Middleware for a Re-configurable Distributed Archival Store Based on Secret Sharing

Shiva Chaitanya[1], Dharani Vijayakumar[2], Bhuvan Urgaonkar[3],
and Anand Sivasubramaniam[3]

[1] Netapp Inc.
[2] VMware Inc.
[3] The Pennsylvania State University

Abstract. Modern storage systems are often faced with complex trade-offs between the confidentiality, availability, and performance they offer their users. Secret sharing is a data encoding technique that provides information-theoretically provable guarantees on confidentiality unlike conventional encryption. Additionally, secret sharing provides quantifiable guarantees on the availability of the encoded data. We argue that these properties make secret sharing-based encoding of data particularly suitable for the design of increasingly popular and important distributed archival data stores. These guarantees, however, come at the cost of increased resource consumption during reads/writes. Consequently, it is desirable that such a storage system employ techniques that could dynamically transform data representation to operate the store within required confidentiality, availability, and performance regimes (or budgets) despite changes to the operating environment. Since state-of-the-art transformation techniques suffer from prohibitive data transfer overheads, we develop a middleware for dynamic data transformation. Using this, we propose the design and operation of a secure, available, and tunable distributed archival store called FlexArchive. Using a combination of analysis and empirical evaluation, we demonstrate the feasibility of our archival store. In particular, we demonstrate that FlexArchive can achieve dynamic data re-configurations in significantly lower times (factor of 50 or more) without any sacrifice in confidentiality and with a negligible loss in availability (less than 1%).

Keywords: Secret sharing, archival storage, confidentiality, performance, availability.

1 Introduction

The last decade has witnessed a deluge of digital data that need to be safely archived for future generations [9]. Rapid increase in sensitive online data such as health-care, customer, and financial records has contributed to this unprecedented growth. The challenges facing such archival data stem from the need to ensure their long-term confidentiality and availability. Many factors mandate these requirements, ranging from preservation, retrieval, and security properties demanded by legislation to long lifetimes expected for cultural and family heritage data. To address data confidentiality, modern storage systems typically employ encryption-based techniques (see survey paper [20]). The use of data encryption for archival lifetimes, however, introduces problems that

have been well-documented [22,21]. The primary drawback is that data secured using keyed cryptography are only *computationally secure*—they are decipherable via crypt-analysis given sufficient computing power/time.

Secret sharing is a data encoding technique that offers the promise of overcoming these shortcomings of an encryption-based archival storage. Secret sharing with parameters (m, n) breaks a data block into n fragments (each of the same size as the original data block) in such a manner that at least m fragments must be obtained to re-construct the original block. These fragments are stored in n different storage nodes and an adversary has to obtain access to at least m fragments to decipher the original data - any set of fewer than m fragments provides no information about the original block. This property provides a quantitative notion of data confidentiality. Additionally, the original data item is resilient to the loss of fragments in the following manner: it can be re-constructed even when $(n - m)$ fragments are lost. This provides a quantitative measure of the availability properties of encoded data.

In this paper, we address the important problem of dynamically re-configuring the secret sharing parameters (m, n) used to encode data in a distributed archival store. The need to re-configure could arise as a result of one or more of the following scenarios. First, a subset of the storage nodes comprising the archival store might become un-available or unreliable due to some form of security compromise or component failure. The data fragments at affected nodes must be considered lost and the archival system must be reverted back to its original settings. Second, there could be infrastructural changes to the storage network (e.g., addition of new nodes) which are likely to happen quite frequently relative to the lifetime of the archival data. Finally, the secrecy, avail-ability, or performance needs of an archival store might change with time (e.g., due to changes in regulations or societal changes resulting in the stored data becoming more sensitive). Existing archival systems that have incorporated secret sharing to achieve the goals of secure long-term preservation of data have either (i) neglected this problem of re-configuration (e.g., Potshards [22]), or (ii) proposed inefficient techniques (e.g., PASIS [24]). Whereas some key aspects of the problem of dynamically re-configuring secret sharing parameters have been studied [4,6,2,3], the approaches emerging out of this body of work have severe drawbacks when used to build a practical archival storage system. In particular, they suffer from the following two main drawbacks:

- **High data access overhead.** Existing re-configuration techniques require access to m fragments for every data object stored using a (m, n) configuration. Many archival storage systems store data across wide-area networks (often with components that need to be accessed via congested or inherently slow links) and use cheap storage media technologies wherein reads to the original data can be quite slow. As we will observe later in this paper, for archival storage, it is desirable for the value of m to be close to n. Thus, the data traffic resulting from a re-configuration can become a limiting factor.

- **High computational overhead.** Existing re-configuration techniques suffer from high computational overheads. These overheads could be prohibitive in the context of archival systems that deal with very large volumes of data. It is desired that a re-configuration technique complete fast enough so the archival system spends a small amount of time in an unstable (and hence, potentially vulnerable) configuration.

1.1   Research Contributions

Our contribution is twofold.

- We propose a re-configuration technique called Multi-share Split that is both lightweight in terms of (i) the computational and I/O overheads it imposes on the archival storage system as well as (ii) tunable in terms of the trade-offs it offers between confidentiality, availability, and performance. We expect that Multi-share Split would enable administrators of archival stores to make appropriate choices to best satisfy the requirements (e.g, completion time targets, network resource constraints) of their systems.
- Using our re-configuration technique, we design and implement a middleware that is used by nodes comprising a distributed archival storage system called FlexArchive. We analyze the security and availability properties offered by FlexArchive and conduct an empirical evaluation of the feasibility and efficacy of FlexArchive using a prototype networked storage system.

1.2   Road-Map

The rest of this paper is organized as follows. We discuss some background material in Section 2. We introduce the proposed FlexArchive system in Section 3 and describe the re-configuration algorithm employed by FlexArchive in Section 4. We develop analytical techniques for characterizing the availability offered by FlexArchive in Section 5. We conduct an empirical evaluation of the efficacy of FlexArchive in Section 6. Finally, we present concluding remarks in Section 7.

2   Background and Related Work

In this section, we provide basic background on secret sharing and its appropriateness for archival storage.

2.1   Basics of Secret Sharing

An (m, n) secret sharing scheme, where $m \leq n, m > 0$, creates n fragments from a data item with the following properties: given any m fragments, one can re-construct the data item; however, fewer than m fragments provide no information about the original data item. Such classes of secret sharing techniques are "perfectly secure" in the sense that they exhibit information-theoretic security. The size of each fragment for secret sharing schemes is provably the same as that of the original data item. Hence, the storage needs are n times the size of the original data.

A number of secret sharing techniques have been proposed that differ very slightly in their computational complexity. We use a secret sharing scheme due to Shamir (often called "Shamir's threshold scheme" [19]). The key idea behind Shamir's threshold scheme is that m points are needed to define a polynomial of degree $(m - 1)$ (e.g., two points for a line, three points for a hyperbola, four points for a third-degree polynomial, and so forth). Shamir's threshold scheme, for representing a data item S with secret sharing parameters (m, n), chooses uniformly random $(m - 1)$ coefficients

$a_1, \cdots, a_{m-1}$, and lets $a_0 = S$. It then builds the polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{m-1} x^{m-1}$. Finally, it computes the values taken by this polynomial for n distinct values of x comprising the set $\{x_1, \cdots, x_n\}$. The n shares of the secret S are now given by the pairs $(x_i, f(x_i))$. Given any m of these pairs, one can find the coefficients of the polynomial $f(.)$ by interpolation, and then evaluate the secret $S = a_0$. Geometrically, on a $X - Y$ plane, one can think of the secret S as being the Y-intercept of the curve defined by the polynomial $f(.)$; the shares are the Y-values at $x_1, \cdots, x_n$. Note that since we are dealing with finite values (the secret and the shares are data values and represented by say, q bits), the $X - Y$ plane is a finite field with the range of values 0 to $2^q - 1$. Since the participants holding the shares could implicitly define the n indices for which the shares are computed (e.g., based on their unique names/identities), each share is simply the value $f(x_i)$ and hence can be represented using q bits, same as those needed for the secret S.

2.2   Long Term Data Confidentiality

Two fundamental classes of mechanisms for enforcing data secrecy are those based on encryption and secret sharing, respectively. Many systems such as OceanStore [17], FARSITE [5], SNAD [14], Plutus [11], and e-Vault [10] address file secrecy, but rely on the explicit use of keyed encryption. Keyed encryption may work reasonably well for short-term secrecy needs but it is less than ideal for the long-term security problem that the current work addresses. Keyed cryptography is only computationally secure, so compromise of an archive of encrypted data is a potential problem regardless of the encryption algorithm used. An adversary who compromises an encrypted archive need only wait for cryptanalysis techniques to catch up with the encryption used at the time of the compromise. If an insider at a given archive gains access to all of its data, he can decrypt any desired information even if the data is subsequently re-encrypted by the archive, since the insider will have access to the new key by virtue of his internal access. Encrypted data can be deciphered by anyone, given sufficient CPU cycles and advances in cryptanalysis. Furthermore, future advances in quantum computing have the potential to make many modern cryptographic algorithms obsolete. For long-lasting applications, encryption also introduces the problems of lost keys, compromised keys, and even compromised crypto-systems. Additionally, the management of keys becomes difficult because data might experience many key rotations and crypto-system migrations over the course of several decades. This must all be done without user intervention because the user who stored the data may be unavailable.

Moving away from encryption to secret sharing enables an archival storage system to rely on the more flexible and secure authentication realm. Unlike encryption, authentication need not be done by a computer and authentication schemes can be easily changed in response to new vulnerabilities. Secret sharing improves the security guarantees by forcing an adversary to breach *multiple* archival sites to obtain meaningful information about the data. Several recently proposed archival systems such as POTSHARDS [22], PASIS [25,7], and GridSharing [23] employ secret sharing schemes for this reason.

2.3   Long Term Data Availability

Recovering from disk failures to large-scale site disasters has long been a concern for storage systems. The long lifetimes of archival data make them prone to latter type of scenarious. Keeton et al. [12] highlighted the importance of efficient storage system design for disaster recovery by providing automated tools that combine solutions such as tape backup, remote mirroring, site fail-over, etc. These tools strive to select designs that meet the financial and recovery objectives under specified disaster scenarios. Totalrecall [1], Glacier [8] and Oceanstore [17] are examples of distributed storage systems where high availability is an explicit requirement. These systems use RAID-style algorithms or more general erasure coded redundancy techniques along with data distribution to guard against node failures. Erasure coded techniques could be thought of as similar to secret sharing minus the "secrecy" gaurantees, in the following sense: similar to secret sharing, an (m, n) erasure coding divides a data block into n fragments such that at least m fragments are needed to re-construct the original block. However, unlike secret sharing, access to less than m data fragments might reveal partial information about the data block.

3   FlexArchive: A Re-configurable Secret Sharing-Based Archival Store

We assume a distributed archival system called *FlexArchive* consisting of multiple storage nodes that are spread across various sites. Each site in FlexArchive could be professionally managed with internal redundancy mechanisms to protect data against component failures. In Figure 1 an "archival site" refers to an SSP-managed (SSP stands for Storage Service Provider) or an internal-to-enterprise storage site. A representative example of the system we assume is Safestore [13], which consists of multiple sites, each owned by a different SSP. Each SSP provides storage services in return for fees based on some agreed-upon contract. The contract might also include penalties for losing data. We assume these penalties to be such that it is in the best interests of the SSPs to provide internal redundancy mechanisms. However, large-scale component failures like disasters and correlated site failures [12,15] are harder to protect against. Considering the increased probability of such events during the lifetime of archival data, the secret distribution algorithm employed by FlexArchive must provide for inter-site redundancy as well. FlexArchive employs a secret distribution scheme where each fragment encoding a data unit goes to a different storage node. Assuming secret sharing parameters (m, n) for a data item under discussion, the value $(n - m)$ captures the archival system's resilience against node failures. Even though SSPs can be employed to safely store and retrieve data, they cannot be trusted to preserve data privacy—e.g., multiple sites can collude to obtain the original data. The parameter m captures the difficulty of accessing data in FlexArchive.

For digital data that must last for decades or even centuries, the original writer of a data item must be assumed to be unavailable when it is read. We assume that sites comprising FlexArchive implement access control policies to identify and verify the permissions of the reader/writer. Another important security property is data integrity,
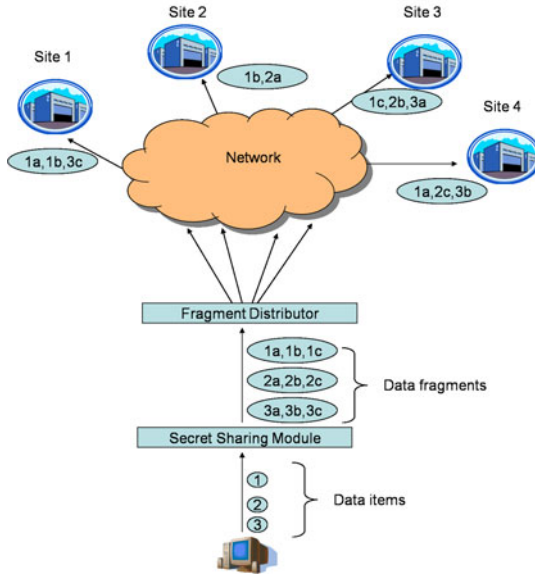
Fig. 1. FlexArchive system model. We show the middleware at a client that facilitates data encoding and placement on servers part of the FlexArchive archival store as well as re-configuration. We show an illustrative system with $N = 4$ storage sites and data being encoded using a configuration confirming to $(m \leq 3, n = 3)$.

which refers to the ability of the system to identify modifications to the data while at rest or in transit. In short-lived storage systems, data integrity is ensured by storing secure hashes along with the data using one-way hash functions such as MD5, SHA1. We assume the problems of user authorization/authentication and data integrity beyond the scope of our current work and focus only on data confidentiality.

We assume that FlexArchive provides an interface to its clients to create data in the form of archival "objects." An object could be a file in a server file system or it could be a smaller granularity unit like a disk block or an extent of blocks. As shown in Figure 1, we assume that each client employs a middleware that implements two units: (i) the secret sharing module and (ii) the fragment distributor. The secret sharing module is responsible for encoding/decoding data and is completely local to the client system. The fragment distributor is responsible for determining the placement of fragments encoded by the secret sharing unit on FlexArchive sites (and decoding fragments read from FlexArchive) and relies on certain information about the current state of FlexArchive for its decision-making. The placement strategy employed by the fragment distribution unit could vary depending on the actual storage structure. For example, in WAN-based distributed storage, it is often the case that the storage nodes across sites (or certain subsets of them) are independent of each other in terms of security breaches and failures. Therefore, a desirable way of assigning fragments representing a data object to sites is to store each fragment on a node within a different site. This would render data fragments independent with respect to failures/compromises and utilize well the properties of secret sharing. Typically, the value of n used for secret sharing is in the range

5-20, significantly smaller compared to the number of participating storage sites, N, that could be in the hundreds or even a few thousands spread across a WAN. We assume that the fragment distributor would use some form of load balancing when distributing the fragments across these sites.

3.1 Re-configuration in FlexArchive: Key Concerns

The usage scenario of FlexArchive is write-once, read-maybe, and thus stresses through-put over low-latency performance. Within each FlexArchive site, writes are assumed to be performed as part of a background process in a way that the accompanying latencies do not affect the foreground system performance. The key factors affecting archival I/O performance are (i) CPU computation at the secret sharing and the fragment distribution units, (ii) network latency between the client and FlexArchive nodes, and (iii) storage access latency at FlexArchive nodes. The choice of secret sharing parameters m and n is dictated by the confidentiality and availability guarantees desired by the client. For example, if it is well understood that the simultaneous occurrence of (i) m or more nodes being compromised and (ii) more than k nodes becoming unavailable simultaneously are extremely unlikely scenarios, a good rule-of-thumb would be to employ an $(m, m + k)$ secret sharing scheme. Selecting the number of shares required to re-construct a secret-shared value involves a trade-off between availability and confidentiality—the higher the number of sites that must be compromised to steal the secret, the higher the number of sites that must remain operational to provide it legitimately. Clearly, no single data distribution scheme is right for all systems. The right choice of these parameters depends on several factors, including expected workload, system component characteristics, and the desired levels of availability and security. PASIS [25,7,24] proposes analytic techniques to ascertain the right secret sharing parameters for a given system. FlexArchive builds upon the insights provided by this body of work for determining appropriate values for these parameters.

We use the term *security budget* to denote the minimum number of fragments that need to be compromised to constitute an attack. For an (m, n) configuration, the security budget is equal to m. Similarly, we use the term *availability budget* to denote the maximum number of sites that can fail while still allowing the reconstruction of a stored data item. Since a preserved data item is available under the loss of at most $(n - m)$ fragments, this number represents its availability budget. After the archival objects have been initially written onto n FlexArchive sites, intermediate changes to the configuration are necessitated by one or a combination of the following.

- **Scenario 1: Node Compromise.** If one of the storage nodes suffers a security breach resulting in the potential exposure of all the data it stores, all affected objects would experience a decrease in their security budgets by one. To revert the system to its original security settings, the system administrator would need to initiate a re-configuration of the remaining fragments of the affected objects to restore their security budgets, i.e., a transformation of a set of fragments belonging to an $(m - 1, n - 1)$ configuration to $(m, n - 1)$. In general, a compromise of k nodes would require a re-configuration of an affected object from a rendered $(m - a, n - a)$ to $(m, n - a)$, where $1 \leq a \leq k$. The parameter a is the number of fragments belonging to the affected object that were stored at the k nodes.

- **Scenario 2: Data Loss.** Permanent loss of data at a storage node reduces the configuration of all affected objects to $(m, n - 1)$. To recuperate, the administrator might wish to deploy a spare storage node or alternate storage space at the affected node. In either case, a re-configuration from $(m, n - 1)$ to (m, n) for all affected objects is required. In general, permanent data loss will require reconfigurations of the form $(m, n - a)$ to (m, n).
- **Scenario 3: Infrastructure Changes.** Due to the lengthy periods of time for which the archival data are stored, they will witness plenty of changes to the underlying storage infrastructure. As a result, the objects may need to be moved to a different configuration to accommodate the new infrastructure. This might mean changes to m or n or both.

Since storage systems retire data slowly compared to the rate of their expansion due to the rapidly decreasing costs of storage and increasing storage densities over time, scenario 3 would often require increasing m or n or both. Scenario 1 would always require increasing the value m, whereas scenario 2 would require increasing the value of n. We focus only on re-configuration techniques that perform up-scaling (increasing m or n or both). A "naive" (m, n) to (m', n') re-configuration technique is as follows: (i) reconstruct the secret by accessing and combining any m fragments, (ii) split the secret into n' fragments using an (m', n') configuration, and (iii) delete the original n fragments and replace them with the n' fragments constructed in the last step. The obvious problem with this approach is the reconstruction of the original secret during the reconfiguration process. The node performing the re-configuration becomes a central point of attack as it can expose the secret. The fact that a secret was originally stored as a (m, n) configuration indicates that an adversary must compromise at least m different entities to obtain a secret. This security property needs to be preserved at all times, even during the re-configuration process.

3.2   Re-configuration in FlexArchive: A Baseline Technique

We now briefly describe a technique by Jajodia et al. [4] to perform an (m, n) to (m', n') re-configuration that preserves the above property. We will use this scheme as a *baseline* against which we will compare our techniques. To the best of our knowledge, this is the only significant body of work that re-configures secret sharing-encoded data without exposing the original secret during the process. A threshold subset m original shareholders perform a (m', n') secret splitting on each of their shares to obtain n' sub-shares each. Following this, each of the n' sub-shares from an original shareholder is sent to a target shareholder. The m sub-shares at each target shareholder are combined via the associated secret construction algorithm to obtain a resultant share. The resultant n' shares at the target shareholders are of a (m', n') configuration for the original secret. The baseline technique is illustrated in Figure 2. The main drawbacks of the baseline technique lie in its high resource consumption.

- **Storage I/O Needs.** This reconfiguration technique consumes excessive storage resources. Accessing m original shares to perform the reconfiguration for each object is a significant system bottleneck. Since the archival storage nodes are designed for write once and read rare type of workloads (typically tapes or log structured storage
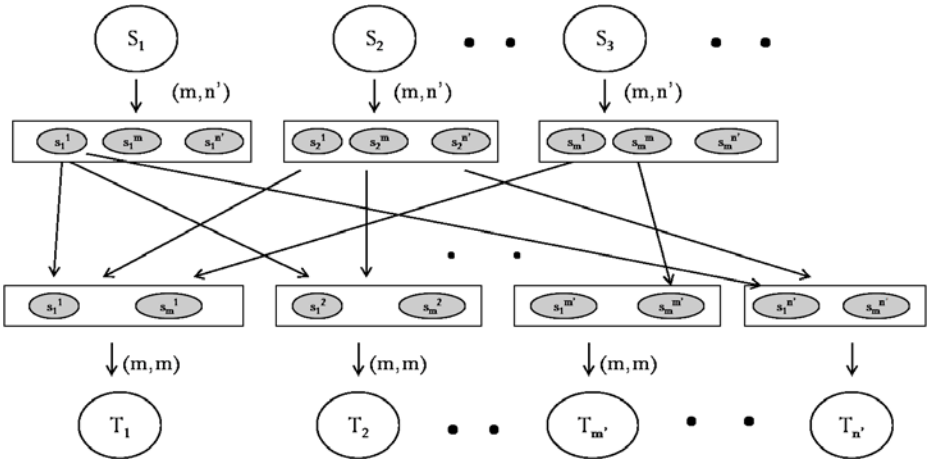
Fig. 2. Baseline re-configuration technique: from starting configuration of (m,n) to (m',n')

on disks [16,18]), the random read access to stored data is quite tedious. Under such conditions, reads to existing data for background tasks such as reconfiguration might bottleneck the actual foreground jobs of archiving. This is further compounded by the fact that re-configurations are typically done as batch jobs for millions of objects at a time. This is because they are triggered by node compromise/failure events causing all objects in the affected nodes to be re-configured. Another important factor is that, at the time of re-configuration, a subset of nodes containing the shares may not be available. In most distributed systems spread over wide geographic sites, nodes experience temporary failures because of various reasons (link failure, remote maintenance etc). There is usually a default churning of nodes, i.e nodes go down temporarily and join the network again at a later time. It is desirable that the re-configuration protocol proceeds with the alive subset of nodes (possibly, with subset of shares $< m$) instead of waiting an indefinite amount of time for m nodes to be available.

- **Network Access.** In addition to the storage access, the baseline technique involves approximately $m * n'$ network messages amongst storage nodes. This can bottleneck a typical archival storage system where all the storage nodes reside in geographically distant locations. The high latency and low bandwidth properties of network links connecting the nodes slows down the completion time of the re-configuration. There is a high possibility that many of the links towards the storage servers would be congested as a result, thereby affecting the foreground archiving jobs.

- **Computational Needs.** The computational requirements of the baseline technique are high: it requires m different instances of (m', n') secret splitting and n' instances of (m, m) secret reconstructions. As we shall observe in the evaluation section, the Shamir secret sharing operations are computationally intensive. These operations are performed in a distributed manner at the various archival sites. Depending on the computing resources available at the various storage sites to perform these operations, the CPU cycles required could become a heavy bottleneck during re-configuration.

4   Re-configuration in FlexArchive

We categorize re-configurations of secret shares as belonging to one of the following three types: (a) Improving only the availability budget (b) Improving only the security budget and (c) Improving both budgets simultaneously. In the remainder of this section, we describe our proposed approach for type (b) reconfiguration - to increase the *security budget* of a set of secret shares. It is desired that the security budget of a data item represented using a (m, n) Shamir encoding be increased by k with access to minimal number (denoted by L) of original shares. The resultant configuration requires an adversary to compromise k more nodes to obtain additional fragments than with the original configuration. It can be seen that any re-configuration technique requires at least $(n - m + 1)$ fragments to be modified. In other words, if m or more original fragments are left unmodified, the security budget remains unchanged. Therefore, $L = n - m + 1$.

4.1   Multi-share Split: FlexArchive Re-configuration Technique

At the heart of our technique, called "Multi-share Split," lies a subroutine called "Multi-transform" that leverages Shamir encoding to create additional (or "secondary") fragments from a subset of the original (or "primary") fragments. In the following discussion, we first describe how Multi-transform works. We then describe the working of the Multi-share Split technique.

How Multi-transform Operates. Multi-transform operates on a set of x primary fragments to produce y secondary fragments ($y > x$, $1 \leq c \leq (y - x)$) with the following property: *to retrieve a subset of size x_1 of the x primary fragments, at least $(x_1 + c)$ of the y secondary fragments must be acquired.* We call this the c-increment property, which has the following implication: access to *any* $(x_1 + c)$ secondary fragments may not result in an exposure of the x_1 primary fragments. Trivially, for $x = 1$, the $(1 + c, y)$ Shamir encoding satisfies the above property. We refer to this special case of Multi-transform as "Uni-transform." Multi-transform is a general technique that works for any values of x, y and c.

   We first show the working of Multi-transform(x, y, c) using a couple of examples. In the first example, let $x = 2, y = 3, c = 1$, i.e., we would like to transform two primary fragments p_1, p_2 into three secondary fragments s_1, s_2, s_3 in such a manner that the following holds: (i) at least two secondary fragments must be acquired to obtain any of the primary fragments, and (ii) all three secondary fragments must be acquired to obtain both the primary fragments. Figure 3(a) illustrates how the secondary fragments can be generated. The primary fragments p_1, p_2 are now secrets themselves and represent the Y-intercepts of a finite field. First, Multi-transform randomly generates a secondary fragment s_1 in the finite field at index 1. The points (p_1, s_1) uniquely define a line in the finite field. Next, Multi-transform evaluates the Y-intercept of this line at index 2 to obtain the secondary fragment s_2. Finally, the line defined by the points (p_2, s_2) is used to generate the secondary fragment s_3 at index 3. Clearly, the three secondary fragments satisfy the desired increment property and hence can be used to replace the two primary fragments. Figure 3(b) shows another example for $x = 3, y = 6, c = 2$. Instead of lines,
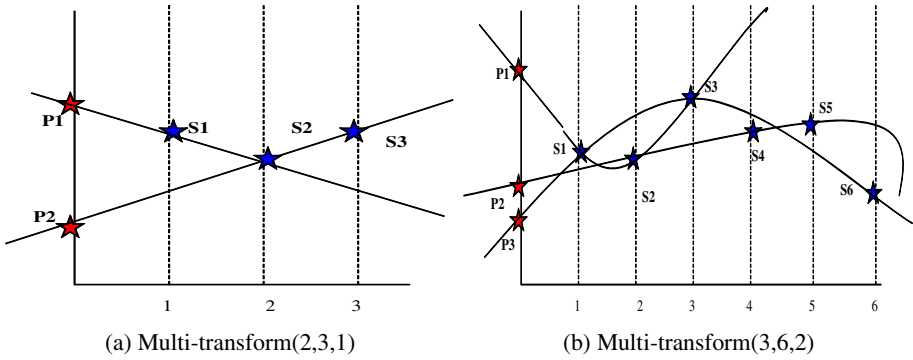
(a) Multi-transform(2,3,1)   (b) Multi-transform(3,6,2)

Fig. 3. Graphical illustration of the Multi-transform(x,y,c) technique for small values of x, y, and c

we now have three second-degree polynomials in the finite field. In the general transformation of x primary fragments to y secondary fragments, there are x polynomials of cth degree. Multi-transform(x, y, c) is described formally in Algorithm 1.

The operation of Multi-transform(x, y, c) comprises several phases. In the first phase, using the first primary fragment, $(c + 1)$ secondary fragments are created by constructing a random polynomial of cth degree. This, in fact, amounts to a $(c + 1, c + 1)$ Shamir splitting of the first primary fragment. Note that this ensures the c-increment property for the first primary fragment. In the second phase, using the second primary fragment and a portion of the $c + 1$ secondary fragments generated so far, additional secondary fragments are created. Similarly, for the third primary fragment and so on till the xth primary fragment. After the first phase, since at each additional phase, at least one new secondary fragment is created ($y \geq x + c$), the c-increment property is preserved in an inductive fashion. The algorithm strives to balance the dependence of primary fragments on the secondary fragments by doing a couple of things. The number of new secondary s at each additional phase after the first phase is determined as $\left(\dfrac{y - (c + 1)}{x - 1} \right)$.
Also, the subset of already generated secondary fragments chosen at each phase (for constructing the random polynomial) is the one with those secondary fragments that have been least utilized so far in the generation of random polynomials. These two heuristics are crucial to ensuring that the y secondary fragments are equally loaded in the sense that the number of primary fragments affected by a loss of a set of secondary fragments depends solely on the cardinality of the set and not on the actual members within the set.

How Multi-share Split Employs Multi-transform. Let us now turn our attention back to the re-configuration problem of improving the security budget of a (m, n) Shamir encoding by k with access to L original shares. Multi-share Split groups a set of L primary fragments into v sets of x primary fragments and applies Multi-transform(x, y, c) on each of these v sets. We set the input parameters for the Multi-transform as follows. For the security budget to be increased by k, we set $c = k$. To reduce the ratio of secondary fragments as compared to Uni-share Split, the values must satisfy $x \geq a$

Input: x primary fragments at Y-intercepts $p_1, \cdots, p_x$

Output: y secondary fragments $s_1, \cdots, s_y$

Let $Rank(s)$ represent a data value stored for each secondary fragment s. Initialize all $Rank(s_i)$ to zero.

Let $S_{ordered}$ be an ordered set (ascending in rank) of all secondary fragments generated so far.

Let $S_{ordered}^k$ be the first k fragments from $S_{ordered}$.

foreach $i = 1$ *to* c **do**

| Randomly generate a secondary fragment s_i at index i

end

Points $\{p_1, s_1, ..s_c\}$ define a cth degree polynomial, evaluate the polynomial at index $c + 1$ to obtain s_{c+1}

Insert each fragment from set $\{s_1, ..s_{c+1}\}$ in $S_{ordered}$.

Increment $Rank(.)$ of each fragment from set $\{s_1, ..s_{c+1}\}$ by one and update $S_{ordered}$

$r_1 = \frac{y - (c+1)}{x - 1}$;

$r_2 = (y - (c + 1)) \bmod (x - 1)$;

$j = c + 2$;

$l = 1$;

if $r_2 > 0$ **then**

| **foreach** $i = 2$ *to* $r_2 + 1$ **do**

| | **foreach** $k = 1$ *to* r_1 **do**

| | | Randomly generate a secondary fragment s_j at index l

| | | $j = j + 1$;

| | | $l = l + 1$;

| | **end**

| | Points $\{p_i, s_{j-r_1}, s_{j-r_1+1}...s_{j-2}, s_{j-1}\} \cup S_{ordered}^{c-r_1}$ define a cth degree polynomial, Evaluate the polynomial at index l to obtain s_j

| | Insert each fragment from set $\{s_{j-r_1}, ..s_j\}$ in $S_{ordered}$.

| | Increment $Rank(.)$ of each fragment from set $\{s_{j-r_1}, ..s_j\} \cup S_{ordered}^{c-r_1}$ by one and update $S_{ordered}$

| | $j = j + 1$;

| | $l = l + 1$;

| **end**

end

foreach $i = r_2 + 2$ *to* x **do**

| **foreach** $k = 1$ *to* $r_1 - 1$ **do**

| | Randomly generate a secondary fragment s_j at index l

| | $j = j + 1$;

| | $l = l + 1$;

| **end**

| Points $\{p_i, s_{j-r_1-1}, s_{j-r_1}...s_{j-2}, s_{j-1}\} \cup S_{ordered}^{c-r_1+1}$ define a cth degree polynomial, Evaluate the polynomial at index l to obtain s_j

| Insert each fragment from set $\{s_{j-r_1-1}, ..s_j\}$ in $S_{ordered}$.

| Increment $Rank(.)$ of each fragment from set $\{s_{j-r_1-1}, ..s_j\} \cup S_{ordered}^{c-r_1+1}$ by one and update $S_{ordered}$

| $j = j + 1$;

| $l = l + 1$;

end

Algorithm 1. The Multi-transform(x,y,c) Algorithm

and $y = x + c$. Therefore, there are different ways of employing the Multi-share Split for a given input, each differing in the choice of x. It can be seen that the encoding offered by Multi-share Split improves the security budget by k. The proof is easily derived from the aforementioned increment property of Multi-transform. We formally describe Multi-share Split in Algorithm 2. The secondary fragments obtained from the v applications of Multi-transform are used to replace the L primary fragments.

Input: (m, n) Shamir encoding, $L, k, x : x \geq a$
Output: $(m + k, n)^{M(L,k,x)}$
$y = x + k$;
$v = \frac{L}{y}$;
if $L \bmod y == 0$ **then**
    foreach *group of x primary fragments out of some v groups* **do**
        Apply Multi-transform(x,y,k) to generate y secondary fragments
    end
end
else
    $r_1 = (L \bmod y) / v$;
    $r_2 = (L \bmod y) \bmod v$;
    foreach *group of x primary fragments out of some r_2 groups* **do**
        Apply Multi-transform$(x, y + r_1 + 1, k)$ to generate secondary fragments
    end
    foreach *group of x primary fragments out of some $v - r_2$ groups* **do**
        Apply Multi-transform$(x, y + r_1, k)$ to generate secondary fragments
    end
end
Replace the L primary fragments with the secondary fragments

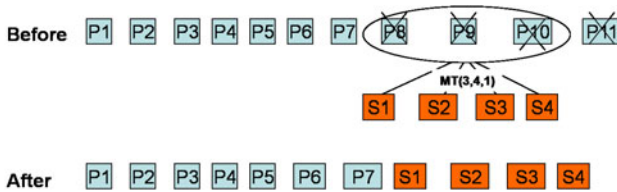---

Algorithm 2. The Multi-share Split Algorithm

Fig. 4. Increasing the security budget of a data item represented using a $(8, 11)$ Shamir encoding from 8 to 9 using Multi-share Split. We use Multi-transform(3,4,1).

Figure 4 shows an example invocation of Multi-share Split. It is instructive to compare this with the special case corresponding to $x = 1$ that we call "Uni-share Split." With Uni-share Split, two primary fragments would be permanently erased and two of them would be used for the generation of secondary fragments. On the other hand, the Multi-share Split shown in Figure 4 involves three primary fragments in generating

secondary fragments and permanently erases only one of them. We denote the configuration resulting from Multi-share Split by $(m + k, n)^{M(L,k,x)}$. Thus, Multi-share Split can results in fewer instances of permanent loss of primary fragments by controlling the number of secondary fragments generated per participating primary fragment. The value of x to be used in a Multi-share Split re-configuration method would be chosen based on the availability properties desired from the resultant configurations. We investigate this issue in the following sections.

5   FlexArchive Availability Characterization

In a regular (m, n) secret sharing configuration (e.g., Shamir's), all the n fragments are equivalent in the following sense: the configuration can tolerate the loss of up to *any* $(n - m)$ fragments to recover the secret. We refer to this as the *fragment-equivalence* property. As seen at the end of last section, Multi-share Split yields configurations that violate this property. This is because the resultant configurations have *a mixture of primary and secondary shares.* A loss of a set of secondary shares could render some other set of secondary shares useless for the secret construction. On the other hand, the loss of a primary share does not affect the reconstruction potency of any of the other shares. We attempt to analytically characterize the availability properties of configurations offered by Multi-share Split.

The heterogeneity in fragments introduced by Multi-share Split renders the estimation of the availability more complex than for configurations with fragment-equivalence. We quantify the availability for a Multi-share Split configuration using a function $CFT(R)$ defined as the (conditional) probability that the data item can be recovered given that R out of the total of N FlexArchive nodes have failed. In the current analysis, we assume that the failures of all nodes are governed by independent and identical stochastic processes. Consequently, any combination of R failed nodes among the total N shares is equally likely. It is easy to enhance this to incorporate different failure behaviors but we omit such analysis here. Furthermore, we assume that each fragment is equally likely to be placed on any of the N nodes. This is likely to correspond closely to a well load-balanced FlexArchive system. For an (m, n) fragment-equivalent configuration of an archival object stored on a system of N nodes, $CFT(.)$ is given by

$$\text{CFT(R)} = \begin{cases} 1 \text{ if } R \leq n - m \\ \sum_{i=m}^{n} \frac{\binom{i}{N-R} \times \binom{n-i}{R}}{\binom{n}{N}} & \text{otherwise} \end{cases}$$

where $\binom{i}{j}$ is the combinatorial function. For a set of fragments obtained by Multi-share Split, a closed expression for $CFT(.)$ is less straightforward. For the sake of simplicity, we focus on the special case of Uni-Share Split (which offers a lower bound on the availability for Multi-share Split in general). Furthermore, we assume that the number of system nodes $N = n$. Let us denote by $(m, n)^{U(L,k)}$ (same as $(m, n)^{M(L,k,1)}$) the configuration obtained from an $(m - k, n)$ Shamir configuration via Uni-share split by increasing the security budget by k. It consists of v sets of secondary fragments obtained by (t, t) splitting. Without loss of generality, we assume $L \bmod t = 0$ and therefore, we ignore the cases where some sets of the secondary fragments are obtained by $(t, t + r)$

splitting. The secondary fragments can be used to reconstruct any of the v original primary fragments belonging to the original $(m - k, n)$ Shamir configuration. Note that $(L - v)$ primary fragments are permanently deleted, i.e., they can never be reconstructed from the secondary fragments. We refer to the deleted (albeit only temporarily) v fragments as *imaginary primary fragments*. The number of primary fragments that were retained, denoted as *retained primary fragments*, is $w = n - L$. Clearly, if the number of lost fragments is greater than $(n - m)$, the data can not be recovered—the resultant configurations from Multi-share Split can not tolerate more fragment failures than the maximum limit allowed by the corresponding Shamir configuration.

We enumerate the possible failure scenarios of $(m, n)^{U(L,k)}$, when the number of lost fragments $i \leq (n - m)$. Suppose the loss of i fragments in the resultant configuration has effectively rendered a loss of z fragments amongst the combined set of retained and imaginary primary fragments. Only if $z \geq (n - m + k)$ does it contribute to the loss of the data. For each such value of z, there are multiple possibilities of the number of fragments lost amongst the retained and imaginary primary fragments. For example, one fragment could be lost from the set of retained primary fragments and $(z - 1)$ from imaginary primary fragments, or 2 from retained primary fragments and $(z - 2)$ from imaginary primary fragments, and so on. In general, j fragments being lost from the set of retained primary fragments could occur in $\binom{w}{j}$ ways. The other $(i - j)$ lost fragments are then secondary which have effectively resulted in the loss of $(z - j)$ imaginary primary fragments (the number of possible combinations of the lost imaginary primary fragments is $\binom{v}{z-j}$). For a fixed set of $(z - j)$ lost imaginary primary fragments, let us denote the possible number of combinations by which $(i - j)$ secondary shares could have resulted it by a function called $c(z - j, t, i - j)$. The function $c(.)$ can be recursively defined as follows:

$$c(A, B, C) = \begin{cases} \binom{B}{C} & \text{if } A = 1 \\ \sum_{i=1}^{B} \binom{B}{i} \times c(A - 1, B, C - i) & \text{otherwise} \end{cases}$$

We, therefore, have:

$$\text{1-CFT(R)} = \frac{1}{\binom{n}{i}} \times \sum_{z=n-m+k+1}^{n} \sum_{j=0}^{z} \binom{w}{j} \times \binom{v}{z-j} \times c(z - j, t, i - j).$$

6 Empirical Evaluation of FlexArchive

6.1 Experimental Setup

We implement the baseline technique and Multi-share Split in our prototype LAN-based archival storage system. The LAN consists of 41 machines with little outside contention for computing or network resources. We use a dedicated machine to host a client with the rest serving as archival stores. All machines have dual Hyper-threaded Intel Xeon processors clocked at 3.06 GHz and 1 GB RAM. The operating system running on the machines is Linux v2.6.13-1.1532. For the computations using Shamir's secret sharing algorithm, we use the *ssss* tool. In all our experiments, we use $GF(2^{16})$, i.e., the size of the finite field used for Shamir's polynomial interpolation is 16 bits.

6.2   Performance Evaluation

We consider re-configurations where the security budget of archival objects is increased by 1, 2, or 3 starting from three different configurations (6,10), (12,20) and (15,20). We consider a varying number of objects in a batch job of re-configuration where each object size is assumed to be 1MB. We vary the batch size from 1 up to 500 objects to understand the scalability with job size and processing parallelism of the different re-configuration approaches. We set the value of L, the number of original shares allowed to be accessed for Multi-share Split to its minimum value, i.e., $L = n - m + 1$.
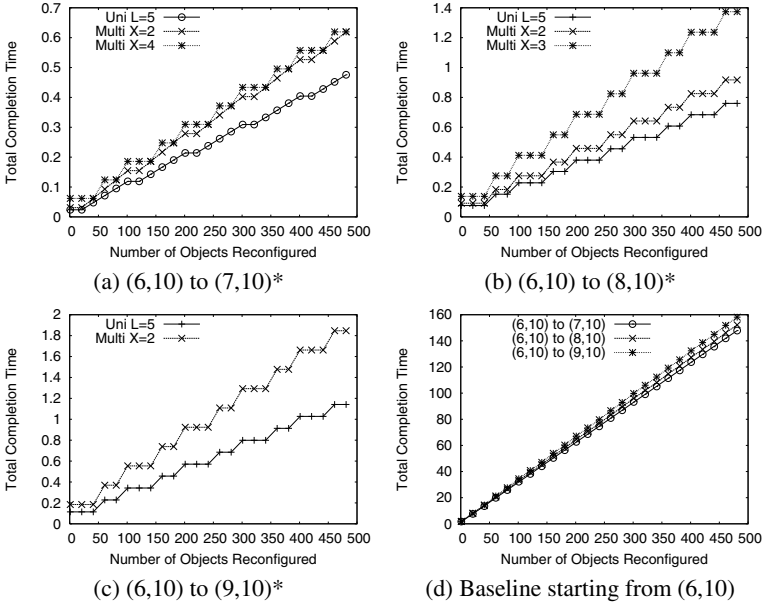

Fig. 5. Comparison of completion time (in minutes) offered by Baseline, Uni-share Split, and Multi-share Split. Recall that Uni-share Split is a special case of Multi-share Split. We use * to denote a non-fragment-equivalent configuration resulting from Multi-share Split. We report the average of several runs with the 95% confidence intervals small enough to be omitted in this figure.

We first consider re-configurations starting from (6,10). We observe that in our LAN setting when the batch size is 100 objects, the completion times of Uni-share Split for moving to (7,10)*, (8,10)*, and (9,10)* are 0.09, 0.15, and 0.22 minutes, respectively. We report these in Figures 5(a)-(d). We use "*" to indicate the non-fragment-equivalent configurations resulting from Multi-share Split. (Recall that these configurations have inferior availability properties than the corresponding fragment-equivalent configuration; we study this degradation in availability in Section 6.3). The corresponding values for Multi-share Split with the aggregation parameter $x = 2$ are 0.12, 0.18, 0.36 minutes. As seen, the completion time increases with the aggregation parameter. Most importantly, we find that Multi-share Split completes by an order of magnitude sooner

than the state-of-the-art Baseline technique. The corresponding values for the Baseline technique are 30.5, 31.4, and 32.6 minutes, respectively. As the batch size increases from 100 to 500 objects, the completion times of Uni-share Split for moving from (6,10) to (7,10)*, (8,10)* and (9,10)* increase by 0.38, 0.61, and 0.92 minutes, respectively. For Multi-share Split with $x = 2$, the corresponding increase in values are 0.49, 0.73, and 1.48 minutes, respectively. On the other hand, the completion times for Baseline increase by 122.0, 125.6, and 130.4 minutes, respectively. We observe that Baseline also scales very poorly with the batch size. This supports our proposition that our Multi-share Split can be deployed to achieve significantly faster re-configuration times compared to the Baseline. Multi-share Split exhibits increased parallelism compared to Baseline and is not limited by the additional synchronization phase of fragment reconstruction required by Baseline.

Finally, we conduct measurements of completion time for starting configurations of (12,10) and (15,20) and make similar observations about the significant speedup offered by Multi-share Split. Due to space constraints, we only mention two salient observations here. First, our experiments yield similar observations as above about the significant speedup offered by Multi-share Split. Second, we find that the completion times for the starting configuration of (12,20) are much larger compared to that of (15,20) for Multi-share Split, whereas the opposite is true for Baseline. This is because Multi-share Split operates only on $(n - m + 1)$ original fragments whereas Baseline uses m fragments.

(a) (6,10) to (7,10)*   (b) (6,10) to (8,10)*   (c) (6,10) to (9,10)*
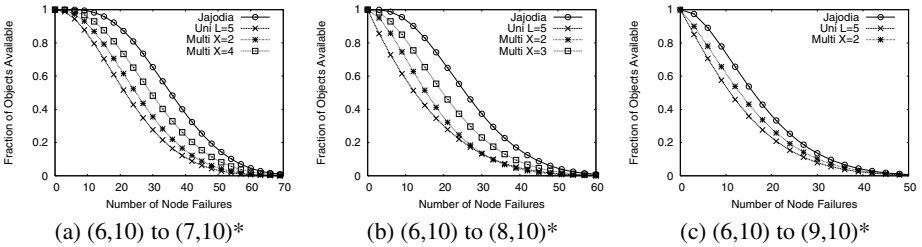
Fig. 6. Comparison of availability of configurations obtained using Baseline, Uni-share Split, and Multi-share Split. We use the $CFT(.)$ as our measure of availability. Recall that Uni-share Split is a special case of Multi-share Split. We use * to denote a non-fragment-equivalent configuration resulting from Multi-share Split.

6.3   Availability Evaluation: Conditional Fault Tolerance

We evaluate the availability resulting after a re-configuration using $CFT(.)$ as definition in Section 5. We measure the $CFT(.)$ of resultant configurations obtained by incrementing the security budget of three fragment-equivalent configurations (6,10), (12,20) and (15,20). As before, due to limited space, we present the results for only (6,10). Figures 6(a)-(c) show $CFT(.)$ for configurations obtained by incrementing the security budget of (6,10) configuration. In all the cases, we assume the stringiest budget in terms of the number of original fragments that are permitted to be accessed during the re-configuration, i.e., $L = n - m + 1$. In Figure 6(a), we compare the $CFT(.)$ of (7,10)* configuration achieved using Uni- and Multi-share Split techniques with

Table 1. Three failure traces used in our study

Trace	Duration	Nature of nodes	No. of nodes	Probe interval
WS	09/2001 to 12/2001	Public Web servers	130	10 mins
PL	03/2003 to 06/2004	PlanetLab nodes	277 on avg	15 to 20 mins
RON	03/2003 to 10/2004	RON testbed	30 on avg	1 to 2 mins

the (7,10) configuration achieved using Baseline when the security budget increment $k = 1$. Figures 6(b),(c) report $CFT(.)$ for final configurations (8,10)* and (9,10)*, respectively.

As expected, we observe that the $CFT(.)$ of configurations using Multi-share Split lag behind those using Baseline. The gap initially increases when the security budget increment k is raised from 1 to 2 but it tends to become smaller for $k \geq 3$. We also observe that $CFT(.)$ using Multi-share Split improves with the aggregation parameter x. In particular, $CFT(.)$ using Multi-share Split with $x > 1$ are superior to those using Uni-share Split. One might argue that a FlexArchive administrator should always prefer Multi-share Split with the largest possible x as it provides the best CFT. However, as already seen this improvement in availability comes with an increased completion time for re-configurations. For example, the completion times for Multi-share Split with $x > 1$ are larger than those for Uni-share Split. In fact, the completion time increases with x in general. This is because of the following reasons. In Uni-share Split, there is inherent parallelism in the sense that the (t, t) splittings are done in parallel on the nodes storing the primary shares. The amount of parallelism is reduced as we move to Multi-share Split with larger values of x. Also, the computation involved in transforming primary fragments to secondary fragments is higher in Multi-share Split than in Uni-share Split.

6.4   Availability Evaluation: Failure Traces

In our availability characterization so far, we have assumed failure independence of nodes. However, the failure correlation between nodes may not be completely absent in real world systems, e.g., due to nodes being managed using common management strategies, system software bugs, DDos attacks, etc. We study three real-world traces with failure information (described in Table 1) to evaluate the impact of realistic failure patterns. WS trace [15] is intended to be representative of public-access machines that are maintained by different administrative domains, while PL and RON traces [15] potentially describe the behavior of a centrally-administered distributed system that is used mainly for research purposes as well as for a few long-running services. A probe interval is a complete round of all pair pings; a node is declared as failed if none of the other nodes can ping it during that interval.

We estimate the "availability of data," for a given configuration as follows. First we assume that the nodes used to store the fragments of an object are chosen uniformly at random from among the nodes associated with the trace. We then determine the probability that the object can be recovered within every probe interval. Availability of data is then expressed using the average of these probability values over the entire trace. With our assumption on placement of fragments, availability of data is given by $\sum_{R=1}^{N}$
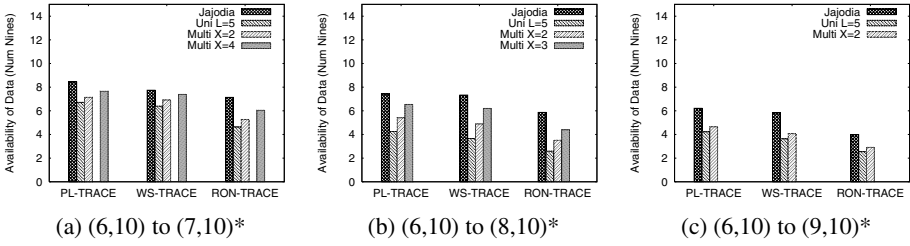
Fig. 7. Availability of configurations resulting from various re-configuration techniques. The initial configuration is fragment-equivalent (6,10). Confidence intervals, found to be very small, have been omitted in the figures.
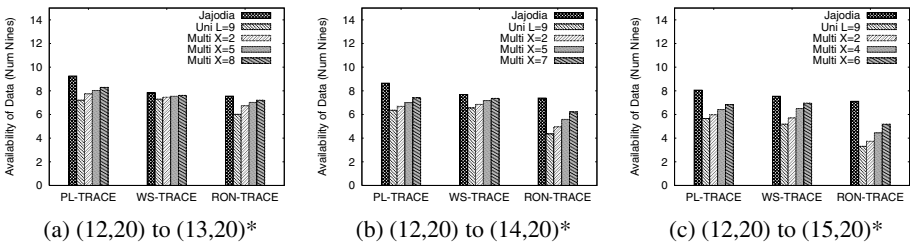

Fig. 8. Availability of configurations resulting from various re-configuration techniques. The initial configuration is fragment-equivalent (12,20). Confidence intervals, found to be very small, have been omitted in the figures.
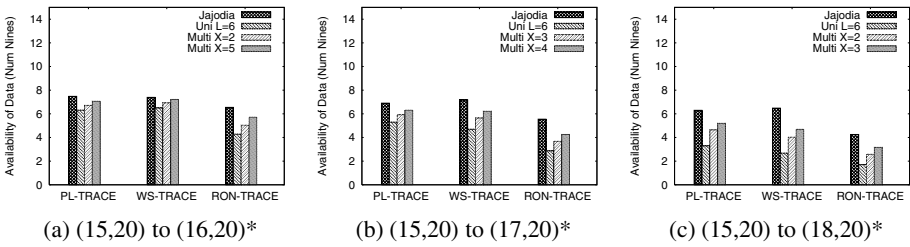

Fig. 9. Availability of configurations resulting from various re-configuration techniques. The initial configuration is fragment-equivalent (15,20). Confidence intervals, found to be very small, have been omitted in the figures.

$(CFT(R) \times FSF(R))$. Here, $CFT(.)$ is the conditional fault tolerance computed for a resultant configuration with the number of system nodes N set to the total number of nodes in a given failure trace. The function $FSF(R)$ represents "failure size frequency" and is the fraction of instances in the trace where exactly R nodes fail. Figures 7- 9 show the availability for different resultant reconfigurations and traces. The "Num Nines" on the y-axes indicate that the availability values are extremely close to one, hence we use the number of leading nines after the decimal point in our presentation. From these

figures, we observe results similar to those seen in Section 6.3 where as we move from Uni-share Split to Multi-share Split with higher values of the aggregation parameter, the availability of data increases and approaches that for the corresponding fragment-equivalent configuration.

7   Concluding Remarks

The motivation for this work stems from the complex trade-offs between the confidentiality, availability, and performance that modern storage systems need to address. We argued that secret sharing-based encoding of data offers two desirable properties—(i) information-theoretically provable guarantees on data confidentiality unlike conventional encryption, and (ii) quantifiable guarantees on the availability of encoded data—that make it particularly suitable for the design of increasingly popular and important distributed archival data stores. These guarantees, however, come at the cost of increased resource consumption during reads/writes and hence could degrade the performance offered to clients. Consequently, we argued, it is desirable that such a storage system employ techniques that could dynamically transform data configuration to operate the store within required confidentiality, availability, and performance regimes (or budgets) despite changes to the operating environment. Since state-of-the-art transformation techniques suffer from prohibitive data transfer overheads, we developed a middleware that facilitates dynamic data re-configuration. at significantly lower overheads. Using this, we proposed the design and operation of a secure, available, and tunable distributed archival store, called FlexArchive, spanning a wide-area network. Using a combination of analysis and empirical evaluation, we demonstrated the feasibility of FlexArchive.

Acknowledgements

References

1. Bhagwan, R., Tati, K., Cheng, Y., Savage, S., Voelker, G.M.: Totalrecall: Systems support for automated availability management. In: Proceedings of the Usenix Symposium on Networked Systems Design and Implementation (2004)
2. Blakley, B., Blakley, G.R., Chan, A.H., Massey, J.L.: Threshold schemes with disenrollment. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 540–548. Springer, Heidelberg (1993)
3. Cachin, C.: On-line secret sharing. In: Boyd, C. (ed.) Cryptography and Coding 1995. LNCS, vol. 1025. Springer, Heidelberg (1995)
4. Desmedt, Y., Jajodia, S.: Redistributing secret shares to new access structures and its applications. Technical Report ISSE TR-97-01, George Mason University, Fairfax, VA (July 1997)
5. Adya, A., et al.: Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In: Proceedings of the 5th Symposium on Operating Systems Design and Implementation, OSDI (2002)

6. Frankel, Y., MacKenzie, P.D., Yung, M.: Adaptive security for the additive-sharing based proactive RSA. In: Kim, K.-c. (ed.) PKC 2001. LNCS, vol. 1992, pp. 240–263. Springer, Heidelberg (2001)
7. Goodson, G.R., Wylie, J.J., Ganger, G.R., Reiter, M.K.: Efficient byzantine-tolerant erasure-coded storage. In: Proceedings of the 2004 International Conference on Dependable Systems and Networking, DSN 2004 (June 2004)
8. Haeberlen, A., Mislove, A., Druschel, P.: Highly durable, decentralized storage despite massive correlated failures. In: Proceedings of the 2nd symposium on Networked Systems Design and Implementation (NSDI) (May 2005)
9. Hey, T., Trefethen, A.: Data deluge - an e-science perspective. In: Grid Computing - Making the Global Infrastructure a Reality (January 2003)
10. Iyengar, A., Cahn, R., Garay, J.A., Jutla, C.: Design and implementation of a secure distributed data repository. In: Proceedings of the 14th IFIP International Information Security Conference (SEC 1998) (September 1998)
11. Kallahala, M., Riedel, E., Swaminathan, R., Wang, Q., Fu, K.: Plutus: Scalable secure file sharing on untrusted storage. In: Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST) (March 2003)
12. Keeton, K., Santos, C., Beyer, D., Chase, J., Wilkes, J.: Designing for disasters. In: Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST) (April 2004)
13. Kotla, R., Alvisi, L., Dahlin, M.: Safestore: A durable and practical storage system. In: Proceedings of the USENIX Annual Technical Conference (2007)
14. Miller, E.L., Long, D.D.E., Freeman, W.E., Reed, B.C.: Strong security for network-attached storage. In: Proceedings of the 2002 conference on File and Storage Technologies, FAST (2002)
15. Nath, S., Yu, H., Gibbons, P.B., Seshan, S.: Subtleties in tolerating correlated failures in wide-area storage systems. In: Proceedings of the 3rd conference on 3rd Symposium on Networked Systems Design and Implementation, NSDI (2006)
16. Quinlan, S., Dorward, S.: A new approach to archival storage. In: Proceedings of the Conference in File and Storage Technologies, FAST (2002)
17. Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., Kubiatowicz, J.: Pond: the oceanstore prototype. In: Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST) (March 2003)
18. Santry, D.S., Feeley, M.J., Hutchinson, N.C., Veitch, A.C., Carton, R.W., Ofir, J.: Deciding when to forget in the elephant file system. In: Proceedings of the 17th ACM Symposium on Operating Systems Principles, SOSP (1999)
19. Shamir, A.: How to share a secret. Communications of the ACM (November 1979)
20. Stanton, P., Yurcik, W., Brumbaugh, L.: Protecting multimedia data in storage: A survey of techniques emphasizing encryption. In: International Symposium Electronic Imaging/Storage and Retrieval Methods and Applications for Multimedia (2005)
21. Storer, M.W., Greenan, K., Miller, E.L.: Long-term threats to secure archives. In: Proceedings of the Workshop on Storage Security and Survivability (2006)
22. Storer, M.W., Greenan, K.M., Miller, E.L., Voruganti, K.: Potshards: Secure long-term storage without encryption. In: Proceedings of the USENIX Annual Technical Conference (2007)
23. Subbiah, A., Blough, D.M.: An approach for fault tolerance and secure data storage in collaborative work environments. In: Proceedings of the 2005 ACM Workshop on Storage Security and Survivability (November 2005)
24. Wong, T.M., Wang, C., Wing, J.M.: Verifiable secret distribution for threshold sharing schemes. Technical Report. CMU-CS-02-114-R, Carnegie Mellon University (October 2002)
25. Wylie, J.J., Bigrigg, M.W., Strunk, J.D., Ganger, G.R., Kilicotte, H.P.: K Khosla. Survivable storage systems. IEEE Computer (August 2000)