

# Secure Dynamic Fragment and Replica Allocation in Large-Scale Distributed File Systems

Alessandro Mei, Luigi V. Mancini, and Sushil Jajodia, *Senior Member, IEEE*

**Abstract**—We present a distributed algorithm for file allocation that guarantees high assurance, availability, and scalability in a large distributed file system. The algorithm can use replication and fragmentation schemes to allocate the files over multiple servers. The file confidentiality and integrity are preserved, even in the presence of a successful attack that compromises a subset of the file servers. The algorithm is adaptive in the sense that it changes the file allocation as the read-write patterns and the location of the clients in the network change. We formally prove that, assuming read-write patterns are stable, the algorithm converges toward an optimal file allocation, where optimality is defined as maximizing the file assurance.

**Index Terms**—File system security, replication, fragmentation, distributed systems, peer-to-peer algorithms, assurance.

## 1 INTRODUCTION

THE growth and diffusion of large network infrastructures have led to several projects that aim to design large-scale distributed systems supporting efficient, secure, available, and location-independent network services. Moreover, the number of users that need these services will surely keep growing in the future. In a classical client/server architecture, the centralization of services allows a reasonable management of complex distributed applications, assuring security, availability, and consistency. In the literature, several architectures follow this project baseline using centralized servers, and most distributed file system prototypes rely on the explicit or implicit assumption of one or more “trusted servers” (e.g., NFS [1], [2], AFS [3], and CODA [4]). Unfortunately, the increasing size of modern network infrastructures reveals the limitation of this approach to build a ubiquitous, always available, distributed file system, especially referred to the security, the overall efficiency, the scalability, and the single point of failure problem.

In order to overcome the above limitations, this paper aims at designing a solution based on a large number of servers coordinated by decentralized algorithms that guarantee the availability and scalability of the system’s functionalities. There is no centralized server for file system services, only a set of cooperating nodes to provide data storage, ubiquitous access, and update to remote users in a scalable and dynamically reconfigurable way. Only clients are trusted while all servers are untrusted; this change has a strong impact on security and availability models. Recent research works started at the University of Berkeley (OceanStore [5]) and at Carnegie Mellon University (PASI

[6]), among others, have built prototypes of such a distributed file system architecture. These projects propose solutions of some of the issues that include: distributed file discovery protocols, identification of user profiles and preferences, scalable authentication and access control, and tolerance of network and server failures.

In a fragmentation scheme [7], [8], a file  $f$  is split into  $n$  fragments, all fragments are signed and distributed to  $n$  remote servers, one fragment per server. The user can reconstruct file  $f$  by accessing  $m$  fragments arbitrarily chosen. The algorithm works in the read- $m$ -write-all context. In general, a read of  $m$  fragments is performed from the closest servers among those that store the  $n$  fragments of the file. A write is performed to all the  $n$  servers. When  $m = 1$ , a fragmentation scheme coincides with an  $n$  replication scheme, where  $n$  copies (replicas) of file  $f$  are stored to  $n$  different remote servers. In the following, we will use the term fragment to refer also to a file replica. A large-scale distributed file system usually bases file availability, confidentiality, and integrity on a combination of file fragmentation, file replication, and file encryption techniques. This paper proposes a model to assess the assurance of a file stored in such a system, where the assurance of a file is the probability that the file has not been compromised under the assumption that the system is the target of a successful attack (see Section 3 for a formal definition).

Intuitively, large-scale (also in geographical terms) service distribution and the use of fragmentation and dynamic reconfiguration techniques make a distributed file system much more available compared with traditional client-server architectures. However, the above reconfigurable distributed service presents some critical problems with respect to security. For example, during fragments reallocation (due to demand changes and operative conditions), one or more compromised servers may maliciously try to collect  $m$  fragments of the same sensitive file in order to reconstruct the file content. Our paper considers these issues, among others, and proposes a completely distributed allocation

- A. Mei and L.V. Mancini are with the Dipartimento di Informatica of the Università di Roma “La Sapienza,” Via Salaria 113, 00198 Roma, Italy. E-mail: {mei, mancini}@dsi.uniroma1.it.
- S. Jajodia is with the Center for Secure Information Sciences, George Mason University, Mail Stop 4A4, Fairfax, VA 22030-4444. E-mail: jajodia@gmu.edu.

Manuscript received 17 Aug. 2002; accepted 15 May 2003.  
For information on obtaining reprints of this article, please send e-mail to: tpd@computer.org, and reference IEEECS Log Number 118640.

algorithm that allows each server to store at most one fragment of the same file. In particular, the main contributions of the paper include:

- A model to measure and to compare the assurance of a file stored on distributed servers employing a combination of fragmentation, replication, and encryption techniques. Some of the servers may be compromised by malicious attackers, and file fragments can be eavesdropped when sent over the network.
- An optimal, adaptable, and distributed allocation algorithm to guarantee high file assurance in a dynamic environment, where files move also to follow the change of location of their clients. The fragment allocation is reconfigured transparently and automatically, as a response to modified resource availability and client demands.

Our model confirms certain intuitions analytically: 1) A higher frequency of writes guarantees higher confidentiality, and 2) keeping a fragment close to where it is read and written frequently not only minimizes cost, but also increases confidentiality. Moreover, in contrast with earlier work in the literature, our model assumes the servers may share vulnerabilities and characterizes the common vulnerabilities by saying that any security weakness in a server is present in a fraction of the total number of servers. The distributed algorithm is optimal in the following sense. Consider a file  $f$  possibly shared by a set of users, and consider an arbitrary assignment of the fragments of  $f$  to a set of distributed file servers. Assume the read-write pattern and the location of each user are generally regular. For example, for a given time interval, user U1 executes one read and two writes per second from the location L1, user U2 executes three reads and one write per second from location L2, and so on. Then, our distributed algorithm will converge to the optimal allocation of the fragments for the global read-write pattern. Optimality is defined in terms of maximizing the assurance of file  $f$ . Moreover, the optimal algorithm is completely distributed, since each server makes the decision to locally change the allocation scheme, and it does so based on locally collected statistics.

The rest of the paper is organized as follows: Section 2 introduces the basic schemes of file allocation and the notation used in the paper. Section 3 proposes a model to measure the assurance of static and dynamic file allocations in a distributed system under attack. Section 4 presents our distributed allocation algorithm and proves that, when the global read-write pattern stabilizes, the scheme is optimal for the file assurance. Section 5 contains some concluding remarks.

## 2 SECRET SHARING, REPLICATION, AND RELATED WORK

An  $(m, n)$  secret sharing scheme,  $1 \leq m \leq n$ , breaks a file  $f$  into  $n$  shares  $f_0, \dots, f_{n-1}$  such that any  $m$  of the shares are enough to reconstruct  $f$ , while a subset of fewer than  $m$  shares give no information on  $f$ . We will refer to such a scheme also as an  $(m, n)$  fragmentation scheme, where each share is a *fragment*, and where  $f_0, \dots, f_{n-1}$  is a  $(m, n)$  fragmentation of file  $f$ . A

$(1, n)$  fragmentation scheme can be also seen as a  $n$  replication scheme, where each fragment is a *replica* of file  $f$ . Classical implementation of  $(m, n)$  secret sharing schemes are Blakley's [9] and Shamir's [7], where each share is as big as the original file  $f$ . More space-efficient schemes exist, like *information dispersal* [8], which is space-optimal, and *short secret sharing* [10], which ensures confidentiality against computationally bounded adversaries. However, while secret sharing provides *perfect* confidentiality, meaning that exactly no information is gained by holding less than  $m$  shares, other schemes do not have this property, and leak partial information on the original file  $f$  to holders of less than  $m$  shares. In all of the above schemes, encoding and decoding are computationally efficient.

Secret sharing schemes, sometimes combined with cryptography, have been used to achieve both availability and confidentiality in peer-to-peer distributed file systems. An example of such systems is CFS [11], which provides a read-only storage service with provable guarantees of robustness, efficiency, and load-balancing. Availability is assured by replication, without encryption, with the idea that files can be encrypted by the client before storing when confidentiality is an issue. Farsite [12] is another replication-based distributed file system where PC clients contribute storage resources in exchange of a highly available and reliable file system. Fragments are encrypted before storage using a so-called convergent encryption which allows identical files to be detected with high probability, even with different names and encrypted with different keys. The Intermemory project [13] aims at building a robust storage substrate for collective holdings of libraries and institutions. It uses a two-level dispersal of erasure-resilient fragments with either 32, 1,024, or 65,536 nodes storing each block. OceanStore [5] provides a global persistent storage service that supports updates on widely replicated encrypted data. PASIS [6] considers a wide range of threshold schemes, which are a combination of secret sharing and information dispersal, achieving better confidentiality, availability, and integrity than conventional replication, but at a cost in performance. In PAST [14], replicas are placed on a diverse set of nodes by a fault-tolerant and self-organizing routing and location infrastructure. The set of nodes storing replicas is determined pseudorandomly, and files stored are associated with a quasi-unique *fileId* generated at the time of the file's insertion into the system. Files are immutable and can be shared by distributing the *fileId*.

In the next sections, we measure the assurance of a file  $f$  stored in a distributed file system using a combination of cryptography, secret sharing, and replication, in the face of compromised servers and assuming an insecure network infrastructure, where fragments can be eavesdropped during communications from/to the clients and during server to server dynamic reallocation. The model can be used to evaluate the assurance provided by most of the above distributed file systems, and can be fairly easily adapted to other coding techniques. Moreover, we propose a dynamic allocation algorithm which tends to allocate fragments in such a way to provide high assurance, and,

under proper hypothesis of stability of the system, actually converges to the allocation of maximal assurance.

### 3 DISTRIBUTING FRAGMENTS OVER A NETWORK

Consider a distributed system consisting of a set  $V$  of  $N$  servers. Given an  $(m, n)$  fragmentation  $f_0, \dots, f_{n-1}$  of a file  $f$ , a *mapping* for  $f$  is a function  $\mu : \{f_0, \dots, f_{n-1}\} \mapsto V$  such that no two fragments are mapped to the same server.

Intuitively, mapping fragments  $f_0, \dots, f_{n-1}$  over a distributed system should improve the assurance of file  $f$ . Indeed, any foe willing to get  $f$  must compromise at least  $m$  servers to retrieve enough fragments. However, assessing the assurance of a mapping  $\mu$  is not straightforward. Even under the assumption of knowing how much effort is needed by an active foe to break into a single server, it is still hard to tell how much effort is needed to break into  $m$  servers of the system. As an example, consider an attacker who exploits a software flaw to compromise part of a distributed system. Typically, the biggest effort consists of finding the flaw and breaking into the first server. After that, not much effort is required to compromise all those servers running the same insecure software.

One simple way, used in the literature, to give an estimation of the assurance of a mapping for a fragmented file is to assume that all servers of the system are heterogeneous [15]. Consequently, the effort spent to break into the first server is probably useless when trying to break into other servers of the distributed system. This idea has been used to measure the assurance of a mapping as the effort needed by an active foe to break into  $m$  servers, and the result claimed [15] is that this effort is  $m$  times the effort to break into a single server. However, this approach seems to be feasible only when  $N$ , the number of nodes in the network, is small, since it is not clear how to build a large number of heterogeneous servers. Moreover, according to the above model, the assurance of a mapping does not depend on  $n$ , but only on  $m$ . This would imply that increasing  $n$  for a fixed  $m$  and, thus, increasing redundancy of a file in a distributed system, does not affect the assurance of the file. This appears to be against intuition since it should be easier that an attacker collects  $m$  fragments of the same file  $f$ , when a large number of fragments of the same file are distributed on the system.

In the following section, we propose a simple model to measure the assurance of a mapping  $\mu$  in a static distributed file system. This model takes into account the fact that the security of a single server is not independent from successful attacks to other servers of the distributed system, that both parameters  $n$  and  $m$  affect the assurance of a fragmented file, and that additional cryptography can be used to secure each fragment/replica in order to improve the confidentiality of file  $f$ .

#### 3.1 Static Systems

During a period when neither read nor write operations occur on a file  $f$ , we say that the system is *static*. In this case, which is usually temporary, fragments are just stored, and the only way an attacker has to get  $m$  fragments of file  $f$  is to break into enough servers. We start our discussion by assuming that servers store fragments as plain-text without employing cryptographic coding; hence, the attacker needs

just to break into  $m$  servers each storing a fragment of file  $f$  to compromise the whole file.

##### 3.1.1 Distribution Assurance

In order to break into a server, an attacker could exploit a system weakness. There are several kinds of such weaknesses: poor system administration, bad users' practices (simple to guess passwords, for example), malicious insiders, and different kinds of software/hardware flaws (e.g., buffer overflow attacks). We cannot realistically avoid such weaknesses in any large and complex system and, of course, we do not even know what weaknesses a system could reveal in the future. However, the number of servers subjects to the same particular weakness can be limited. One way is to apply information system diversity building servers, which are as heterogeneous as possible, with different software, different hardware, and different administrators. Indeed, even if the attacker finds a software, hardware, or human flaw, he can exploit it to break into a limited number of servers, namely, those using the same weak component. If this component is used only by a fraction of servers, the potential harm of its weakness is contained. Note that this assumption is weaker than assuming that all servers are heterogeneous, and does not limit in any way the size  $N$  of the system.

Assume that an  $N$  server distributed system is built in such a way that the *same* weakness is present in at most  $\lceil \lambda N \rceil$  servers, where  $\lambda \in \mathbb{R}$  and  $0 < \lambda < 1$ . Then, a single attack by an active foe cannot affect more than  $\lceil \lambda N \rceil$  servers, forcing the foe to find another weakness if willing to compromise other servers. Let  $S$  be the event "the distributed system has been successfully attacked," and let  $Q$  be the event "at most  $m - 1$  servers each storing a fragment of file  $f$  have been compromised," we will measure the *distribution assurance*  $A_\circ(\mu)$  of a mapping  $\mu$  for a file  $f$  as the conditional probability of  $Q$  given  $S$  (i.e., the probability that under a successful attack fewer than  $m$  fragments of file  $f$  have been compromised).

Under these assumptions, when  $m > \lceil \lambda N \rceil$ , the distribution assurance of  $\mu$  is just 1, as it is not possible for the attacker to gain  $m$  fragments of file  $f$  with a single attack. Otherwise, in the more common case when  $m \leq \lceil \lambda N \rceil$ ,  $A_\circ(\mu)$  can be computed in the following way:

$$A_\circ(\mu) = 1 - \sum_{i=m}^n \frac{\binom{\lceil \lambda N \rceil}{i} \binom{\lfloor N - \lambda N \rfloor}{n-i}}{\binom{N}{n}}. \quad (1)$$

*Pure Fragmentation:* An example of this class of file allocation schemes can be found in the PASIS system [6]. With pure fragmentation, as  $m$  increases, Fig. 1 shows the behavior of  $A_\circ(\mu)$  of a file fragmented into 15 pieces and distributed over a 100 server system for different values of  $\lambda$ . We can see that the distribution assurance grows quickly to the maximum value and that systems with higher diversity (small values of  $\lambda$ ) guarantee a higher assurance for a given  $m$ . Clearly, the maximum distribution assurance of a fragmentation scheme is obtained when  $m = n$ , and its value is:

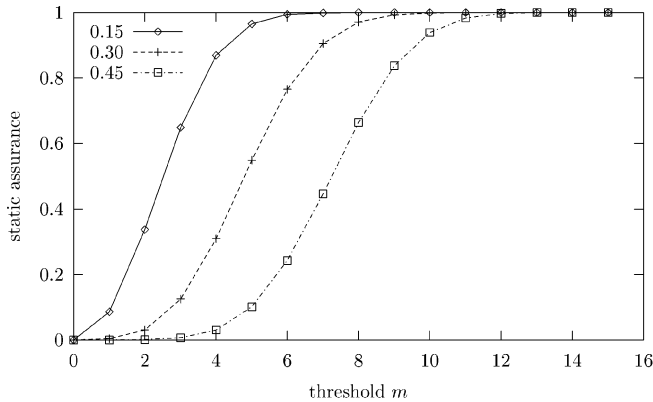


Fig. 1. Distribution assurance of a mapping  $\mu$  such that  $N = 100$ ,  $n = 15$ , and  $\lambda = 0.15, 0.30, 0.45$ .

$$A_{\phi}(\mu) = \begin{cases} 1 & n > \lceil \lambda N \rceil \\ 1 - \frac{\binom{\lceil \lambda N \rceil}{n}}{\binom{N}{n}} & n \leq \lceil \lambda N \rceil. \end{cases} \quad (2)$$

Note that, even when  $n \leq \lceil \lambda N \rceil$ , the distribution assurance is very close to 1 for a large enough  $n$ . For example, when  $N = 100$ ,  $\lambda = .45$ , and  $n = 15$ ,  $A_{\phi}(\mu) \approx 0.99999864$ . Fig. 2 shows how large  $m$  should be, depending on  $n$ , in order to obtain an assurance of 0.9999, 0.999999, and 0.99999999 on a 150 server distributed system. According to intuition, in order to get the same distribution assurance when the number of fragments  $n$  increases, threshold  $m$  should also increase.

These results show that file fragmentation in a large-scale distributed file system is an effective way to guarantee assurance, even in the presence of compromised servers. The problem of choosing a particular value of  $n$  and  $m$  for a file  $f$  is a trade off between assurance and availability. Indeed, increasing  $m$  for a fixed  $n$  improves the assurance of file  $f$  (see Fig. 1), while increasing  $n$  for a fixed  $m$  allows the distributed file system to tolerate a larger number of compromised servers. A deeper analysis of these trade offs is out of the scope of this paper.

*Pure Replication:* Examples of this class of file allocation schemes are AFS [3], CODA [4], and CFS [11]. When  $m = 1$ , a fragmentation scheme is essentially a replication scheme, and each fragment is a replica of file  $f$ . In this case, the distribution assurance of a mapping  $\mu$ ,

$$A_{\phi}(\mu) = \frac{\binom{N-\lambda N}{n}}{\binom{N}{n}}, \quad (3)$$

drops very quickly to zero as  $n$  increases. As an example, if  $N = 100$ ,  $\lambda = .30$ , and  $n = 15$ , then  $A_{\phi}(\mu) \approx 0.003611$ , if  $n = 20$ , then  $A_{\phi}(\mu) \approx 0.000420$ , and if  $n = 25$ , then  $A_{\phi}(\mu) \approx 0.000041$ . This analysis shows that, when  $n$  is large and the distributed file system is the target of a successful attack, almost surely file  $f$  is compromised.

### 3.1.2 Static Assurance

To improve security of the system, especially when a replication scheme is used, fragments (replicas) can be encrypted before being stored. Of course, this encryption must rely on a secret key which is not stored in the server

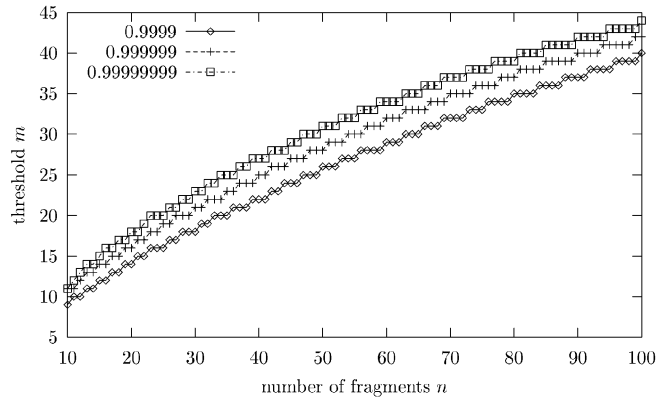


Fig. 2. Value for  $m$  (as a function of  $n$ ) required to obtain a distribution assurance of at least 0.9999, 0.999999, and 0.99999999 on a 150 server distributed system built in such a way that  $\lambda = 0.3$ .

itself. Indeed, if a malicious user breaks into a server, everything in the server is compromised, including the file secret key. Therefore, we assume that each fragment is encrypted by using a key known only by authorized clients, and that this is done by means of an encryption system  $K$ .

In this new setting, the attacker has a more difficult task. First, he has to retrieve  $m$  fragments of file  $f$ , and then he has to break encryption system  $K$ . In the following, we assume that breaking an encryption system is not impossible. Practice says that no encryption system is perfectly safe. As an example, a malicious user might apply a dictionary attack to break  $K$  and find the client's secret key. This is just one among the many attacks to compromise  $K$  and probably not the most common. It might be easier, by using social engineering techniques, to retrieve the clients secret keys, if they do not put enough care in securing them.

We measure the strength of an encryption system  $K$  as its probability of being broken in any possible way, which includes: dictionary attacks, brute force on the user secret key, mathematical crypto-analysis, password guessing, and social engineering techniques. Let  $\mathbb{P}_K$  be such a probability:

$$\mathbb{P}_K = \mathbb{P}(\text{"the encryption system } K \text{ is compromised"}). \quad (4)$$

Let  $S$  be the event "the distributed system has been successfully attacked," and let  $Q'$  be the event "at most  $m - 1$  servers each storing a fragment of file  $f$  have been compromised or the attacker is not able to decrypt the collected fragments"; we will measure the *static assurance*  $A_K(\mu)$  of a mapping  $\mu$  for a file  $f$  using encryption system  $K$  as the conditional probability of  $Q'$  given  $S$  (i.e., the probability that under a successful attack file  $f$  has not been compromised):

$$A_K(\mu) = 1 - \mathbb{P}_K(1 - A_{\phi}(\mu)). \quad (5)$$

*Fragmentation with Encryption:* This scheme can be used to build systems with an extremely high static assurance. Indeed, following the above analysis, pure fragmentation alone is able to guarantee a distribution assurance as close to 1 as desired. Adding cryptography to pure fragmentation gives an independent degree of security yielding extremely high values of static assurance. However, since data are encrypted, file sharing among a dynamic subset of users

can be harder to implement in fragmentation with encryption than in pure fragmentation.

*Replication with Encryption:* Considering that file replication alone gives very low static assurance when the number of replicas increases, many research projects of large-scale distributed file systems propose to encrypt files before storing them. Examples of such systems include OceanStore [5] and FarSite [12]. However, under the assumption that for any encryption system  $K$ ,  $P_K$  cannot be made as close to zero as desired (there is always a nonzero probability to compromise the keys), a replication with encryption scheme does not allow us to approximate at will a static assurance of 1, while a fragmentation scheme guarantees such a property. Indeed, the maximum static assurance a replication scheme can achieve is equal to

$$A_K(\mu) = 1 - \mathbb{P}_K \left[ \frac{\lambda N}{N} \right], \quad (6)$$

reached when  $n = 1$ , which does not approximate 1 for a fixed number of servers. Note that  $P_K$  cannot be made as close to zero as desired by simply taking longer and longer keys. Consider this scenario: By accident, a legitimate user loses the file secret key which is found by an attacker. While with a fragmented distributed file the attacker still has to retrieve  $m$  fragments, with a replicated distributed file the attacker's task is simplified, as shown by our study. The strength of encryption system  $K$  depends on this kind of scenario, and this might make it impossible to get  $P_K$  as close to zero as required.

### 3.1.3 Integrity

So far, we have focused on confidentiality. Another important issue is assuring integrity of file  $f$ . A possible attack by a single compromised server storing one of the fragments of file  $f$  is to modify in some way the fragment. When a client reads that file, it collects  $m$  fragments and reconstructs the file. However, since one of the  $n$  fragments is modified, the reconstruction might generate a file which is different from the original file  $f$ . Moreover, if afterward the same client performs a write operation, all fragments will be rewritten with incorrect data, and the final result is that file  $f$  is lost. This attack is particularly vicious since a single compromised server is enough to destroy the whole file  $f$ . To prevent this unfortunate scenario from happening, any malicious modification of a fragment should be detected. This is possible by computing a hash-digest for each fragment and storing the whole list of  $n$  hash-digests with each fragment. Alternatively, each fragment can be signed by the client, so that any modification of a fragment is equivalent to its destruction.

As far as  $n - m$  fragments at most of file  $f$  are compromised, the integrity of  $f$  is assured. Indeed, even if all compromised fragments are destroyed by the attacker,  $m$  of them are still in the system, enough to reconstruct the file. Consequently, by taking as threshold  $m' = n - m + 1$  in the place of  $m$ , the distribution assurance  $A_o(\mu)$  measures how the system guarantees the integrity of file  $f$ . Note that static assurance, which assumes that fragments are encrypted, cannot be used in this setting since cryptography does not give any contribution to preventing the destruction of fragments.

## 3.2 Dynamic Systems

In a distributed file system, the fragments of a file  $f$  are sent over the network as a consequence of the read and write operations invoked on  $f$  by the legitimate clients. Moreover, dynamic allocation protocols might move the fragments of  $f$  in order to optimize the system performance, for example, by moving them on the less loaded servers. During the periods when fragments are sent over the network, we say that a distributed file system is *dynamic*.

A dynamic file system is less secure than a static one. Indeed, file fragments move through a network that must be considered untrusted, in general. Data can be eavesdropped, destroyed, or modified by attackers who do not need to break into any server of the system, but simply wait for data to pass through a link or a router of the network.

Static assurance does not take into account the risk involved in the fragment movements. Therefore, we extend our model in such a way to measure the assurance in a dynamic system, where security leaks can occur when data is sent over an insecure network. The topology of the network is fundamental to understand and evaluate how risky it is to send a fragment from one particular server to another of the distributed file system, hence, we start our discussion from the network model.

### 3.2.1 The Network Model

In most cases, it is unreasonable to assume that we can choose the network topology upon which a large-scale distributed file system is implemented. Even though we can build our local area networks according to security criteria, or require them to comply with such criteria, they are interconnected through a large-scale interconnection network which is probably out of our control and whose structure have to be considered unknown.

We model the network as composed of  $k$  subnetworks  $S_0, \dots, S_{k-1}$  (e.g., local area networks) interconnected by a larger network (e.g., a wide area network). Each subnetwork consists of a number of servers and clients of the distributed file system, and can be connected to the larger interconnection network at multiple points.

Consider a file  $f$  fragmented among servers of the distributed file system according to a mapping  $\mu$ . When a client performs a read operation, the client has to retrieve at least  $m$  fragments of file  $f$ . Some of the fragments may be stored in the same local subnetwork, while the remaining fragments must be obtained from servers on other subnetworks (see Fig. 3). In both cases, moving fragments can be intercepted by attackers. Consider the first case: A fragment is sent within the same subnetwork. Since we have control on the structure of the subnetwork, the risk of eavesdropping can be made relatively small. For example, one could build every subnetwork according to a star topology where every client machine and every server machine are on different links. In this way, a compromised server  $CS$  in the local subnetwork cannot "sniff" packets exchanged among other servers or clients of the same subnetwork, unless the packet is explicitly sent to  $CS$ . Hence, the probability that an attacker intercepts a fragment within such a subnetwork can be considered negligible, and the network traffic within the same subnetwork can be considered secure with respect to confidentiality. Of course, the security of the central switch of the star topology is now critical. However, this device can be made resistant to remote attacks, for example,

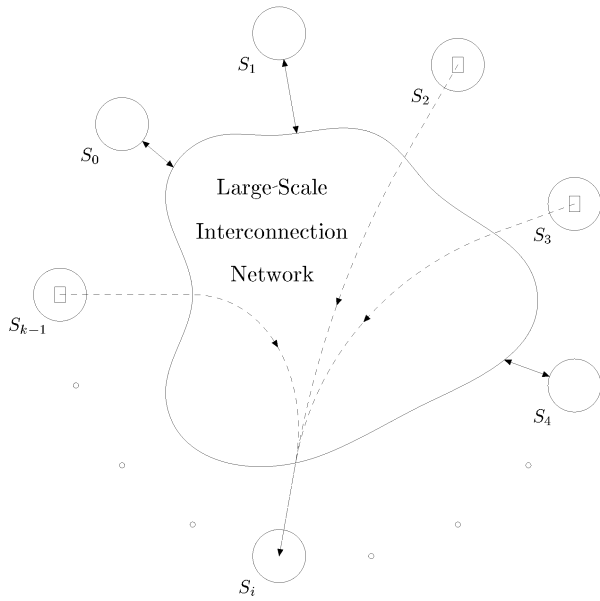


Fig. 3. A client in subnetwork  $S_i$  has performed a read operation. Since only  $m - 3$  fragments are stored in  $S_i$ , three additional fragments are read from  $S_2, S_3$ , and  $S_{k-1}$ .

by forbidding any kind of remote access (even for maintenance) to the device. All network administration and configuration should be executed using a direct local login on the device. Thus, as a first approximation, we consider trusted the central switch of a local area network.

The second case presents a different issue: The large-scale interconnection network is out of our control and cannot be assumed trusted. In the real case, this interconnection network might be the Internet, which is undoubtedly insecure. Every fragment sent across the interconnection network risks being eavesdropped, corrupted, or destroyed. Thus, it is important to measure the amount of file fragments that a distributed file system sends across the large-scale interconnection network to provide its functionalities, e.g., to satisfy the read/write operation invoked by the users. As discussed in the following, this measure can strongly influence the overall assurance of the file.

### 3.2.2 Dynamic Distribution Assurance

In order to analyze the assurance of a dynamic system, it is useful to introduce a definition of file distribution. Given an  $(m, n)$  fragmentation  $f_0, \dots, f_{n-1}$  of a file  $f$ , given a distributed system consisting of  $k$  subnetworks  $S_0, \dots, S_{k-1}$  interconnected by a larger network, and given a mapping  $\mu$  for file  $f$ , the *distribution* of file  $f$  according to  $\mu$  is the function  $d: \{0, \dots, k-1\} \rightarrow \{0, \dots, n\}$  mapping each subnetwork index  $i$  to the cardinality of the set of fragments of file  $f$  stored in the servers of subnetwork  $S_i$  according to  $\mu$ . A distribution encapsulates less information than a mapping; however, for our discussion, it is enough to know how many fragments are stored in subnetwork  $S_i$ , not which server inside  $S_i$  actually stores each of the  $d(i)$  fragments. In the following, we will denote with  $d$  the *distribution* of file  $f$  according to  $\mu$ .

When a client in  $S_i$  performs a read operation on file  $f$ ,  $d(i)$  fragments can be safely retrieved inside the same local network. If  $d(i) < m$ ,  $m - d(i)$  fragments must be requested to servers outside subnetwork  $S_i$ , traversing the insecure

large-scale interconnection network. As an example, in Fig. 3, a client in subnetwork  $S_i$  is performing a read operation. All fragments in subnetwork  $S_i$  are not enough to reconstruct  $f$ , and three fragments have to be retrieved from subnetworks  $S_2, S_3$ , and  $S_{k-1}$ , respectively. We assume that, while moving in the interconnection network, each of the three fragments can be intercepted by a malicious user independently with probability  $\pi$ . Therefore, if  $\rho$  fragments move through the insecure network, the probability that  $j$  of them are captured is

$$\alpha_\rho(j) = \begin{cases} \binom{\rho}{j} \pi^j (1 - \pi)^{\rho-j} & j \leq \rho, \\ 0 & j > \rho. \end{cases}$$

Assume that, during a unit of time, only one operation on file  $f$  is requested in the distributed file system. This operation comes from a client in subnetwork  $S_i$ , which stores  $d(i)$  fragments of  $f$ . Let  $\rho$  be the number of additional fragments needed to serve the operation, namely, either  $\rho = m - d(i)$  if  $d(i) < m$ , or  $\rho = 0$ , otherwise, in case of a read, and  $\rho = n - d(i)$  in case of a write. Therefore, an attacker willing to compromise file  $f$  in this unit of time has to retrieve enough fragments either breaking into servers, or intercepting part of the  $\rho$  moving fragments. Let  $S$  be the event "the distributed system has been successfully attacked," and let  $Q''$  be the event "at most  $m - 1$  fragments of file  $f$  are compromised in a unit of time"; we will measure the *dynamic distribution assurance*  $Q_o^i(\mu)$  related to a *single operation from subnetwork*  $S_i$  of a mapping  $\mu$  for a file  $f$  as the conditional probability of  $Q''$  given  $S$  (i.e., the probability that under a successful attack fewer than  $m$  fragments of file  $f$  have been compromised in a unit of time):

$$Q_o^i(\mu) = 1 - \left( \sum_{i=m}^n \beta(i) + \sum_{i=0}^{m-1} \beta(i) \sum_{j=m-i}^{\rho} \alpha_\rho(j) \right),$$

where

$$\beta(i) = \frac{\binom{[ \lambda N ]}{i} \binom{[ N - \lambda N ]}{n-i}}{\binom{N}{n}} \quad (7)$$

is the probability that exactly  $i$  fragments are stored in the compromised servers. Note that we assume the worst case when all moving fragments come from not compromised servers, and are thus useful for rebuilding  $f$  if intercepted.

This analysis holds for a single read or write operation. However, in general, there are multiple reads and writes on file  $f$  coming from many subnetworks in a unit of time. Let  $r_i^t$  and  $w_i^t$  be the number of read and write operations on file  $f$  requested in the  $t$ th unit of time by clients in subnetwork  $S_i$ , where  $t$  is a nonnegative integer. The sequence  $(r_i^0, w_i^0), (r_i^1, w_i^1), \dots$  is the *read-write frequency pattern* of file  $f$  from subnetwork  $S_i$ . A pattern is said to be *stable* if  $(r_i^t, w_i^t)$  does not depend on  $t$ . When the read-write frequency pattern from subnetwork  $S_i$  is stable, and when no ambiguity is possible, we will denote it with  $(r_i, w_i)$ .

A number of fragments move through the insecure interconnection network to serve all read and write operations, other fragments might be moved by dynamic allocation protocols. However, an attacker cannot just intercept  $m$  of those fragments to compromise  $f$ . Indeed, after each write, the version number of fragments of file  $f$  is

increased. A key observation is that fragments with different version numbers cannot be used together to reconstruct  $f$ . Therefore, we are interested in estimating in average how many fragments with the *same* version number are sent over the insecure network in a unit of time.

Assume at least one write operation during the  $t$ th unit of time. Let  $\delta_t$  be the number of fragments moved as a consequence of a dynamic reallocation in the  $t$ th unit of time, then, the average number of moving fragments with the same version number is

$$\epsilon_t(d) = \frac{\sum_{s=0}^{k-1} r_s^t(m \ominus d(s))}{\sum_{s=0}^{k-1} w_s^t} + \frac{\sum_{s=0}^{k-1} w_s^t(m \ominus d(s))}{\sum_{s=0}^{k-1} w_s^t} + \frac{\delta_t}{\sum_{s=0}^{k-1} w_s^t}, \quad (8)$$

where

$$a \ominus b = \begin{cases} a - b & \text{if } a > b, \\ 0 & \text{otherwise.} \end{cases}$$

The first, second, and third part of the sum in (8) counts how many fragments move in average due to read operations, write operations, and reallocation between two consecutive writes, respectively. Note that the above value for  $\epsilon_t(d)$  is an upper bound of the actual value, since the same fragment can be moved multiple times to serve different read operations.

Fragments moved by a dynamic allocation protocol cause a subtle security leak. Not only can they be intercepted, there is a probability that they are moved from an uncompromised server to a compromised one, giving an additional chance to the attacker of collecting  $m$  fragments of file  $f$ . Taking into account these movements, the probability  $\beta^*(i)$  that exactly  $i$  fragments are stored in compromised servers during the time lapsed between two consecutive writes in the  $t$ th unit of time comes to

$$\beta^*(i) = \frac{\binom{\lceil \lambda N \rceil}{i} \binom{\lfloor N - \lambda N \rfloor}{n-i+\delta_t/W_t}}{\binom{N}{n+\delta_t/W_t}}, \quad (9)$$

where  $W_t = \sum_{s=0}^{k-1} w_s^t$ .

Now, we can measure the *dynamic distribution assurance*  $A_o^*(\mu)$  of a mapping  $\mu$  for a file  $f$  at time  $t$  as the dynamic distribution assurance of a single access to file  $f$  requiring  $\epsilon_t(d)$  fragments outside its local subnetwork:

$$A_o^*(\mu) = 1 - \left( \sum_{i=m}^n \beta^*(i) + \sum_{i=0}^{m-1} \beta^*(i) \sum_{j=m-i}^{\epsilon_t(d)} \alpha_{\epsilon_t(d)}(j) \right).$$

The average number of transmitted fragments  $\epsilon_t(d)$  with the same version number is an important component of the dynamic distribution assurance. In general, it is possible to show that the file assurance decreases when the number of write operations is negligible compared with the number of read operations.

*Fact 1:*

$$\lim_{\epsilon_t(d) \rightarrow +\infty} A_o^*(\mu) = 0.$$

This fact suggests that a high assurance levels for read-only files are difficult to guarantee, and that for read/write files,

the clients should perform on average a write operation every  $x$  read operations, where  $x$  is a function of the minimal assurance desired for file  $f$ .

### 3.2.3 Dynamic Assurance

Like in the static case, fragments (replica) can be encrypted with the client's private key before being stored. This is the choice of several research prototypes of secure large-scale distributed file systems using replication like OceanStore and FarSite. Our model, even in the simplified static case, shows that, when using replication schemes, this choice is necessary if you want your system to be reasonably safe against successful attacks, as discussed in Section 3.1. Moreover, when a fragmentation scheme is used, the same discussion shows that very high levels of static assurance can be achieved without encryption. In this latter case, it can be a good idea to encrypt fragments only when they are sent over the network, if we believe that the risk of being intercepted is high. Note that the access to a shared file can be simplified if the file fragments are not kept encrypted in the server, but they are encrypted only before they are sent. Moreover, implementing a so-called *recovery* functionality, which allows to recover the file in case the client lose his keys, is clearly straightforward.

To be more general, our model considers two different encryption systems,  $K_1$  and  $K_2$ .  $K_1$  is used to encrypt file fragments when they are stored; a second encryption using  $K_2$  is performed on the same fragments when sent over the network. Like in the static case, we assume to know probability  $\mathbb{P}$  that  $K_1$  is broken by an attacker, and probability  $\mathbb{P}_{K_2|K_1}$  that  $K_2$  is broken given that  $K_1$  is broken. For example, OceanStore encrypts replicas only when stored, therefore,  $\mathbb{P}_{K_1}$  depends on the strength of such an encryption, while  $\mathbb{P}_{K_2|K_1}$  is equal to 1 since  $K_2$  is "no encryption" and is thus breakable with probability 1.

The *dynamic assurance*  $A_{K_1, K_2}^*(\mu)$  of a mapping  $\mu$  for a file  $f$  at time  $t$ , using encryption systems  $K_1$  to store and  $K_2$  to transmit the file fragments is equal to

$$A_{K_1, K_2}^*(\mu) = 1 - \mathbb{P}_{K_1} \left( \sum_{i=m}^n \beta^*(i) + \mathbb{P}_{K_2|K_1} \sum_{i=0}^{m-1} \beta^*(i) \sum_{j=m-i}^{\epsilon_t(d)} \alpha_{\epsilon_t(d)}(j) \right).$$

The above equation can be derived by noting that fragments found in compromised servers can be used after breaking  $K_1$ , while fragments intercepted over the network can only be used after breaking both  $K_1$  and  $K_2$ .

The dynamic assurance is a consistent extension of its static version. Indeed, when  $\epsilon_t(d) = 0$ , the two measures coincides. In general, it is easy to see the following fact.

*Fact 2:*  $A_{K_1, K_2}^*(\mu) \leq A_{K_1}(\mu)$ .

The above fact captures the intuitive idea that a dynamic system is always less secure than a static one, whatever encryption is used for communications.

A natural question is whether it is possible to find a mapping  $\mu$  such that  $A_{K_1, K_2}^*(\mu)$  is maximal. The answer is yes and it can be shown that the maximum of  $A_{K_1, K_2}^*(\mu)$  coincides with the minimum of the following cost function:

$$\text{cost}(d) = \sum_i [r_i(m \ominus d(i)) + w_i(n - d(i))]. \quad (10)$$

This is somewhat intuitive, noting that  $\text{cost}(d)$  measures the number of fragments that have to move through the insecure interconnection network in order to serve all read and write operations in a unit of time. When this number is minimal, the attacker has less chance to intercept enough fragments to compromise file  $f$ , and the dynamic assurance is thus maximal.

However, the read-write frequency patterns of file  $f$  is not known in advance, and it can change over time, so that an optimal static allocation is not feasible. To overcome this problem, in Section 4, we propose a distributed algorithm for file fragment dynamic allocation such that, if the read-write patterns are stable for a certain interval, then, starting from an arbitrary mapping, fragments are moved among the servers in such a way to converge to a mapping  $\mu$  of maximal dynamic assurance.

### 3.2.4 Other Attacks to a Dynamic System

Consider, as an example, the following scenario: A malicious user has compromised one server and has been able to get a fragment of a file  $f$  stored in this server. Due to the dynamic reallocation of file fragments, even if the malicious user is not able to compromise other servers, it is still possible for him to compromise file  $f$ . He can just wait for enough fragments to step through the compromised server, and collect all of them.

In order to overcome this problem, we propose to add a restriction to the fragments movement. The rule we introduce is: *No fragment can move to a server that currently stores, or has stored even for a short time in the past, a different fragment of the same file.* Enforcing this rule as it is can be expensive in terms of resources to be used. Indeed, it is necessary to record what fragments have been stored in each server since the system has been started. Moreover, a server cannot be trusted for storing this information for itself. Otherwise, a malicious server could pretend that it never stored any fragment of  $f$  in the past, just to receive other fragments to collect.

To prevent security holes of such a logging system, we propose a simple, static, and distributed criterion to decide whether a particular server is authorized to store a particular fragment of file  $f$  or not. Let  $f$  be a file split into  $n$  fragments  $f_0, \dots, f_{n-1}$ , and let  $i$  be the index uniquely identifying a server. Server  $s_i$  is authorized to store fragment  $f_j$  of file  $f$  if and only if:

$$F(i) \equiv j \pmod{n}, \quad (11)$$

where  $F$  is a hashing function mapping the set of server indexes into  $\{0, \dots, n-1\}$ . In this way, server  $s_i$  is authorized to store only one particular fragment of file  $f$ , and the other noncompromised servers will never send to server  $s_i$  a different file fragment. Moreover, as long as  $F$  is chosen uniformly at random from a universal hash family [16], the probability that two servers can store the same fragment is less than or equal to  $1/n$ . This is useful to ensure that, with high probability, every client is close to a server that can store fragment  $f_j$ , for all  $j$ .

By using the above idea, the set of possible servers for each fragment gets smaller and the sets of servers of two different fragments do not overlap, consequently, (1) becomes slightly different:

$$A_\phi(\mu) = 1 - \sum_{i=m}^n \left( \frac{\lceil \lambda N \rceil}{N} \right)^i. \quad (12)$$

Accordingly, (7) and (9) should also be changed in a similar way. However, note that (1) and (12) are essentially the same when the number of servers in the system  $N$  is large compared with the number of fragments  $n$ .

## 4 DYNAMIC ALLOCATION ALGORITHM

In this section, we propose a dynamic allocation algorithm which, assuming stable read-write frequency patterns, moves fragments between servers in such a way to converge to a mapping with maximal dynamic assurance. This is proven by showing that the algorithm converges to a distribution which minimizes cost function (10). The algorithm is distributed and is described in a C-like language in Figs. 4 and 5. Initially, fragments are stored in the system according to an arbitrary mapping compliant with (11).

Consider a client in subnetwork  $S_i$ . When the client performs a read operation, it executes the following algorithm. If  $m$  fragments of file  $f$  can be found in subnetwork  $S_i$ , it retrieves  $m$  fragments choosing those with minimal index. In this way, if multiple clients are present in the subnetwork and more than  $m$  fragments are locally stored, the exceeding fragments are not read by any client and are free to be moved away if necessary. Otherwise, if less than  $m$  fragments can be found, other fragments are requested outside the local network choosing the least recently requested fragments by this client. In this way, each client cyclically requests all fragments stored outside its local network. When a client performs a write operation, he rewrites all fragments.

When a client  $c$  in  $S_i$  performs a read operation and addresses a fragment  $f_r$  stored in subnetwork  $S_j$ ,  $c$  attaches to its request some information which identifies itself, the index  $i$  of the subnetwork, and its updated average read and write frequencies  $\langle r, w \rangle$  for fragment  $f_r$ . Frequency  $r$  is summed up by the server in  $S_j$  storing  $f_r$  for all clients in  $S_i$  that perform read operations on  $f_r$ , and the resulting value is stored into a local server variable  $\tilde{r}_i$ . Similarly, local server variable  $\tilde{w}_i$  is updated using write frequency  $w$ .

During the execution of the algorithm, it may happen that a fragment  $f_r$  in subnetwork  $S_j$  is not read any more by a particular subnetwork  $S_i$ . This is due to:

1. all clients in subnetwork  $S_i$  do not perform read operations any more,
2.  $i \neq j$  and the remote subnetwork  $S_i$  does not retrieve remote fragments any more when reading since it stores  $m$  fragments locally, or
3.  $i = j$  and local clients can find  $m$  local fragments in the subnetwork with smaller index.

In all of the above cases, frequencies  $\langle \tilde{r}_i, \tilde{w}_i \rangle$  store an out of date value that no longer will be updated since the server will not receive any request from subnetwork  $S_i$ , giving new frequency information. In order to deal with this situation, all local frequency variables are periodically reset to zero. The server performs such a reset by executing function `Server::move(fragment  $f_r$ )` every  $\Delta t$  units of time. The `move()` function also checks whether the fragment  $f_r$  has to be moved to a subnetwork performing

```

file Client::read(file  $f$ , frequencies  $\langle r, w \rangle$ ) {
    %  $f$  is the file to read
    %  $r$  is the read frequency of this client
    %  $w$  is the write frequency of this client

    Let  $I = \{f_{i_1}, \dots, f_{i_l}\}$  be the set of indexes of all fragments stored
        in the local sub-network,  $l = |I|$ ;
    if ( $l \geq m$ ) Let  $J = \{f_{i_1}, \dots, f_{i_m}\} \subseteq I$ ;
    else {
        Let  $X$  be such that  $|X| = m - l$  and  $X$  contains the indexes
            of the least recently requested fragments stored outside
            the local sub-network;
        Let  $J = I \cup X$ ;
    }
    Request each fragment in  $J$ , attaching to the request frequencies  $\langle r, w \rangle$ ;
    Check client signatures and remove from  $J$  the indexes of corrupted fragments;
    while ( $|J| < m$  and there are other available fragments to request) {
        Let  $X$  be such that  $|X| = m - |J|$  and  $X$  contains the indexes
            of the least recently requested fragments stored outside
            the local sub-network that have not been read yet during
            this operation;
        Request each fragment in  $X$ , attaching to the request frequencies  $\langle r, w \rangle$ ;
        Check client signatures and remove from  $X$  indexes of corrupted fragments;
        Let  $J$  be equal to  $I \cup X$ ;
    }
    if ( $|J| = m$ ) {
        Reconstruct file  $f$ ;
        return  $f$ ;
    }
    else failure("file unavailable");
}

void Client::write(file  $f$ , data  $g$ , frequencies  $\langle r, w \rangle$ ) {
    %  $f$  is the file to rewrite with data  $g$ 
    %  $r$  is the read frequency of this client
    %  $w$  is the write frequency of this client

    Split  $g$  into  $n$  fragments  $g_0, \dots, g_{n-1}$ ;
    Sign each fragment  $g_0, \dots, g_{n-1}$  with client's private key;
    Rewrite all fragments of  $f$  with  $g_0, \dots, g_{n-1}$ , attaching to the request frequencies  $\langle r, w \rangle$ ;
}

```

Fig. 4. Algorithm executed by a client either to read or to write a given file  $f$ .

a higher number of operations on file  $f$ . In the rest of this paper, we will denote the time between two consecutive resets as a  $\Delta t$ -period.

If a fragment has to be moved from subnetwork  $S_j$  to subnetwork  $S_{j'}$ , the server storing the fragment chooses at random a server in  $S_{j'}$  that can store the same fragment index according to (11). If subnetwork  $S_{j'}$  is large enough, almost surely such a server exists, due to the properties of the hash function  $F$  introduced in (11). Moreover, when multiple eligible servers can be found, randomness ensures a good level of load-balancing. This is performed in procedure  $\text{moveto}(f_r, j')$  in Fig. 5. Conversely, when  $S_{j'}$  does not contain any eligible server, subnetwork  $S_{j'}$  has to be ignored when considering this fragment movement. Provided that subnetworks are large enough, this event is extremely unlikely, therefore, in the following, we assume that all subnetworks can potentially store any fragment.

In the next section, we show that the above algorithm is optimal, in the sense that it converges to an optimal file allocation according to the cost function (10).

#### 4.1 Optimality

Assuming stable read-write frequency patterns  $(r_i, w_i)$ ,  $i = 0, \dots, k-1$ , consider the problem of finding a mapping

for file  $f$  with minimal cost according to (10), that is, a mapping where the file  $f$  has the maximal assurance. For the ease of explanation, we will assume at least one active client, i.e., there exists an index  $i$  such that  $r_i + w_i > 0$  for given file  $f$ . Moreover, we can assume, without loss of generality, that subnetworks are sorted in such a way that if  $i < j$ , then  $(r_i + w_i) \geq (r_j + w_j)$ .

Of course, some subnetworks perform more read and write operations than others. In particular, some of the subnetworks have a key property: The number of read and write operations per unit of time on file  $f$  is at least as high as the number of write operations on  $f$  from any other subnetwork. The indexes of such subnetworks can be collected in a set  $H$  defined as follows:

$$H = \{i : w_j \leq r_i + w_i \forall j\}.$$

A few simple facts about  $H$  can be shown. First of all,  $H$  is not empty. This is true because  $(r_0 + w_0) \geq (r_i + w_i) \geq w_i$  for all  $i$ , hence,  $0 \in H$ . Second, since  $(r_i + w_i) \geq (r_{i+1} + w_{i+1})$ , if  $i+1 \in H$ , then  $i \in H$ . In other words,  $H$  is equal to an interval  $\{0, \dots, h-1\}$ , where  $h$  is the number of elements of  $H$ . Consequently,  $\overline{H}$  is equal to interval  $\{h, \dots, k-1\}$ . In the following, we will say "subnetwork  $S$  is in  $H$  (or  $\overline{H}$ )," meaning "the index of subnetwork  $S$  is in  $H$  (or  $\overline{H}$ )."

```

Server::onReadInvocation(client c, file f, frequencies (r,w)) {
  Let j be the sub-network where client c is locally connected;
  Update frequency  $\tilde{r}_j$  with r and frequency  $\tilde{w}_j$  with w;
  if (fragment has been moved) failure("fragment not present");
  else send(fragment, c);
}

Server::onWriteInvocation(client c, file f, fragment g_r, frequencies (r,w)) {
  Let j be the sub-network where client c is locally connected;
  Update frequency  $\tilde{r}_j$  with r and frequency  $\tilde{w}_j$  with w;
  if (fragment has moved) failure("fragment not present");
  else Update fragment with g_r;
}

Server::move(fragment f_r) {
  Let j be the index of the sub-network where the server is
  locally connected;
  if ( $\exists j'$  such that  $\tilde{r}_{j'} + \tilde{w}_{j'} > \tilde{r}_j + \tilde{w}_j$ , and  $\tilde{r}_{j'} + \tilde{w}_{j'} > \tilde{r}_{j''} + \tilde{w}_{j''}$  for all  $j''$ )
  moveto(f, j');
  reset  $\tilde{r}_0, \dots, \tilde{r}_{k-1}$  and  $\tilde{w}_0, \dots, \tilde{w}_{k-1}$  to zero;
}

```

Fig. 5. Algorithm executed by servers to decide whether to move a fragment to another subnetwork.

Now, we can formally show that no locally optimal distribution can assign any fragment to a subnetwork  $S_i$  in  $\overline{H}$ .

**Lemma 1.** *If  $d$  is a locally optimal distribution for file  $f$ , then  $d(i) = 0$  for all subnetworks  $S_i$ ,  $i \in \overline{H}$ .*

**Proof.** Let  $d$  be a locally optimal distribution such that  $d(i) > 0$ , where  $i \in \overline{H}$ . Consider the distribution  $d'$  obtained from  $d$  by moving one fragment from  $S_i$  to  $S_j$ , where  $j$  is such that  $w_j > r_i + w_i$ . Note that such an index  $j$  always exists since  $i \in \overline{H}$ . Let  $\Delta$  be equal to  $\text{cost}(d) - \text{cost}(d')$ , which is the improvement of  $d'$  over  $d$ .

$$\begin{aligned}
\Delta &= r_j(m \ominus d(j)) + w_j(n - d(j)) + r_i(m \ominus d(i)) + \\
&\quad + w_i(n - d(i)) - r_j(m \ominus (d(j) + 1)) + \\
&\quad - w_j(n - (d(j) + 1)) - r_i(m \ominus (d(i) - 1)) + \\
&\quad - w_i(n - (d(i) - 1)) = \\
&= [(m \ominus d(j)) - (m \ominus (d(j) + 1))]r_j + w_j + \\
&\quad - [(m \ominus (d(i) - 1)) - (m \ominus d(i))]r_i - w_i = \\
&= b_0 r_j + w_j - b_1 r_i - w_i,
\end{aligned}$$

where  $b_0$  is a bit set to one if and only if  $d(j) < m$ , and  $b_1$  is a bit set to one if and only if  $d(i) \leq m$ . Hence,

$$\Delta = b_0 r_j + w_j - b_1 r_i - w_i \geq w_j - (r_i + w_i) > 0.$$

We get a strictly positive improvement by moving one fragment from  $S_i$  to  $S_j$ . Consequently,  $d$  cannot be locally optimal.  $\square$

Thanks to the previous lemma, we can concentrate on the allocation of fragments of  $f$  to subnetworks in  $H$ . Two main cases arise depending on the number of fragments and parameter  $m$ . The first one is when  $n < mh$ . In this case, there are not enough fragments of file  $f$  to distribute  $m$  of them to each subnetwork in  $H$ . Moreover, there is no gain in allowing a subnetwork  $S_i$  to store more than  $m$  fragments. Indeed, there would exist another subnetwork  $S_j$  storing less than  $m$  fragments, and subnetwork  $S_j$  would benefit from storing an extra fragment both for reading and for writing, while  $S_i$  uses fragments exceeding the first  $m$  only

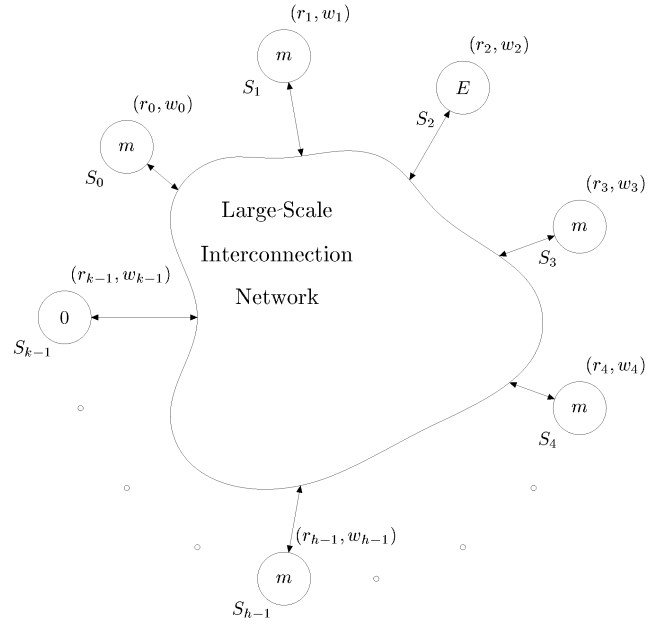


Fig. 6. Optimal distribution when  $mh \leq n$ .  $E = n - m(h - 1)$  fragments are stored in subnetwork  $S_s$ , where  $s = 2$ ; the other subnetworks in  $H$  store  $m$  fragments each. Subnetworks in  $\overline{H}$  are empty.

when writing. Since both  $S_i$  and  $S_j$  are in  $H$ , it is not possible that the write frequency of  $S_i$  exceeds the combined read and write frequency of  $S_j$ .

When  $n \geq mh$ , we have the second case, shown in Fig. 6. Here, there is no point in letting a subnetwork store less than  $m$  fragments, as there are enough fragments to cope with the reading needs of all subnetworks. Hence, after mapping  $m$  fragments of file  $f$  to each of the  $h$  subnetworks  $S_0, \dots, S_{h-1}$ , the remaining  $n - mh$  fragments are accessed by the clients only when writing  $f$ . Intuitively, the best choice is to map all of the remaining fragments to the subnetwork with the highest write frequency related to file  $f$ . Let  $s$  be the index of such a network in  $H$ , i.e.,  $s \in H$  is such that  $w_s \geq w_j \forall j \in H$ .

**Lemma 2.** *Let distribution  $d_{OPT}$  for a file  $f$  be defined as follows:*

$$d_{OPT}(s) = \begin{cases} n \ominus ms & \text{if } n < m(s + 1), \\ m & \text{if } m(s + 1) \leq n \leq mh, \\ n - m(h - 1) & \text{if } mh < n, \end{cases}$$

$$d_{OPT}(i) = \begin{cases} m & \text{if } n \geq m(i + 1), \\ n \ominus mi, & \text{otherwise,} \end{cases}$$

$$d_{OPT}(j) = 0,$$

where  $i \in H \setminus \{s\}$  and  $j \in \overline{H}$ . Then, distribution  $d_{OPT}$  has optimal cost.

**Proof.** Cost function (10) is convex. Note that functions  $g_1(x) = m \ominus x$  and  $g_2(x) = n - x$  are convex, and the cost function is thus convex as it is a linear combination of convex functions. By Lemma 1, all feasible distributions  $d$  are those such that  $\sum_i d(i) = n$ ,  $d(i) \geq 0$  for all  $i \in H$ , and  $d(j) = 0$  for all  $j \in \overline{H}$ , which is a convex space. It is known that local minima of a convex function defined on a convex space are also global minima [17]. Consequently, it is enough to show that  $d_{OPT}$  is a local minimum.

Take a generic distribution  $d'$  from the neighborhood of  $d_{\text{OPT}}$  in the convex space defined above. Distribution  $d'$  can be obtained from  $d_{\text{OPT}}$  by moving a fragment from subnetwork  $S_a$  where  $d_{\text{OPT}}(a) > 0$ , to subnetwork  $S_b$  where  $d_{\text{OPT}}(b) < n$  and  $a \neq b$ . Consider  $\Delta = \text{cost}(d_{\text{OPT}}) - \text{cost}(d')$ , which is the improvement of  $d'$  over  $d_{\text{OPT}}$ :

$$\begin{aligned} \Delta &= r_b(m \ominus d_{\text{OPT}}(b)) + w_b(n - d_{\text{OPT}}(b)) + \\ &\quad + r_a(m \ominus d_{\text{OPT}}(a)) + w_a(n - d_{\text{OPT}}(a)) + \\ &\quad - r_b(m \ominus (d_{\text{OPT}}(b) + 1)) + \\ &\quad - w_b(n - (d_{\text{OPT}}(b) + 1)) + \\ &\quad - r_a(m \ominus (d_{\text{OPT}}(a) - 1)) + \\ &\quad - w_a(n - (d_{\text{OPT}}(a) - 1)) = \\ &= b_0 r_b + w_b - b_1 r_a - w_a, \end{aligned}$$

where  $b_0$  is a bit set to one if and only if  $d(b) < m$ , and  $b_1$  is a bit set to one if and only if  $d(a) \leq m$ . Now, we proceed by cases.

If  $n < mh$ , then  $d_{\text{OPT}}(i) > 0$  for all  $i < \lceil n/m \rceil$ . Consequently,  $a < \lceil n/m \rceil$ ,  $d_{\text{OPT}}(a) \leq m$ , and  $b_1 = 1$ . Consider now  $b$ . If  $b < a$ , then  $d_{\text{OPT}}(b) = m$ ,  $b_0 = 0$ , and  $\Delta = w_b - (r_a + w_a) \leq 0$  since  $a, b \in H$ . Otherwise, if  $b > a$ , then  $\Delta \leq r_b + w_b - (r_a + w_a) \leq 0$  since subnetworks are sorted by read write frequencies.

If  $n \geq mh$ , then  $d_{\text{OPT}}(i) \geq m$  for all  $i \in H$ , consequently,  $b_0 = 0$ . Consider now,  $a$ . If  $a \neq s$ , then  $b_1 = 1$ , and  $\Delta = w_b - (r_a + w_a) \leq 0$  since  $a, b \in H$ . Otherwise, if  $a = s$ , then  $\Delta = w_b - (b_1 r_s + w_s) \leq w_b - w_s \leq 0$  by definition of  $s$ .

In all cases,  $\text{cost}(d_{\text{OPT}}) \leq \text{cost}(d')$ , hence,  $d_{\text{OPT}}$  is a local minimum and the lemma is proved.  $\square$

Although our results hold in the general case, for the ease of explanation we assume that  $(r_i + w_i) \neq (r_j + w_j)$  for all  $i \neq j$ . Moreover, we assume that  $\Delta t$  is long enough that each fragment  $f_r$  receives a read operation from every reading client and a write operation from every writing client during any  $\Delta$ -period. This requirement can be ensured by choosing  $\Delta t$  such that:

$$\Delta t \geq \max_c \left\{ \frac{n - m + 1}{R_c}, \frac{1}{W_c} \right\}, \quad (13)$$

where  $R_c$  and  $W_c$  are the read and write frequencies of a client  $c$ . Note that servers are not synchronized and, thus, periodic resets do not occur at the same instant on all servers. Under these hypotheses, the next theorem formally shows that our dynamic allocation algorithm converges to an optimal distribution.

**Theorem 1.** *If the read write frequency patterns are stable, the dynamic allocation algorithm of Figs. 4 and 5 converges to an optimal mapping for file  $f$ .*

**Proof [(sketched)].** Consider a fragment  $f_r$  stored in a server in subnetwork  $S_i$  such that  $i \in \overline{H}$ , just after a periodic reset of frequencies. Since  $\overline{H}$  is not empty, it follows that  $w_s > 0$ , where  $s$  is such that  $w_s \geq w_j$  for all  $j$ . Before the next reset occurs, at least one write request from each client in subnetwork  $S_s$  reaches  $f_r$ . At the next reset, the server will move fragment  $f_r$  outside  $\overline{H}$  since

$\tilde{w}_s > \tilde{r}_i + \tilde{w}_i$  for all  $i \in \overline{H}$ . Such a fragment  $f_r$  will never be moved back into a subnetwork  $S_i \in \overline{H}$ , as  $w_s > (r_j + w_j)$  for all  $j \in \overline{H}$ . As a consequence, all fragments in subnetworks in  $\overline{H}$  will be eventually moved to subnetworks in  $H$ .

We now focus on subnetworks in  $H$ . We say that fragment  $f_r$  *knows* subnetwork  $S_i$  if fragment  $f_r$  has been reached by a read operation issued by each reading client of subnetwork  $S_i$ , and by a write operation issued by each writing client of subnetwork  $S_i$ .

We start from the case  $n < mh$ . Consider subnetwork  $S_0$ . If  $S_0$  stores  $q$  fragments for a complete  $\Delta t$ -period, where  $q < m$ , both read and write operations generated at subnetwork  $S_0$  reaches fragments outside  $S_0$ . Hence, at least  $m - q$  fragments outside  $S_0$  know subnetwork  $S_0$ . These fragments, at their next reset, will move to  $S_0$ , since  $r_0 + w_0$  is the largest frequency in the system. However, after  $m$  fragments moved to subnetwork  $S_0$ , clients in  $S_0$  do not access more than  $m$  fragments while reading. The servers storing these fragments, after a reset operation, will move their fragment to another subnetwork  $i$  in  $H$ , if such a subnetwork exists. Consequently, subnetwork  $S_0$  will eventually store exactly  $d_{\text{OPT}}(0)$  fragments. A similar argument holds for  $S_1, S_2, \dots, S_{\lceil n/m \rceil - 1}$ , and the algorithm thus converges to  $d_{\text{OPT}}$ .

Consider now, the case  $n \geq mh$ . For the argument given above, all subnetworks in  $H$  will eventually store at least  $m$  fragments. When this even occurs, the exceeding fragments will be accessed only on write operations. All these fragments will thus move to subnetwork  $S_s$ , which has the highest write frequency, and the algorithm converges again to distribution  $d_{\text{OPT}}$ .  $\square$

## 5 CONCLUDING REMARKS

This paper first proposes a model to measure the assurance level of static and dynamic file allocation schemes in a large-scale distributed system, and then discusses and analyzes an optimal dynamic allocation algorithm that provides high assurance, availability, performance, and scalability. Though a subset of these constraints may be considered as traditional research topics, however, they are only partially satisfied by the classical infrastructures (future community may grow up to millions of members). In the future infrastructures, a large number of geographically distributed, autonomous servers should use peer-to-peer algorithms in order to provide data storage and ubiquitous access and update to clients in a scalable and dynamically reconfigurable way.

This paper focuses mainly on dynamic secure file allocation in such large-scale distributed infrastructures. Related future works also include the issues of load-balancing, job scheduling, and the effects of malicious node and denial of service attacks. We also plan some experimental evaluations to define and tune the parameters to preserve the scalability of the solutions.

## ACKNOWLEDGMENTS

A. Mei and L.V. Mancini have been funded by the WEB-MINDS project, supported by the Italian MIUR under the FIRB program.

## REFERENCES

- [1] R. Sandberg, D. Goldberg, S. Kleinman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network File System," *Proc. USENIX Summer Conf.*, June 1985.
- [2] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, "NFS Version 3 Design and Implementation," *Proc. USENIX Summer 1994 Technical Conf.*, June 1994.
- [3] J.H. Morris, M. Satyanarayanan, M.H. Conner, J.H. Howard, D.H.S. Rosenthal, and F.D. Smith, "Andrew: A Distributed Personal Computing Environment," *Comm. ACM*, vol. 29, no. 3, 1986.
- [4] Carnegie Mellon Univ., Pa., Coda File System, <http://www.coda.cs.cmu.edu/>, 2003.
- [5] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, 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," *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2000.
- [6] J.J. Wylie, M.W. Bigrigg, J.D. Strunk, G.R. Ganger, H. Kiliccote, and P.K. Khosla, "Survivable Information Storage Systems," *Computer*, vol. 33, no. 8, Aug. 2000.
- [7] A. Shamir, "How to Share a Secret," *Comm. ACM*, vol. 22, no. 11, 1979.
- [8] M.O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," *J. ACM*, vol. 36, no. 2, 1989.
- [9] G. Blakley, "Safeguarding Cryptographic Keys," *Proc. AFIPS Nat'l Computer Conf.*, June 1979.
- [10] H. Krawczyk, "Secret Sharing Made Short," *Advances in Cryptology, Proc. 13th Ann. Int'l Cryptology Conf.*, Aug. 1993.
- [11] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS," *Proc. 18th ACM Symp. Operating Systems Principles*, 2001.
- [12] W.J. Bolosky, J.R. Douceur, D. Ely, and M. Theimer, "Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs," *Proc. Int'l Conf. Measurement and Modeling of Computer Systems*, 2000.
- [13] A.V. Goldberg and P.N. Yianilos, "Towards an Archival Inter-memory," *Proc. IEEE Advances in Digital Libraries Conf.*, 1998.
- [14] A. Rowstron and P. Druschel, "Storage Management and Caching in Past, A Large-Scale, Persistent Peer-to-Peer Storage Utility," *Proc. 18th ACM Symp. Operating Systems Principles*, 2001.
- [15] J.J. Wylie, M. Bakkaloglu, V. Pandurangan, M.W. Bigrigg, S. Oguz, K. Tew, C. Williams, G.R. Ganger, and P.K. Khosla, "Selecting the Right Data Distribution Scheme for a Survivable Storage System," Technical Report CMU-CS-01-120, Carnegie Mellon Univ., May 2001.
- [16] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, UK: Cambridge Univ. Press, 1995.
- [17] A.L. Peressini, F.E. Sullivan, and J.J. Uhl, *The Mathematics of Nonlinear Programming*. Springer Verlag, 1988.



**Luigi V. Mancini** received the PhD degree in computer science from the University of Newcastle upon Tyne, United Kingdom, in 1989. He is a full professor of computer science in the Dipartimento di Informatica at the University "La Sapienza" in Roma, Italy. From 1989 to 1992, he was an assistant professor in the Dipartimento di Informatica at the University of Pisa, Italy. From 1992 to 2000, he was an associate professor in the Dipartimento di Informatica e Scienze dell'Informazione at the University of Genova, Italy, and then at the Dipartimento di Informatica of the University of Roma "La Sapienza." Since 1994, he has been a visiting professor at the Center for Secure Information Systems, George Mason University, Virginia. His current research interests include: computer network and information security, fault-tolerant large-scale distributed systems, and hard-real-time distributed systems. He published more than 50 scientific papers in international conferences and journals such as: *ACM TISSEC*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Parallel and Distributed Systems*, and *IEEE Transactions on Software Engineering*. He served on numerous program committees of international conferences. Currently, he is the PI for the University of Roma "La Sapienza" of the project "WEB-MINDS," funded by the Italian government (2002-2005).



**Sushil Jajodia** received the PhD degree from the University of Oregon, Eugene. He is the BDM International Professor of information and software engineering and the director of the Center for Secure Information Systems at the George Mason University (GMU), Fairfax, Virginia. He served as the chair of the Department of Information and Software Engineering during 1998-2002. He joined GMU after serving as the director of the Database and Expert Systems Program at the US National Science Foundation. Before that, he was the head of the Database and Distributed Systems Section at the Naval Research Laboratory, Washington. He has also been a visiting professor at the University of Milan and the University of Rome "La Sapienza," Italy, and at the Isaac Newton Institute for Mathematical Sciences, Cambridge University, England. His research interests include information security, temporal databases, and replicated databases. He has authored four books, edited 19 books, and published more than 250 technical papers in the refereed journals and conference proceedings. He received the 1996 Kristian Beckman award from IFIP TC 11 for his contributions to the discipline of information security, and the 2000 Outstanding Research Faculty Award from GMU's School of Information Technology and Engineering. Dr. Jajodia has served in different capacities for various journals and conferences. He is the founding editor-in-chief of the *Journal of Computer Security* and has served on the editorial boards of *ACM Transactions on Information and Systems Security* and the *International Journal of Cooperative Information Systems*. He is the consulting editor of the *Kluwer International Series on Advances in Information Security*. He serves as the general chair of the 10th ACM Conference on Computer Security (CCS 2003) and as program chair of the Sixth IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems. He also serves as the chair of the ACM Special Interest Group on Security, Audit and Control (SIGSAC), and the IFIP WG 11.5 on Systems Integrity and Control. He is a senior member of the IEEE and a member of IEEE Computer Society and the ACM. The URL for his web page is <http://csis.gmu.edu/faculty/jajodia.html>.



**Alessandro Mei** received the Laurea degree (summa cum laude) in computer science from the University of Pisa, Italy, in 1994, and the PhD degree in mathematics from the University of Trento, Italy, in 1999. Afterward, he held a postdoctorate position at the University of Trento. In 2001, he joined the Faculty of Science at the University of Rome "La Sapienza" as an assistant professor. His main research interests include security issues of distributed systems and algorithms for distributed, parallel, and reconfigurable systems.

► For more information on this or any other computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.