum based approach using Notification propagation, with a reliability model based on server trust-reputation forms the comprehensive design proposed. Treating servers and their data shares equally at data reconstructions during data retrievals is rather inadequate in untrusted environments. The design provides a suitable platform for use of Soft Decision Decoding to overcome this inadequacy. The design has been validated by the simulation, study, and analysis carried out for Reed Solomon coded storage with varying levels of resiliency and concurrency. The proposed design can be suitably adapted in typical distributed information storages catering to global networked audience in public, untrusted, and unreliable operating environments.

Keywords: distributed storage systems, data retrievability, data recoverability, block codes, soft decision decoding

1. Introduction

Distributed storages operate with a huge user base, across public networks and no longer function in trusted and reliable environments (M. Placek and R. Buyya, 2006). In this regard there is continuous effort to maintain and assure guarantees to persistent storage and access. Storage designs evolve to incorporate measures that circumvent issues introduced by malicious activities and hybrid failures (crash and non-crash). It centers primarily on enabling trusted & reliable servers and correct & consistent data. A comprehensive design guaranteeing persistent storage and access in untrusted and unreliable environments is in focus. The work primarily focuses on active distributed storage systems employing erasure coding for data dispersal (DSS-D).

The 'distributed attribute' of DSS-D (block codes – Information Dispersal Algorithm (IDA), Error Correcting Codes (ECC)&c) offers confidentiality in addition to availability, and resiliency for storage and retrieval. To provide data retrievability, it is generally required that a minimal set of any k servers be available for reconstruction (M. Rabin, 1989, J. A. Garay et al., 2000, G. R. Goodson et al., 2004, J. Hendricks et al., 2007). Further, data may be retrieved using Hard Decision Decoding (HDD) which treats the data shares of the k servers equally.

In the presence of interleaved reads and writes, design for active DSS-D must extend to preserve the same level of write / update of file F among all storage servers and to return the same up-to-date version in successive reads of F. Clearly, with interleaved reads and writes data retrievability is restricted to k up-to-date file shares requiring a minimal set of any k up-

to-date servers. To meet this restriction on the minimal set of servers and file shares, it is essential that the design provides for both dynamic data support and consistency. Further, operation in untrusted and unreliable environments is expected and storages invariably endure hybrid failures (crash and non-crash, malicious activities). Evidently, the restriction on file shares for data retrievability becomes more stringent; k correct and up-to-date file shares are required. It implies that a minimal set of any k correct & up-to-date servers need to be judiciously selected at retrievals for reconstruction. Further, it is rather infeasible to continue retrieving files by treating servers and their shares equally – usage of HDD is limited here.

Hybrid failures are a threat, not only to data correctness but also to the coordination that has to be achieved among the servers for dynamic data support and consistency. Data recoverability at errant servers (that have suffered loss of propagation or file share corruption) is crucial for persistent storage & access and storage durability. Therefore, it is not only mandatory to identify correct & up-to-date servers at retrievals, but it is equally important to restore file shares at those servers that have suffered loss of propagation or file share corruptions.

Evidently, addressing the issues of concurrency, consistency, dynamic data support along with dynamic identification of errant servers, file share repair, and trust are central to persistent storage and access in DSS-D. In this paper, we address these issues to guarantee retrievability and recoverability for persistent storage and access at DSS-D. The proposed DSS-D design is equipped with mechanisms for dynamic data support, consistency, trust management, dynamic identification of errant servers, and correction of their file shares. Integration of a Quorum based approach using Notification propagation, with a reliability model based on server trust-reputation forms the comprehensive design proposed.

Our contributions in the proposed work for erasure coded active storages comprises a sufficiently comprehensive design that:

i. guarantees significant resiliency to malicious activities and hybrid failures at storage.

ii. does not compel a rather imperceptive and equal treatment of servers and their file shares in unreliable and untrusted environments.

iii. permits reads, writes, and updates to be performed persistently and reliably with only a marginal increase in the associated complexity.

iv. integrates a reliability model based on server trust-reputation in providing a platform for the use of SDD in reconstructions at data retrievals.

v. permits reliable and persistent access despite severe defective fractions at storage.

The paper presents a qualitative assessment of the proposed design examining the probability of availability of the quorum of servers, the response time and the communication overhead in terms of the number of messages required to perform reads, writes and updates. The study, analysis, and validation of the byte oriented access of the RS code set $(255, k)$ for varying levels of resiliency and concurrency bring out the effectiveness of the design in tolerating severe defective fractions at storage. Significantly, the design can be adapted suitably in typical distributed information stores of organizational data (such as federal, healthcare &c) that cater to a global networked audience in any public untrusted and unreliable operating environments.

The rest of the paper is organized as follows: Related work in Section 2 discusses the need and requirements for persistent access and storage in erasure coded active storages operating in hybrid failure prone, untrusted environments. On the onset, Section 3 describes four issues central to guaranteeing retrievability and recoverability for persistent storage and access in DSS-Ds. Integration of the Quorum based approach with Notification propagation for support of dynamic data and consistency is elaborated in detail here. Further, the section describes reliable handling of reads, writes and updates and the same for concurrent operations. The section progresses to explain the integration of a reliability model based on server trust-reputation. The proposed Extended Trust Model explicates the use of 'Server Trust' for interleaved and/or concurrent reads, writes and updates. Section 4 analytically evaluates the

design for response time, availability and communication overhead. Further, validation of the design through the simulation carried out for byte oriented access with a set of $(255, k)$ RS codes for varying levels of resiliency and concurrency is discussed. Interesting observations that validate availability, persistent access and storage are listed and discussed. Finally, Section 5 concludes the paper highlighting the comprehensiveness of the design in enabling the most appropriate mechanisms to prevent malicious activities and hybrid failures from obstructing availability, persistent storage & access, and storage durability.

2. Related Works

DSS (Distributed Storage System) operate in globally networked servicing arena with continuous efforts to improve collective performance. To enable recovery amongst server errors and failures, DSS typically spread data across servers with redundancy (K. D. Bowers et al., 2009, L. Gao et al., 2010). While replication is used in storages, dispersal techniques (M. Rabin, 1989, J. A. Garay et al., 2000, G. R. Goodson et al., 2004, J. Hendricks et al., 2007) provide a classic alternative in achieving redundancy. Information Dispersal Algorithm (IDA) (M. Rabin, 1989) fragments data into shares and disperses them among storage servers, reducing the level of redundancy needed to achieve robustness amidst server errors and failures. A significant line of research has been on the use of block (erasure) codes (K. D. Bowers et al., 2009, G. R. Goodson et al., 2004, M. Lillibridge et al., 2003, C. Wang et al., 2009, A. Juels, and B. S. Kaliski Jr., 2007, C. Cachin and S. Tessaro, 2005, C. Cachin and S. Tessaro, 2006, L. Pamies-Juarez, F. Oggier and A. Datta, 2013) to achieve dispersal.

Storages operate with a huge user base, across public networks and face the challenges of unreliability, unpredictability and potentially malicious behavior (M. Placek and R. Buyya, 2006). They do not limit themselves to providing assurance for static files – archival storages; they expand to provide for file updates – active storages. Efforts to provide persistent storage and access in DSSs are directed towards addressing concurrency, consistency, durability and their related issues in the presence of data updates at storage. Research has focused on consistency issues in the presence of Byzantine failures for erasure coded storage (G. R. Goodson et al., 2004, C. Cachin and S. Tessaro, 2006). Although Cachin and Tessaro (2006) ensure consistent stored information despite Byzantine faults, they do not address the issues of concurrency and versioning. In addition to versioning, Goodson et al. (2004) also perform file share repair among the servers. They exploit local data versioning within the storage nodes to provide consistency and perform file share repair by using additional historical fragments fetched from the servers. RobuSTore (H. Xia and A. A. Chien, 2007) in an attempt to increase data access performances in distributive environments, combines rateless erasure code with speculative access. Updates, though not the main focus, are also considered. With no adversaries assumed in the distributive environment, accesses to stored data are devoid of correctness issues here.

Adversary models representing malicious activities in distributed storage have progressed (K. M. Martin, 2008) from passive to active and to mobile adversaries. Significant line of research has taken into consideration active adversaries (K. D. Bowers et al., 2009, C. Wang et al., 2009, J. Kubiatowicz et al., 2000). Focus of research is on strong (C. Wang et al., 2009) and mobile adversary as well (K. D. Bowers et al., 2009). OceanStore (J. Kubiatowicz et al., 2000), provides for both active and archival storages in untrusted environment, using repetition coding for the former and erasure coding for the latter. A primary tier of replicas cooperate with one another in a Byzantine agreement protocol to choose the final commit order for updates; updates are epidemically communicated to the secondary tier. HAIL (K. D. Bowers et al., 2009) addresses the related issues regarding verification and correction of file shares among mobile adversaries. It uses a proactive strategy that cryptographically verifies and reactively reallocates file shares to prove that stored files are intact and retrievable. The protocols here provide assurance for static files and not for file updates.

In general, data reconstruction in DSS-D is possible from any set of k data shares out of the n dispersed (C. Cachin and S. Tessaro, 2005). In

storages that operate amongst hybrid failures, this flexibility of reconstructing the original data from a minimal possible (k) set of data shares is linked primarily to the selection of reliable servers. Clearly, data availability directly depends on the minimal possible (k) set of data shares from reliable servers. Verification of dispersed data among servers can assure the supply of the minimal data share set for reconstruction. Integrity checks on remote storage can be done by clients – playing the role of a verifier (K. D. Bowers et al., 2009, C. Wang et al., 2009) or can be enforced within the storage servers themselves (J. A. Garay et al., 2000, G. R. Goodson et al., 2004, J. Hendricks et al., 2007, C. Cachin and S. Tessaro, 2006). Cryptography based verification approaches have been used in (M. Lillibridge et al., 2003, C. Wang et al., 2009, A. Juels, and B. S. Kaliski Jr., 2007, C. Cachin and S. Tessaro, 2005, D. L. G. Filho and P. S. L. M. Barreto, 2006, M. A. Shah et al., 2007, H. Krawczyk, 1993) to perform integrity checks.

Clearly, levels of unpredictability and unreliability in the operating environments are not merely governed by hardware related failures. Occurrences of malicious activities and non-crash faults bring in the issues of trust; the operating environment is no longer the usually more trusted one, it is rather partially trusted and at the extreme totally untrusted (M. Placek and R. Buyya, 2006). Efforts to provide persistent storage and access should further extend to include those that circumvent issues introduced by malicious activities. It centers primarily on enabling trusted & reliable servers and correct & consistent data. While trust can be established via Byznatine protocol (A. Adya et al., 2002, J. Kubiatowicz, et al., 2000), it can be also established based on the reliable information provided on their behavior by a trusted third party – not always feasible (F. Gómez Mármol and G. Martínez Pérez , 2010). On the other hand, an alternative way to establish trust (H. Li and M. Singhal, 2007) is via Reputation Scheme – rewarding and penalizing good and bad behavior (M. Placek and R. Buyya, 2006). Trust / Reputation models have been developed using Analytic expressions, Fuzzy Logic, Bayesian Networks, Social Networks and Bio-inspired algorithms (A. Boukerche et al., 2007, G. Zacharia and P. Maes, 2000, C. Huang et al., 2006, S. Kamvar et al., 2003). A trust-reputation approach based on the concept of 'Server Trust' is established in (C. K. Shyamala and T. R. Padmanabhan, 2014). The work presents a well-organized scheme for selection of the most reliable set of servers to supply reliable data shares for reconstruction of data based on the trust-reputation of the servers' themselves.

It is evident that hybrid failures in storage environments restrict reconstruction of the original data to a set of reliable servers (to serve correct data shares). This demands that the k file shares used in decoding be correct; it also implies that file shares need not be treated equally in decoding. Conventional RS decoding treats all file shares equally and reconstructs the file using the well established Hard Decision Decoding (HDD). For such an approach with DSS-D, every storage server – malicious as well as non-malicious – has to be treated equally; this is not practical. This requirement is best served by using a Soft Decision Decoding (SDD) algorithm (B. Yamuna, and T. R. Padmanabhan, 2012) in place of a naïve approach using HDD. SDD algorithms chalk out an optimal iterative procedure to identify the correct k file share set in reconstructing the original data.

File share corruptions are not of the 'silent event' category (K. D. Bowers et al., 2009). Therefore, provisions for storage correctness need not necessarily be proactive. Data share verification and correction at servers follow every successful reconstruction, ensuring storage correctness. This reactive strategy is most appropriate for ensuring correctness at DSS-D.

From all the above, it is evident that active distributed storages operating in a globally networked servicing arena should incorporate measures that circumvent issues introduced by malicious activities and hybrid failures. It centers primarily on dynamically identifying errant servers and restoring the data and enabling trusted & reliable servers to provide correct & consistent data. The paper focuses on meeting these requirements for persistent access and storage in erasure coded active storages operating in hybrid failure prone, unreliable, and untrusted environments. It addresses interleaved and/or concurrent reads & writes and their related issues in providing (persistently and reliably) correct & up-to-date data, while ensuring the same level of writes and correctness at storage.

3. Proposed Work

A DSS-D with n storage servers guarantees a successful read of stored file (data) F, with the availability of a minimal set of any k servers. Here, at most $n-k$ failures are tolerated implying resiliency level of $n-k$. Further, a confidentiality level of k is also implied; that is, no collusion of at most $k-1$ storage servers (among the total of n) can retrieve F. Figure 1 is an overview of data retrieval and restoration in DSS-D, highlighting the need and the role played by mechanisms that allow dynamic identification of errant servers and correction of file shares in assuring persistent access and storage.

A two-tiered locally centralized architecture (M. Placek and R. Buyya, 2006) is adopted for the DSS-D design. T2, the Storage tier, is an untrusted tier of servers to which the storage responsibility is entrusted. T1 is the Client-Servicing tier of high performing, trusted super nodes, sufficient in number, interacting with the T2 servers for file read, update and write $-F(r-u-w)$. DSS-D clients (Service Clients) have no direct access to the T2 servers; they contact directly T1 servers for $F(r-u-w)$. For an (n,k) code dispersal in DSS-D, each of the n storage servers stores a file share / fragment of file F. A minimum of k of these file shares is required to reconstruct a file dispersed among n storage servers (C. Cachin and S. Tessaro, 2005, S. Lin and D. J. Costello Jr., 2004). The design here uses $(n=255,k)$ Reed Solomon (RS) code (Lin and D. J. Costello Jr., 2004, S. Peter, 2002, R. H. Morelos-Zaragosa, 2006). However, it is general enough to be applicable to any other block code. The byte oriented access of this code makes it the most logical choice for DSS-D. Availability, confidentiality, resiliency levels, and concurrency potential for a set of $(255,k)$ RS codes are in Table 1.

3.1. DSS-D Design Issues

The issues that arise in guaranteeing retrievability and recoverability for persistent access and storage in DSS-D and the proposed mechanisms adopted in our design are:

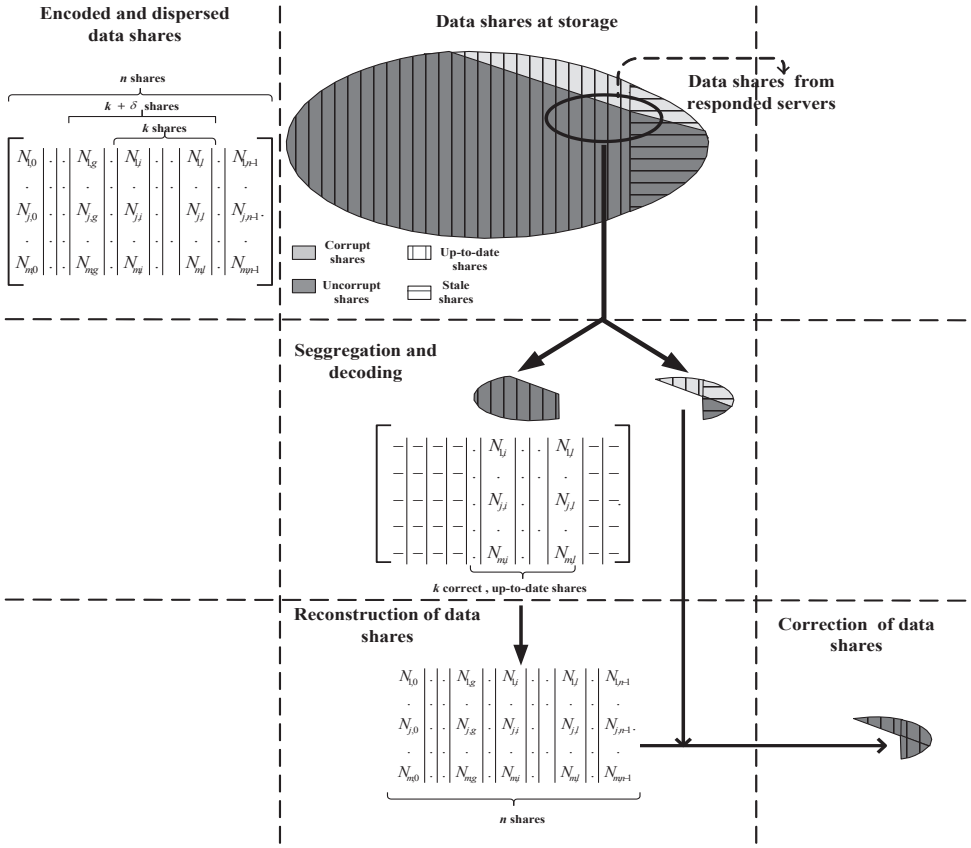


Figure 1. Overview of data retrieve and restore in DSS-D.

Code	Availability	Confidentiality level	Resiliency level	Concurrency potential
(n/k)	k	$k-1$	$n-k$	n/k
$(255, 46)$	46	45	209	5.54
$(255, 64)$	64	63	191	3.98
$(255, 110)$	110	109	145	2.31
$(255, 238)$	238	237	17	1.07

Table 1. Attributes and their values for $(255, k)$ code.

1. In active DSS-D storages, data retrievability requires that a minimal set of any k servers be available for reconstruction, each with its own up-to-date file share. An (n, k) code implicitly offers potential for concurrency, based on the extent to which n is greater than k, as given in Table 1. Providing monotonic reads and sequential consistency with concurrent reads and writes demands preserving the same level of write at the T2 tier – the first issue. For this, the proposed design is augmented with a push-pull mechanism to preserve the same level of writes at T2. Using Notification Propagation, file writes are pushed to the T2 tier; subsequently, the file shares are pulled by the corresponding servers. A file write is performed on $(k+\delta)$ file shares with $(k+\delta)$ participating T2 servers. Write notification is propagated to the rest of the non-participating T2 servers (having the remaining $n-(k+\delta)$ file shares). Updates are supported at file level; a file is updated by overwriting its shares at the servers. Concurrent writes are performed by serialization of writes (concurrent writes are not aborted); this enables ordering and servicing of concurrent write requests on files.

2. In the presence of hybrid failures at T2 tier, data retrievability demands a minimum of k up-to-date & correct file shares – the next issue. The availability of reliable servers – a minimal set of k reliable servers – is required for reconstruction. In our design, we adopt a reliability model based on server trust-reputation built over the periods of DSS-D servicing (C. K. Shyamala and T. R. Padmanabhan, 2014) to provide a judicious means for selecting a set (minimal) of k reliable servers for file reconstruction. We utilize the trust-reputation approach based on the concept of 'Server Trust' with SDD to identify the correct k file share set.

3. With concurrent reads and writes, the design discussed so far is still insufficient to guarantee that a read must always return the last updated file – the third issue. Measures should be in place to ensure that only the same correct & up-to-date copy of the file is returned at every successive file read. For a concurrent read with write, the last updated file / the file not older than the last read should be returned (old copy of the file is never returned on a read). We exploit the Quorum based approach to reach a consensus on the latest updated copy of the file shares. A file read can be performed with $(k+\delta)$ file shares from k reliable servers, only after a consensus on the latest updated copy of the file shares is reached in the Quorum.

4. Hybrid failures at T2 tier pose a threat for storage correctness – the fourth issue. We adopt a reactive approach towards guaranteeing storage correctness. Successful reconstructions with k reliable servers (providing k correct & up-to-date file shares) allow all n file shares to be generated. With these n file shares, dynamic identification and correction of incorrect and stale (not up-to-date) file shares in the T2 tier follow directly. Thus, data recoverability is guaranteed in the design.

We propose a sufficiently comprehensive design that addresses these four issues central to guaranteeing retrievability and recoverability for persistent storage and access in DSS-Ds. The design integrates:

i. Notification propagation to preserve the same levels of writes

ii. Reliability model based on server trust-reputation for selection of reliable servers

iii. Quorum based approach to reach a consensus on updated file shares

The integration makes it possible – with the highest probability – to supply k correct & up-to-date file shares during reads; similarly correct and up-to-date version number for data updates. The design is augmented with a reactive approach to ensure correctness at the storage tier.

3.2. Data Updates

Active storages allow updates on data; the challenge is to assure that a read always returns the result of the latest completed update. With updates of file F in DSS-D, a read of F should return only the up-to-date version of F and significantly, the same up-to-date version of F should be returned for every successive read of F.

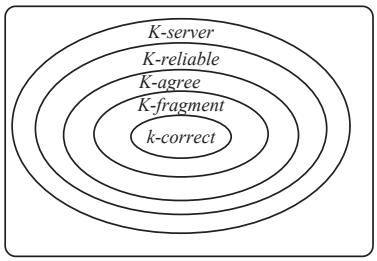
Consistency: It should be always practicable with a high probability to return the same up-to-date version of file F, for any two or more successive reads of F. Consistency guarantees can be assured if the design supports:

i. consensus on updated version of the data for reads

ii. propagation of the same level of data on writes.

3.2.1. Support for Consistency

The unit of operation in $F(r{-}u{-}w)$ is (k), the minimum number of fragments of file F required to perform an operation of F. Structuring of server and fragment sets for performing an $F(r{-}u{-}w)$ is illustrated in Figure 2. Servers are pooled and *K-server* is formed as in Figure 3. A file F and each of its fragments have distinct identification (Figure 5). A new file write $-F(w)$ – is performed by writing the file fragments to the responded server set in the



K-Server set: *K-server* = $\{k_s + \delta_s\}$ servers

k_s being the minimum no. of T2 servers that are required to respond for file servicing, and δ_s being the no. of T2 servers that may additionally respond to file servicing.

K-Reliable server set: *K-reliable* = $\{k_s + \delta_s\}$ reliable servers

k_s being the minimum no. of reliable T2 servers that are required for file servicing, and δ_s being the no. of reliable T2 servers that may additionally be present for file servicing.

K-Agreement server set: *K-agree* = $\{k_s + \delta_s\}$ agreeing servers

k_s being the minimum no. of servers in *K-reliable* that are required to agree on the highest version number V_h of a file F, and δ_s being the no. of servers in *K-reliable* that may additionally agree on the highest version number V_h of a file F.

K-fragment set: *K-fragment* = $\{k_f + \delta_f\}$ reliable fragment shares

k_f being the no. of reliable fragment shares in *K-agree* that are required to be available with the ascertained quorum, and δ_f being the no. of reliable fragment shares in *K-agree* that may additionally be available in the quorum.

Correct fragment set: *k-correct* = $\{k_f\}$ correct fragment shares

k_f being the no. of correct fragment shares in *K-fragment* that are required for file reconstruction.

Here, δ_s and δ_f are ≥ 0.

Figure 2. Structure of server and fragment sets.

1. Randomly select $(k + \delta)$ T2 servers.
2. Transmit client request.
3. Attempt to form the K-Server set

If the attempt fails,

retransmit client request to the other randomly selected δ T2 servers
repeat until K-Server set is formed.

Figure 3. Procedure to identify *K-server*.

1. Form (responded server set) *K-server* with $\{k_s + \delta_s\}$ that responded to the read request.
2. Form *K-reliable* with $\{k_s + \delta_s\}$ reliable servers from *K-server* using STI.
3. Get a consensus on the highest version no. V_h. Form *K-agree* with $\{k_s + \delta_s\}$ reliable servers agreeing on V_h for File$_i$.
4. Form *K-fragment* with $\{k_f + \delta_f\}$ reliable fragment shares from *K-agree*.
5. Generate *k-correct* (k_f correct fragment shares) using – SDD – an iterative reconstruction procedure. Reconstruct the file (File'$_i$) and perform file read.
6. Generate all n fragment shares using File'$_i$.
7. For all the servers in {*K-server* – servers that supplied *k-correct*}, verify the fragment shares and restore.

Figure 4. Procedure for quorum formation for $F(r)$.

1. Form (responded server set) *K-server* with $\{k_s + \delta_s\}$ that responded to the write request.
2. Generate the new version no. (current logical clock (L. Lamport, 1978)) for File$_i$.
3. Encode File$_i$ and generate the n fragment shares.
4. Write any $\{k_f + \delta_f\}$ fragment shares of File$_i$ in *K-server*.

File ID	**Fragment ID**
File tag || Version no. || Server ID	Fragment tag || File ID
Unique file tag || current logical clock || T1 server ID	Unique fragment tag || File ID

Figure 5. Procedure for quorum formation for $F(w)$.

1. Form (responded server set) *K-server* with $\{k_s + \delta_s\}$ that responded to the update request.
2. Form *K-reliable* with $\{k_s + \delta_s\}$ reliable servers from *K-server* using STI.
3. Get a consensus on the highest version no. V_h. Form *K-agree* with $\{k_s + \delta_s\}$ reliable servers agreeing on V_h for File$_i$.
4. Generate the updated version no. (File$_i$ last write logical clock + 1) for the updated file File$_{ui}$.
5. Encode File$_{ui}$ and generate n fragment shares.
6. Write $\{k_f + \delta_f\}$ fragment shares of File$_{ui}$ in *K-server*.

Figure 6. Procedure for quorum formation for $F(u)$.

write quorum (Figure 5). Updates on files are supported at the file level; a file is updated by overwriting its shares at the servers. An update on $F - F(u)$ – is performed as illustrated in Figure 6. A consensus is reached on V_h – the highest version number – of F in the update quorum. The version number is generated and the file fragments for the file update are written

to the responded server set in the quorum. A file read $-F(r)$ – is performed as illustrated in Figure 4. A consensus on the latest updated copy of the file shares is reached in the read quorum. A reliable fragment set, *K-fragment* is formed with *K-agree*. Using SDD, a correct fragment set, *k-correct* is obtained from *K-fragment*. The file is reconstructed and all n fragments are generated facilitating subsequent verification and correction.

3.2.2. Dynamic Data Support

The same level of write of F among all T2 servers is required in order to enable a read of F to return the up-to-date version. Dynamic data support at DSS-D mandates integration of an appropriate propagation mechanism with the quorum design discussed above. The challenge here is to achieve a well-organized coordination among the client servicing servers and the servers that share storage responsibility. To meet this requirement, we take advantage of Notification Propagation (L. Gao et al., 2010) to propagate writes of F to the non-participant T2 servers. Significantly, the integration of the Quorum based approach with Notification propagation in our design eliminates the need for the intersection of reads and writes (D. Agrawal and A. El Abbadi, 1990) mandatory in the traditional quorums.

A push technology is employed to push write (new file write / file update) information to the non-participant T2 servers. The proposed propagation scheme propagates notification of new file writes and file updates to the non-participants using Instate and Invalidate messages respectively. Non-participants of a new file write are a class of servers – instated server class – that should be informed of the file write. Non-participants of a file update are a class of servers – invalidated server class – that should not only be informed but also be invalidated for the file update. The push is subsequently followed by a pull; the non-participants equipped with information about the latest write pull the necessary (pull technology) from T1. Client initiated file writes and updates are acknowledged when a write is performed in the quorum for a minimum of k fragments at a minimum of k servers. Server initiated writes follow file writes and updates. There are two cases of notification as in Figure 8:

i. Invalidate Through: Notification is sent to the T2 non-participants and they are invalidated on a file update.

ii. Invalidate Suppress: In the case where invalidation has been performed in the previous update of F, the notification is suppressed.

3.3. Reliable Reads and Writes

A new file write is straightforward (Figure 5). For file update, the quorum based approach of the design requires a consensus on V_h of F (Figure 6). The design permits V_h of F to be fetched from the current T1 (performing the servicing) on failure to reach a consensus in the quorum (Figure 8). A read requires a quorum of reliable servers to agree on V_h of F before proceeding to reconstruct the file. The design permits file reads with cases of read hit and varying levels of read miss (Figure 7). A Read Hit occurs when the client-servicing T1 is itself the parent T1 of the requested file. A Read Miss-Level1 occurs when *K-agree* serves the *k-correct* for successful reconstruction. A Read Miss-Level2 occurs if *K-agree* is not formed or if it is unable

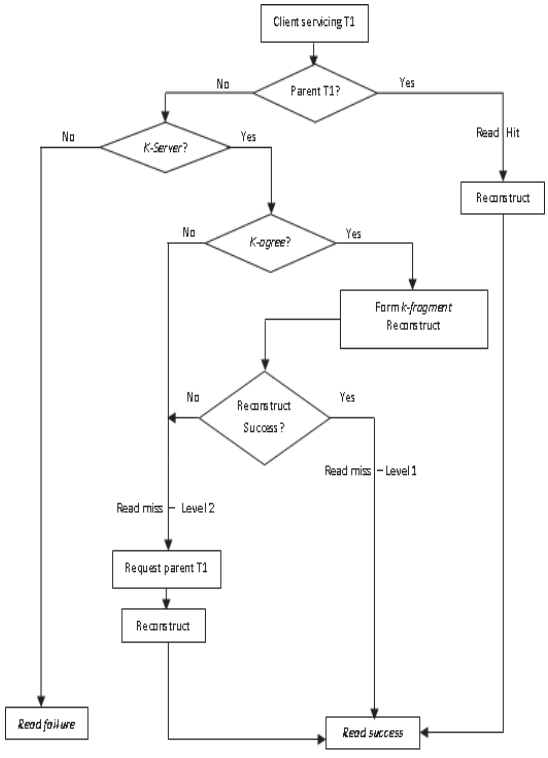


Figure 7. Read hit and miss cases.

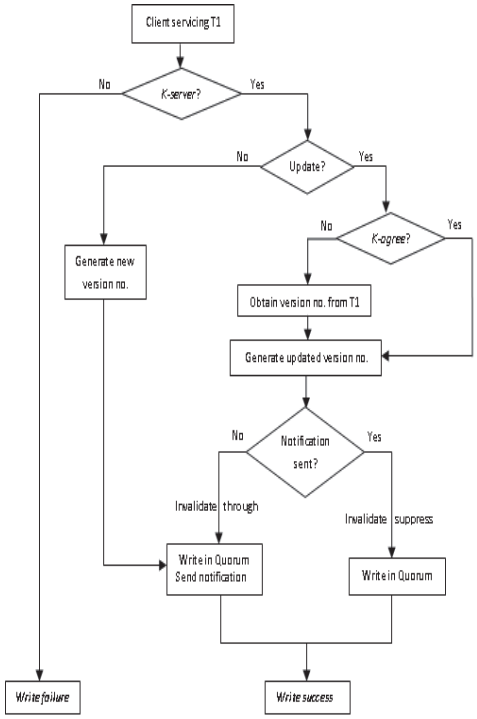


Figure 8. Write-Update notification cases.

to serve the *k-correct*. It requires the *k-correct* to be fetched from the Parent T1 for successful reconstruction. Clearly, the design permits file reads, writes, and updates to be performed persistently and reliably; failure occurs only when K-server cannot be obtained.

3.3.1. Concurrent Reads and Writes

An (n,k) code implicitly offers potential for concurrency, based on the extent to which n is greater than k, as given in Table 1. Concurrency potential in general is given by the quantity n/k, signifying the capacity of the DSS-D to support concurrent reads and writes. Distributed replication refers to the number of *k-correct* sets of F, supplied for concurrent reads and updates of F. This implies that $\lfloor \boldsymbol{n/k} \rfloor$ such sets are possible to be formed concurrently at any given time by the system to service as many reads and updates. The quantity $\boldsymbol{n - (k \lfloor n/k \rfloor)}$ signifies the number of storage server faults and failures that can be tolerated by the system in satisfying $\lfloor \boldsymbol{n/k} \rfloor$ concurrent reads and updates. Further, in the worst case, there can be a total of $\boldsymbol{n-k}$ failures in the system and yet enabling it to service a single read or update. Typical figures for a $(n = 255, k)$ scheme are given in Table 2. The design permits concurrent reads and writes without conflicts.

3.4. Trust Model

Data retrievability and recoverability guarantees in DSS-D rely heavily on the selection of reliable k servers from the available servers. There should be a judicious means for selecting a minimal set of k reliable servers among the available servers for file reconstruction. Step 2 in Figure 4 as well as in Figure 6 requires the reliable server set to be generated from the respondent servers. This involves the judicious selection of the servers. We adopt the reliability model (C. K. Shyamala and T. R. Padmanabhan, 2014) based on server trust-reputations built over the periods of DSS-D servicing, to provide an appropriate judicious selection procedure.

The model is – context specific, multifaceted and dynamic – comprehensive (Y. Wang and J. Vassileva, 2003). The basic model meets the four primary requirements (F. Gómez Mármol and G. Martínez Pérez, 2010) for DSS-D access:

A. Prevention of behavioral oscillations along time

RS code	Concurrency potential	Distributed replication	Resiliency Level
$(\boldsymbol{n,k})$	$\boldsymbol{n/k}$	$\lfloor \boldsymbol{n/k} \rfloor$	$\boldsymbol{n - (k \lfloor n/k \rfloor)}$
$(255, 46)$	5.54	5	25
$(255, 64)$	3.98	3	63
$(255, 110)$	2.31	2	35
$(255, 238)$	1.07	1	17

Table 2. Concurrency details for $(255, k)$ code.

B. Redemption of past malicious servers who have become benign

C. Prevention of abuse of a good achieved reputation

D. Assurance of reasonable level of participation for re-entrants.

Cumulative file share submission status in the past performances of servers forms the metric – 'Server Trust Index' (STI). Server STIs are used for selection of reliable server at data retrievals. They are refreshed based on the servers' performance in the current servicing by appropriate reward / punishment.

Update of STIs is given by:

$$\text{STI} - \text{new} = \begin{cases} S^* + \alpha \text{ on reward} \\ S^* - p \text{ on punishment} \end{cases} \quad (1)$$

where, $S^* = \dfrac{\sum_{i=n-c}^{n} a^{n-i} S_i}{n_c}$ and n, a^{n-i}, S_i; $c+1$ and n_c being total number of servicing, weightage for the performance in i^{th} servicing and STI at i^{th} servicing, moving average window length and last c servicing respectively.

In the distributed penalty scheme, penalty for j^{th} server is:

$$p_j = \frac{(S_j - S_0)\,\beta\, n_d}{\sum_{i=1}^{n_d} S_i - S_0} \quad (2)$$

n_d, S_0, β and S_j being the total number of errant participants in current servicing, least STI among n_d participants, punishment unit and last STI of the j^{th} server respectively.

The subjective approach in the model treats errant severs in a singular way. It provides a statistical means of reward & punishment, and selection of k reliable servers.

3.4.1. Extended Trust Model

The integrated design for active DSS-D extends the basic trust model to meet the two additional requirements (F. Gómez Mármol and G. Martínez Pérez, 2010):

E. Different trust-reputation rating based on the type of servicing

F. Importance of a transaction and its associated risk influencing subsequent punishment / reward.

The ordering of the $F(r{-}u{-}w)$ DSS-D servicing in terms of importance and risk is directly related to the role it plays in verification and data restoration at the T2 severs. $F(r)$ is treated as the most important as it guarantees both verification and data restoration at the T2 severs. The $F(u)$ requirement that a consensus be reached on the highest file version number allows verification at this level and not data restoration. The $F(w)$ requires any set of k servers to perform the write; it plays no role in verification and restoration and is treated as the least important. The same ordering applies to the trust-reputation rating and risk associated with $F(r{-}u{-}w)$.

For $F(r{-}u{-}w)$ servicing, $\alpha_r, \alpha_u, \alpha_w$ is used in place of the reward α in (1) and $\beta_r, \beta_u, \beta_w$ in place of β in (2). The more important the DSS-D servicing with respect to verification and restoration, the greater are the associated α and β. The distributed penalty scheme in (2) varies the trust-reputation rating for the $F(r{-}u{-}w)$ servicing. A new file write performed on the respondent server set results in only rewarding the participants. On a file read or update the participants are suitably rewarded only on enabling consensus of version number and supply of correct fragments as required; they end up receiving penalty otherwise. After every servicing, the T2 servers are updated for their STI with the awarded reward or penalty using the extended trust model.

4. Design Evaluation

The DSS-D design is qualitatively evaluated for response time, availability, and communication overhead. Concurrent and non-concurrent modes of simulation for $(255, k)$ codes for varying degrees of concurrency and resiliency levels are reported and discussed in the subsequent subsection.

4.1. Analytical Evaluation

4.1.1. Response Time

A client connects to a T1 server through LAN connection, and a T1 connects to T1 and T2 servers through WAN connection, with RTTs of t_{lan}(6 ms RTT) and t_{wan}(86 ms RTT) respectively (L. Gao et al., 2010). Let, r be the read % and $1-r$ be the write %. If r_d is hit and r_i is miss-level1, then miss-level2 is $r-r_d-r_i$. The best case and the worst case average response times are evaluated and given in Table 3.

4.1.2. Availability

New file write or file update is performed with the access of at least one T1 along with the quorum. It is rejected when (i) none of the T1s is available, or (ii) when enough number (k) of T2s is not available even though a T1 is accessed. Simultaneously, any k or more T2s not agreeing on version number does not affect the resultant file update (Figure 8). Read (Figure 7) is performed with the access of (i) the parent T1 (hit), or (ii) any one of the non parent T1s along with the quorum (miss- level1), or (iii) any one of the non parent T1s along with the availability of the parent T1 on an unsuccessful quorum (miss- level2).

Let, r be the read % and $1-r$ be the write % and the number of servers in T1 tier be n_1 with p_1, the probability of a T1 server being unavailable and $1-p_1$, the probability of it being available. Let the number of servers in T2 tier be n_2 with p_2, the probability of a T2 server being unavailable and $1-p_2$, the probability of it being available. Let $1-pshm$ be the probability of a T2 server being non-errant and $pshm$ be the probability of it being errant. The availability of $F(r-u-w)$ and its non-availability are given below in (i) and (ii) respectively:

(i)

$$(1-p_1^{n_1})\left\{\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}\,(1-p_2)^{k+i}\,(p_2)^{n_2-k-i}\right\}$$
$$+\,r\frac{1}{n_1}+r\left(1-\frac{1}{n_1}\right)\left[\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}(1-p_2)^{k+i}\cdot\right.$$
$$\left.\cdot(p_2)^{n_2-k-i}\left\{{}^{k+i}C_{k+j}\sum_{j=0}^{i}(1-pshm)^{k+j}(pshm)^{j}\right\}\right]$$
$$+r\left(1-\frac{1}{n_1}\right)(1-p_1)\left(1-\left[\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}(1-p_2)^{k+i}\cdot\right.\right.$$
$$\left.\left.(p_2)^{n_2-k-i}\left\{{}^{k+i}C_{k+j}\sum_{j=0}^{i}(1-pshm)^{k+j}(pshm)^{j}\right\}\right]\right)$$

(ii)

$$(1-r)\left\{(p_1^{n_1})+(1-p_1^{n_1})\left(\sum_{i=0}^{k-1} {}^{n_2}C_i\,(1-p_2)^i\,(p_2)^{n_2-i}\right)\right\}$$
$$+\,rp_1^{n_1}+r(1-p_1^{n_1})\sum_{i=0}^{k-1} {}^{n_2}C_i\,(1-p_2)^i\,(p2)^{n_2-i}$$
$$+\,r(1-p_1^{n_1})(p_1)\left(1-\left[\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}\,(1-p_2)^{k+i}\cdot\right.\right.$$
$$\left.\left.\cdot\,(p_2)^{n_2-k-i}\left\{{}^{k+i}C_{k+j}\sum_{j=0}^{i}(1-pshm)^{k+j}(pshm)^{j}\right\}\right]\right)$$

Here,

- $(1-p_1^{n_1})\left\{\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}(1-p_2)^{k+i}(p_2)^{n_2-k-i}\right\}$ is the probability that at least one T1 and, simultaneously, any k or more T2s are available for a write/update.

where,

$(1-p_2)^{k+i}(p_2)^{n_2-k-i}$: probability that one set of $k+i$ T2 alone is available and all others are not available.

${}^{n_2}C_{k+i}(1-p_2)^{k+i}(p_2)^{n_2-k-i}$: probability that all possible sets of $k+i$ T2 are available and, simultaneously, the others are not available.

$(1-p_1^{n_1})$: probability that at least one T1 is available (one or more up to n_1 T1s available) and, simultaneously, $\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}(1-p_2)^{k+i}(p_2)^{n_2-k-i}$: probability that all possible sets of k or more T2 are available.

- $p_1^{n_1}+(1-p_1^{n_1})\left\{\sum_{i=0}^{k-1} {}^{n_2}C_i(1-p_2)^i(p_2)^{n_2-i}\right\}$ is the probability that none of the T1s is available or that enough number (k) of T2s are not available with availability of one or more T1s for a write/update.

where,

$p_1^{n_1}$: probability that none of the T1s is available.

$(1-p_1^{n_1})$: probability that at least one T1 is available (one or more up to n_1 T1s available), but, k of T2s are not available.

$(1-p_2)^i(p_2)^{n_2-i}$: probability that only a specific set of i T2s are available and all others are not available.

${}^{n_2}C_i(1-p_2)^i(p_2)^{n_2-i}$: probability that all possible sets of i T2s are available and all others are not available.

$\sum_{i=0}^{k-1} {}^{n_2}C_i(1-p_2)^i(p_2)^{n_2-i}$: probability that the number of T2s available is less than k in all possible ways.

- $\frac{1}{n_1} + \left(1-\frac{1}{n_1}\right)\left[\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}(1-p_2)^{k+i}(p_2)^{n_2-k-i}\cdot \left\{{}^{k+i}C_{k+j}\sum_{j=0}^{i}(1-pshm)^{k+j}(pshm)^j\right\}\right] + \left(1-\frac{1}{n_1}\right)(1-p_1)\left(1-\left[\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}(1-p_2)^{k+i}(p_2)^{n_2-k-i}\cdot \left\{{}^{k+i}C_{k+j}\sum_{j=0}^{i}(1-pshm)^{k+j}(pshm)^j\right\}\right]\right)$

  is the probability that the file is reconstructed directly from the parent T1 (hit) or indirectly from the quorum with any one of the non parent T1s' access (miss -level1). Further, in case of unsuccessful attempt at quorum with a non-parent T1 for read, the file can be reconstructed by reverting to the parent T1 itself (miss -level2).

where,

$\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}(1-p_2)^{k+i}(p_2)^{n_2-k-i}$ is the probability that any set of k or more T2 are available.

Further, for any i in this, the probability that at least k of the $k+i$ T2s are regular: ${}^{k+i}C_{k+j}\sum_{j=0}^{i}(1-pshm)^{k+j}(pshm)^j$.

$\left(1-\frac{1}{n_1}\right)(1-p_1)$: probability that at least one T1 is available (one or more up to n_1 T1s available) along with the parent T1s availability.

- $p_1^{n_1} + (1-p_1^{n_1})\sum_{i=0}^{k-1} {}^{n_2}C_i(1-p_2)^i(p_2)^{n_2-i} + (1-p_1^{n_1})(p_1)\left(1-\left[\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}(1-p_2)^{k+i}\cdot(p_2)^{n_2-k-i}\left\{{}^{k+i}C_{k+j}\sum_{j=0}^{i}(1-pshm)^{k+j}(pshm)^j\right\}\right]\right)$

  is the probability that there is a no read at a hit $(p_1^{n_1})$ or

  at a miss, level1 $(1-p_1^{n_1})\sum_{i=0}^{k-1} {}^{n_2}C_i(1-p_2)^i(p_2)^{n_2-i}$, or

  level2 $(1-p_1^{n_1})(p_1)\left(1-\left[\sum_{i=0}^{n_2-k} {}^{n_2}C_{k+i}(1-p_2)^{k+i}(p_2)^{n_2-k-i}\left\{{}^{k+i}C_{k+j}\sum_{j=0}^{i}(1-pshm)^{k+j}\cdot(pshm)^j\right\}\right]\right)$.

4.1.3. Communication Overhead

Let $|Q_r|$, $|Q_w|$, and $|Q_u|$ be the size of the read, write, and update quorum respectively and let $|Q_{ri}|$ be the number of errant T2 servers in the read quorum. All messages are assumed to be of equal cost. A read hit costs 2 messages and a read miss of level1 costs $1+|Q_r|+|Q_{ri}|+n_1$ messages, whereas the cost of a read miss of level2 is $2+|Q_r|+|Q_{ri}|+n_1$ messages. The cost of a write is $1+|Q_w|+n_2+n_1$ messages. The cost of an update with invalidate suppress is $1+2|Q_u|+n_1$ messages, whereas the cost of an update with invalidate through is $1+|Q_u|+n_2+n_1$ messages.

Best case average response time				
Read hit	or	Read miss- level 1	+	Notification suppress
$(r_d)[t_{lan}]$	or	$(r_i)[t_{lan}+3t_{wan}]$	+	$(1-r)[t_{lan}+3t_{wan}]$
Worst case average response time				
Read miss - level 2			+	Notification through
$(r-r_d-r_i)[t_{lan}+4t_{wan}]$			+	$(1-r)[t_{lan}+4t_{wan}]$

Table 3. Response Time.

4.2. Simulation and Observations

The basic structure for simulation of archival DSS-Ds has the key parameters given in Table 4. To facilitate both concurrent and non-concurrent modes of active storages, the key parameters are augmented with concurrency level (cl) and resiliency level at cl (kd). These parameters situate the level of concurrency and the

Parameter	Description	Parameter	Description
n	total no. of servers	m	no. of sets > 2 for ensuring graded reliability among the servers based on their STI
k	minimal server subset	$p_i(fl)$	failure probability for the i reliability sets
F	failure set for hybrid failures	X	server's state set for errant and non-errant states
$p(fl)$	failure probability	mtn	re-entry interval for maintenance and re-entry
a	reliability weights for STI values	rl	run length
l	window length for past STIs		

Table 4. Configuration parameters for simulation of archival DSS-Ds.

level of resiliency for each servicing in a run. For an (n,k) code with concurrency potential of n/k %, a distributed replication of $\lfloor n/k \rfloor$, and a resiliency level of $n-(k\lfloor n/k \rfloor)$, cl and kd range with $1 \ldots \lfloor n/k \rfloor$ and $1 \ldots (n-(k\lfloor n/k \rfloor))/cl$ respectively.

The DSS-D with a set of $(255,k)$ codes for varying levels of concurrency and resiliency levels are analyzed in both concurrent and non-concurrent modes of servicing. STI updations in each run of the simulation are associated with α and p for $F(r{-}u{-}w)$ and conform to (1) and (2) of the extended trust model. Apart from the obvious encoding / decoding efforts required to write and read data with *k-correct*, levels of complexity vary primarily in file reads and updates based on the effort expended towards obtaining V_h and *k-correct* in the quorums. The reads and updates are categorized (Table 5) based on this variation in the required effort in the quorums.

A few clarifications on the levels of complexity encountered in file writes, reads and updates (given in Table 5) are in order here. File reads and writes primarily demand k reliable, agreeing servers. $O(k)$ complexity signifies the effort being limited to the first k in the respective server set; but, with the presence of defective fractions in this set, the associated complexity for k increases as its power – $O(k^{defective\ fraction})$. With greater defective fractions in this set, the need of an access to a T1 server further increases the complexity marginally to $O\left((k^{defective\ fraction})+1\right)$. Significantly, the order of complexity is limited to the maximum of $O\left(k^{defective\ fraction}\right)$ itself.

A new file write with any k available servers is straightforward. It has a complexity of $O(k)$ and is always Type A. A read hit by default has $O(k)$ complexity, whereas a read miss varies in the levels of complexity. A read miss requires *k-correct* to be obtained from the quorum for $F(r)$. It can be obtained from the first k in *K-reliable* with $O(k)$ or from any k in *K-reliable* in a read miss-level1 of complexity $O\left(k^{defective\ fraction}\right)$. Failing which, it may be fetched from the parent T1 in a read miss-level2 of complexity $O\left((k^{defective\ fraction})+1\right)$. This results in Type A, B and C categorizations respectively for file reads (as evident from Table 5). File updates too vary in the levels of complexity, irrespective of whether they are associated with invalidate through or suppress. Consensus on V_h, may be obtained from the first k in *K-reliable* with a complexity of $O(k)$, or it can be obtained from any k in *K-reliable* with a complexity of $O(k^{defective\ fraction})$. Failing which, it may be fetched from the client-servicing T1 with only a marginal increase in complexity of $O\left((k^{defective\ fraction})+1\right)$. This results in Type A, B and C categorizations respectively for updates (as evident from Table 5).

A detailed simulation in the non-concurrent mode using STI based server segregation among hybrid failures has already been reported (C. K. Shyamala and T. R. Padmanabhan, 2014). Maintenance schedules (re-entry interval) permit affected servers to re-enter the system after repair, but a more balanced schedule is required to permit servicing without interruption (to guarantee at least k correct severs available in every run). Variation of service duration with

F(r-u-w)	**Complexity at Quorum**	**Remarks**
1.Update	V_h consensus in *K-reliable*	
Type A	$O(k)$	first k in *K-reliable*
Type B	$O(k^{defective\ fraction})$	any k in *K-reliable*
Type C	$O\left((k^{defective\ fraction})+1\right)$	not in *K-reliable*
2. Read	*k-correct* in *K-agree*	
Type A	$O(k)$	Client-servicing parent T1 – Read hit / first k in *K-agree* – Read miss- Level1
Type B	$O(k^{defective\ fraction})$	any k in *K-agree*– Read miss-Level1
Type C	$O\left((k^{defective\ fraction})+1\right)$	not in *K-agree* - Read miss-Level2
3.Write	k in *K-server*	
Type A	$O(k)$	any k in *K-server*

Table 5. $F(r-u-w)$ Complexity.

mtn shown in Figure 9 is adopted from (C. K. Shyamala and T. R. Padmanabhan, 2014); clearly any *mtn* of 50 or less permits least k correct severs to be available in every run, (facilitating servicing without interruption) enabling a well balanced schedule.

Concurrent and non-concurrent modes of simulation were carried out for $(255,k)$ codes with an *mtn* of 50 for varying degrees of concurrency and resiliency levels (Table 2). Failure set (F) and failure probability $(p(fl))$ defined hybrid failures (defective fraction), with number of non-malicious failures less than malicious failures. The effect of defective fractions at T2 servers on availability (see Section 4.1) and related complexity in $F(r-u-w)$ were studied and analyzed.

Consider the DSS-D simulation after the defective servers have been repaired / replaced as part of the maintenance activity. Let x be the probability of defectives at the access; x represents the absolute defective fraction at every subsequent access. Fraction of non-defectives at the 1^{st} access $=(1-x)$. At the 2^{nd} access a fraction of $(1-x)$ out of the 1^{st} access remains non-defective; that is, the non-defective fraction at the second access is $(1-x)^2$. The non-defective fraction at the i^{th} access is $(1-x)^i$. The cumulative defective fraction in the m_{tn} successive accesses until the next maintenance instance is:

$$\sum_{i=1}^{m_{tn}}\{1-(1-x)^i\} \tag{3}$$

dno – the average defective fraction during the interval $\{1,m_{tn}\}$ is given by:

$$\begin{aligned} dno &= \frac{1}{m_{tm}}\sum_{i=1}^{m_{tn}}\{1-(1-x)^i\} \\ &= \frac{1}{m_{tm}}\left\{m_{tn}-(1-x)\frac{1-(1-x)^{m_{tn}}}{x}\right\} \end{aligned} \tag{4}$$

Here, *dno* can be seen to approach 1 as $x\rightarrow 1$. For the specific cases illustrated here, m_{tn} has been taken as 50. Values of *dno* (%) computed for different x (%) values are in Table 6.

Writes being Type A by default do not offer any additional insight and hence are excluded in the sequel. Type A servicing fails even with a single defective server in the first k of the *k-reliable*; it is not resilient to malicious activities

x	0.9	1.8	2.7	3.6	4.5	5.4	6.3	7.2
dno	19.91	34.89	46.27	55.01	61.80	67.15	71.40	74.84

Table 6. dno (%) computed for different x (%) values.

at T2. Figure 10 depicts the variation of Type A servicing for $F(r-u)$ for different minimal sets of k with varying defective fractions. The highlighted lines signify the plot based on the simulation found to be essentially of Type A (calling for lowest effort at the respective quorums). As long as the minimal set size k is greater than 110, this quantum falls down rapidly, even with the increase of defective fractions to 20% (d_1), (the point identified by '*'). Beyond d_1, irrespective of the minimal set size, k the quantum essentially becomes zero, giving way to Types B and C.

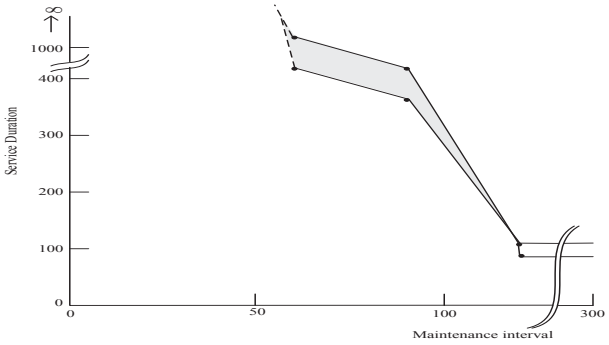


Figure 9. Variation of service duration with *mtn*.

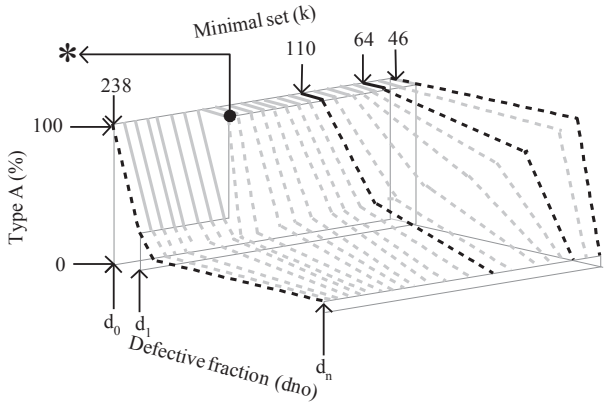


Figure 10. Variation of Type A servicing for varying minimal set (k) and defective fraction (dno).

Figures 11 and 12 depict the various types of servicing for non-concurrent and concurrent modes. They put Type A, B, and C servicing in the proper perspective for the complete $(255, k)$ code set, as discussed below.

In the non-concurrent mode (Figure 11a) Type A servicing drops to zero beyond the range of dno 0% to 35%, confirming the lack resiliency beyond this. Type B servicing fails with greater than $n-k$ defective servers (up till n) in the *k-reliable*; it is highly resilient to malicious activities at T2. Further, in the mid range of dno 46% to 67% Type B servicing dominates, subsequently it tapers, giving way to ascendancy of Type C. Increase in percentage of malicious activities at T2 servers does not obstruct servicing; persistent access is provided even with severe defective fractions with Type C servicing. The results signify provision of persistent and reliable servicing in the proposed DSS-D design.

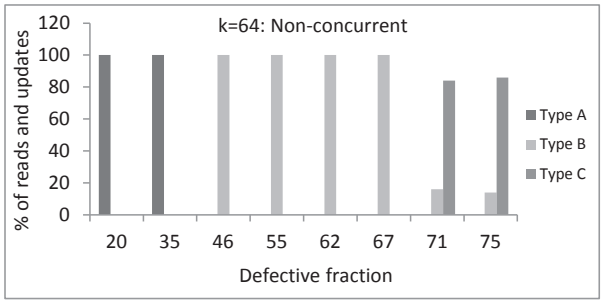


Figure 11a. Code $(255, 64)$ Non-concurrent mode.

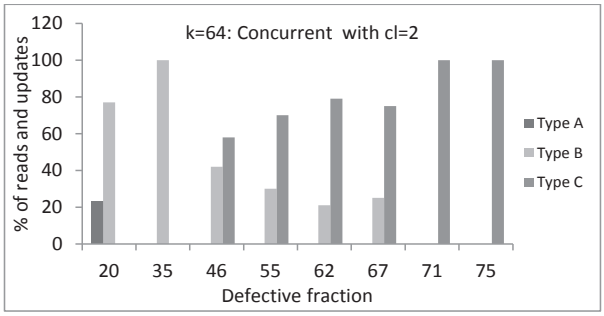


Figure 11b. Code $(255, 64)$ Concurrent mode (2).

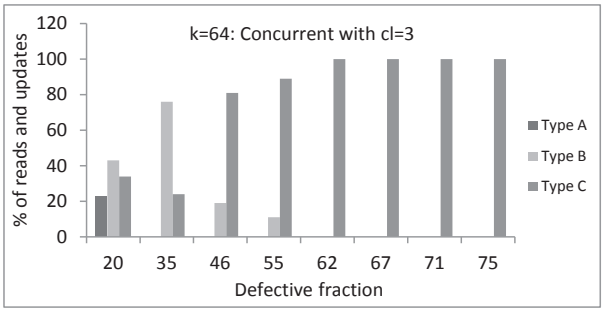


Figure 11c. Code $(255, 64)$ Concurrent mode (3).

A look at the resiliency offered by the proposed design for concurrent servicing is revealing. For $(255, 64)$ code at concurrency level $cl = 2$ Figure 11b, Type A servicing is limited to *dno* in the range 0 to 20%, the rise of Type B at 20 % itself is quite early and extends to 67%, giving way to Type C beyond. For $cl = 3$ Figure 11c, the maximum concurrency level for the code, Type C is present at a very early value *dno* (20%) itself. This trend is in contrast to non-concurrent mode of servicing (Figure 11a), where the dominance of B and C type services takes over at conspicuously higher *dno* values.

Similar trends with other codes (Figure 12) are indicative of the consistency of the behaviour. The results discussed here further emphasise persistent and reliable servicing in the proposed DSS-D design for concurrent servicing as well.

The study and analysis carried out for the code set $(255, k)$ for varying levels of resiliency and concurrency validate availability with persistent access and storage in the proposed design.

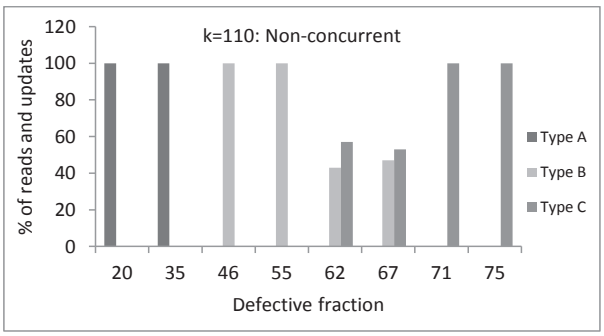


Figure 12a. Code $(255, 110)$ Non-concurrent mode.

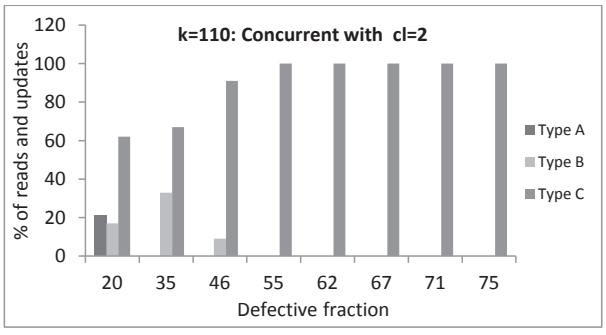


Figure 12b. Code $(255, 110)$ Concurrent mode (2).

i. Malicious activities at T2 do not obstruct availability, persistent access, or storage durability.

ii. Type A servicing is not resilient to malicious activities; it fails even with a single defective server in the first k of the *k-reliable*. Type B servicing, on the other hand, is highly resilient to malicious activities; it fails with greater than $n - k$ defective servers (up to n) in the *k-reliable*. Even in the presence of severe defective fractions persistent access is still guaranteed with Type C servicing.

iii. Types A, B and C in $F(r{-}u{-}w)$ are associated with an increase in complexity in the same order, independent of the response time and communication overheads (see Section 4.1).

iv. The verification of T2 tier correctness in the non-concurrent mode at file reads and updates is equally in attendance in concurrent modes as well. Reads in addition allowed restoration of files shares at the participating errant T2 servers. This reactive strategy at file reads guarantees persistence at the storage.

v. Supply of k correct & up-to-date file shares for file reads and similarly correct & up-to-date version number for file updates is enabled reliably and persistently.

4.2.1. Comparison with Systematic Coding Based Schemes at Storage

System security, system availability, data retrievability, data recoverability and storage durability have a direct effect on continual access guaranteed at storage. Table 7 compares the proposed scheme with those that employ systematic coding among untrusted entities at storage for the above listed criteria. The comparison clearly brings out the effect and control of:

i. data dispersal using non-systematic coding

ii. trusted layer of super servers on persistent access and storage in untrusted environments

I. Scheme using systematic form		II. Scheme using non-systematic form	
A. Security: Resilience to collusion attacks			
Code Level Feature: Encoding	Storage Level Feature: Server discrimination	Code Level Feature: Encoding	Storage Level Feature: Server indiscrimination
Data is encoded as information and parity blocks	The n servers are discriminated as k information servers, storing information blocks and $m\,(=n-k)$ parity severs, storing parity blocks	Data is encoded as mere fragment shares	Servers are not discriminated as information and parity servers; each server stores a unique fragment share
a. Collusion of the k information servers directly yields the original data		a. Collusion of k servers does not directly yield the original data	
b. Distribution security is limited by mere k server collusions		b. Distribution security is not limited by mere k server collusions	
B. Availability: Clients servicing despite failed servers			
Code Level Feature: Redundancy	Storage Level Feature: Fault tolerance	Code Level Feature: Redundancy	Storage Level Feature: Fault tolerance
$n-k$ is the measure of redundant information in the codeword	Redundant servers are $n-k$ in number; Storage system availability is limited to $n-k$ server failures	$n-k$ is the measure of redundant information in the codeword	Redundant servers are $n-k$ in number; Storage system availability is limited to $n-k$ server failures
Storage system is rendered unavailable beyond $n-k$ servers failures			

II. Scheme without trusted layer		II. Scheme with trusted layer	
C. Retrievability: Retrieve data despite corruptions/errors at available servers			
Code Level Feature: Error correction capability	Storage Level Feature: Data retrievability	Code Level Feature: Error correction capability	Storage Level Feature: Data retrievability
Code decoding is limited to either $\lfloor (n-k)/2 \rfloor$ errors or $(n-k)$ erasures	Retrievability is limited to either $\lfloor (n-k)/2 \rfloor$ available but erroneous servers or $(n-k)$ unavailable servers	Code decoding is limited to either $\lfloor (n-k)/2 \rfloor$ errors or $(n-k)$ erasures	Retrievability is not limited to $\lfloor (n-k)/2 \rfloor$ available but erroneous servers; it is limited to only $(n-k)$ unavailable servers
a. Data retrievability depends on the availability of k non-erroneous shares at the available servers		a. Data retrievability is relatively independent of the availability of k non-erroneous shares at available servers; On the unavailability of k non-erroneous shares, correct shares are fetched from the parent server (trusted layer) that originated the latest write	
b. Data retrievability is limited by the unavailability of k non-erroneous servers		b. Data retrievability is not limited by the unavailability of k non-erroneous servers	
D. Recoverability: Restore servers' data share errors / corruptions			
Code Level Feature: Error correction capability	Storage Level Feature: Storage durability	Code Level Feature: Error correction capability	Storage Level Feature: Storage durability
Code decoding is limited to either $\lfloor (n-k)/2 \rfloor$ errors or $(n-k)$ erasures	Durability is limited to either $\lfloor (n-k)/2 \rfloor$ erroneous servers or $(n-k)$ unavailable servers	Code decoding is limited to either $\lfloor (n-k)/2 \rfloor$ errors or $(n-k)$ erasures	Durability is not limited to $\lfloor (n-k)/2 \rfloor$ available but erroneous servers; it is limited only to $(n-k)$ unavailable servers
a. Restoring of shares at erroneous servers depends on the availability of k non-erroneous shares at the available servers		a. Restoring of shares at erroneous servers is relatively independent of the availability of k non-erroneous shares at the available servers; On unavailability of k non-erroneous shares, correct shares are fetched from the parent server (trusted layer) that performed the latest write	
b. There is a very high probability of storage durability eventually deteriorating		b. There is a relatively less probability of storage durability eventually deteriorating	

Table 7. Comparison with systematic coding based storage schemes.

5. Conclusion

A comprehensive design for active DSS-D storages guaranteeing persistent storage and access in untrusted and unreliable environment is in focus. Issues of concurrency, consistency, dynamic data support, along with dynamic identification of errant servers, file share repair, and trust have been addressed to guarantee retrievability and recoverability. The design integrates notification propagation to preserve the same levels of updates, reliability model based on server trust-reputation for selection of reliable servers, and quorum based approach, to reach a consensus on up-to-date file shares for file reads and writes. Significantly, the design does not compel a rather imperceptive and equal treatment of servers and their file shares in unreliable and untrusted environments. Modeling server trust-reputation for use of SDD in erasure coded distributed storages is a significant contribution.

The design permits reads, writes, and updates to be performed persistently and reliably, with only a marginal increase in the associated complexity. Verification of T2 tier correctness in the non-concurrent mode at file reads and updates is equally in attendance in concurrent modes as well. File reads, in addition, allow restoration of file shares at the participating errant T2 servers.

The design guarantees significant resiliency to malicious activities and hybrid failures at storage. Malicious activities at T2 do not obstruct availability, persistent access, or storage durability. Even in the presence of severe defective fractions, persistent access is guaranteed with Type C servicing. The effectiveness of the design in tolerating severe defective fractions at storage has been established – forming a significant contribution of the work. The proposed design can be suitably adapted in typical distributed information storages catering to global networked audience in public, untrusted, and unreliable operating environments.

References

[1] A. ADYA, W. J. BOLOSKY, M. CASTRO, G. CERMAK, R. CHAIKEN, J. R. DOUCEUR, J. HOWELL, J. R. LORCH, M. THEIMER, AND R. P. WATTENHOFER, Farsite: federated, available, and reliable storage for an incompletely trusted environment, *SIGOPS Operating Systems Review* 36, (2002), 1–14.

[2] A. BOUKERCHE, L. XU, K. EL-KHATIB, Trust-based security for wireless ad hoc and sensor networks, *Computer Communications*, 30(11–12) ,(2007), 2413–2427.

[3] A. JUELS, AND B. S. KALISKI JR., PORs: Proofs of Retrievability for Large Files, *CCS '07 ACM*, New York, USA, (2007), 584–597.

[4] B. YAMUNA, AND T. R. PADMANABHAN, A Reliability Level List Based SDD Algorithm for Binary Cyclic Block Codes, *International Journal of Computers, Communications & Control*, 7(2), (2012), 388–395.

[5] C. CACHIN AND S. TESSARO, Asynchronous Verifiable Information Dispersal, *24th Symposium SRD*, (2005),191–202.

[6] C. CACHIN AND S. TESSARO, Optimal resilience for erasure-coded Byzantine distributed storage, *DSN'06*, (2006), 115—124.

[7] C. HUANG, H. HU, Z. WANG, A dynamic trust model based on feedback control mechanism for P2P applications, *Autonomic and Trusted Computing LNCS*, (Springer, Wuhan, China, No. 4158 (2006), 312–321.

[8] C. K. SHYAMALA AND T. R. PADMANABHAN, A trust-reputation model offering data retrievability and correctness in distributed storages, *International Journal of Computers and Applications*, 36(2), 2014.

[9] C. WANG, Q. WANG, K. REN AND W. LOU, Ensuring Data Storage Security in Cloud Computing, *IACR*, Eprint archive 2009/08, (2009).

[10] D. AGRAWAL AND A. EL ABBADI, Integrating security with fault-tolerant distributed databases, *The Computer Journal*, 33(1), (1990), 71–78.

[11] D. L. G. FILHO AND P. S. L. M. BARRETO, Demonstrating data possession and uncheatable data transfer, *IACR*, eArchive 2006/150, (2006).

[12] F. GÓMEZ MÁRMOL AND G. MARTÍNEZ PÉREZ, Towards pre-standardization of trust and reputation models for distributed and heterogeneous systems, *Computer Standards & Interfaces*, Elsevier Science Publishers 32(4), (2010) ,185–196.

[13] G. R. GOODSON, J. J. WYLIE, G. R. GANGER, AND M. K. REITER, Efficient byzantine-tolerant erasure coded storage, *34th International Conference on Dependable Systems and Networks*, (2004), 135–144.

[14] G. ZACHARIA, AND P. MAES, Trust management through reputation mechanisms, *Applied Artificial Intelligence 14*, (2000), 881–907.

[15] H. KRAWCZYK, Distributed fingerprints and secure information dispersal, *Proc. of the* 12^{th} *ACM Symposium on Principles of Distributed Computing* (PODC), (1993), 207–218.

[16] H. LI AND M. SINGHAL, Trust management in distributed systems, *IEEE Computer*, 2, (2007), 45–53.

[17] H. XIA AND A. A. CHIEN, RobuSTore: a distributed storage architecture with robust and high performance, *Supercomputing Conference* – SC , (2007), 44–11.

[18] J. A. GARAY, R. GENNARO, C. JUTLA, AND T. RABIN, Secure distributed storage and retrieval, *Theoretical Computer Science*, 243(1–2), (2000), 363–389.

[19] J. HENDRICKS, G. R. GANGER, AND M. K. REITER, Verifying distributed erasure-coded data, *26th ACM Symposium PODC*, (2007).

[20] J. KUBIATOWICZ, D. BINDEL, Y. CHEN, P. EATON, D. GEELS, R. GUMMADI, S. RHEA, H. WEATHERSPOON, W. WEIMER, C. WELLS AND B. ZHAO, Oceanstore: An architecture for global-scale persistent storage, *Proceedings of ACM ASPLOS*, 35(11), (2000), 190–201.

[21] K. D. BOWERS, A. JUELS, AND A. OPREA, HAIL: A High-Availability and Integrity Layer for Cloud Storage, *CCS '09 ACM*, (2009) New York, USA, 187–198.

[22] K. M. MARTIN, Challenging the adversary model in secret sharing schemes, *Proceedings of the Royal Flemish Academy of Belgium for Science and the Arts Coding and Cryptography II*, (2008), 45–63.

[23] L. GAO, M. DAHLIN, J. ZHENG, L. ALVISI AND A. IYENGAR, Dual-Quorum: A Highly Available and Consistent Replication System for Edge Services, *IEEE Transactions on Dependable and Secure computing*, 7(2), (2010).

[24] L. LAMPORT, Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM*, 21(7), (1978), 558–565.

[25] L. PAMIES-JUAREZ, F. OGGIER AND A. DATTA, Decentralized erasure coding for efficient data archival in distributed storage systems, *Distributed Computing and Networking*, Springer Berlin Heidelberg, (2013), 42–56.

[26] M. A. SHAH, M. BAKER, J. C. MOGUL, AND R. SWAMINATHAN, Auditing to keep online storage services honest source, *USENIX HotOS 11*, (2007).

[27] M. LILLIBRIDGE, S. ELNIKETY, A. BIRRELL, M. BURROWS AND M. ISARD, A cooperative Internet backup scheme, USENIX Annual Technical Conference, *General Track 2003*, (2003), 29–41.

[28] M. PLACEK AND R. BUYYA, *A Taxonomy of Distributed Storage Systems*, The University of Melbourne, (2006).

[29] M. RABIN, Efficient dispersal of information for security, load balancing, and fault tolerance, *JACM*, 36(2), (1989), 335–348.

[30] R. H. MORELOS-ZARAGOSA, *The Art of Error Correcting Coding*, John Wiley and Sons, 2006.

[31] S. KAMVAR, M. SCHLOSSER, H. GARCIA-MOLINA, The Eigen Trust Algorithm for Reputation Management in P2P Networks, *Proc. of the International World Wide Web Conference*, (2003) Budapest, Hungary, 640–651.

[32] S. LIN AND D. J. COSTELLO JR., *Error Control Coding Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 2004.

[33] S. PETER, *Error Control Coding from Theory to Practice*, John Wiley and Sons, 2002.

[34] Y. WANG AND J. VASSILEVA, Trust and Reputation Model in Peer-to-Peer Networks, *Proc. of IEEE Conference on P2P Computing*, Linkoeping, (2003) Sweden, 150–157.

Received: September, 2014
Revised: February, 2015
Accepted: March, 2015

Contact addresses:
C. K. Shyamala
Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham (University)
Coimbatore
India
e-mail: ck_shyamala@cb.amrita.edu

T. R. Padmanabhan
Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham (University)
Coimbatore
India
e-mail: trp@amrita.edu

MS. C. K. SHYAMALA is Assistant Professor in the Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham (University), Coimbatore and has research interests in distributed storages, cryptography and information security. She is currently pursuing her doctoral work in distributed storages. She has 2 years of industrial and 15 years of teaching experience. She is co-author of the book 'Cryptography and Security', (Wiley India) and has publications in reputed journals.

With over 40 years of teaching and research experience, DR. T. R. PADMANABHAN is presently Professor Emeritus in the School of Engineering, Amrita Vishwa Vidyapeetham (University), Coimbatore. He has five books to his credit and has publications in reputed journals. His present areas of interest are cryptography, information security, digital communications, ECC and VLSI. He is a senior member of IEEE, fellow of IE(I), IETE, and CSI.