Selecting the Right Data Distribution Scheme for a Survivable Storage System

Jay J. Wylie, Mehmet Bakkaloglu, Vijay Pandurangan,
Michael W. Bigrigg, Semih Oguz, Ken Tew, Cory Williams,
Gregory R. Ganger, Pradeep K. Khosla

May 2001

CMU-CS-01-120

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Survivable storage system design has become a popular research topic. This paper tackles the difficult problem of reasoning about the engineering trade-offs inherent in data distribution scheme selection. The choice of an encoding algorithm and its parameters positions a system at a particular point in a complex trade-off space between performance, availability, and security. We demonstrate that no choice is right for all systems, and we present an approach to codifying and visualizing this trade-off space. Using this approach, we explore the sensitivity of the space to system characteristics, workload, and desired levels of security and availability.

This research is part of the PASIS project [40, 48] at Carnegie Mellon University.

This work is partially funded by DARPA/ATO's Organically Assured and Survivable Information Systems (OASIS) program (Air Force contract number F30602-99-2-0539-AFRL). We thank the members and companies of the Parallel Data Consortium (at the time of this writing: EMC Corporation, Hewlett-Packard Labs, Hitachi, IBM Corporation, Intel Corporation, LSI Logic, Lucent Technologies, Network Appliances, PANASAS, Platys Communications, Seagate Technology, Snap Appliances, Sun Microsystems, Veritas Software Corporation) for their insights and support.

Keywords: survivability, security, storage systems, distributed file systems, wide-area storage

1 Introduction

Digital information is a critical resource, creating a need for distributed storage systems that provide sufficient data availability and data security in the face of failures and malicious attacks. Many research groups [13, 14, 17, 24, 37, 38, 40, 41] are now exploring the design and implementation of such *survivable storage systems*. These systems build on mature technologies from decentralized storage systems [1, 3, 8, 20, 33, 34] and also share the same high-level architectures. In fact, development of survivable storage with the same basic architecture was pursued over 15 years ago [12, 15, 16]. The challenge now, as it was then, is to achieve acceptable levels of performance and manageability. Moreover, a means to evaluate survivable storage systems is required to facilitate the design of these systems.

One key to maximizing survivable storage performance is mindful selection of the *data distribution scheme*. A data distribution scheme consists of a specific algorithm for data encoding & partitioning and a set of values for its parameters. There are many algorithms applicable to survivable storage, including encryption, replication, striping, erasure-resilient coding, secret sharing, and various combinations. Each algorithm has one or more tunable parameters. The result is a large toolbox of possible schemes, each offering different levels of performance (throughput), availability (probability that data can be accessed), and security (effort required to compromise the confidentiality or integrity of stored data). For example, replication provides availability at a high cost in network bandwidth and storage space, whereas short secret sharing provides availability and security at lower storage and bandwidth cost but higher CPU utilization. Likewise, selecting the number of shares required to reconstruct a secret-shared value involves a trade-off between availability and confidentiality: if more machines must be compromised to steal the secret, then more must be operational to provide it legitimately. Secret sharing schemes and other data distribution algorithms are described in Section 2.

No single data distribution scheme is right for all systems. Instead, the right choice for any particular system depends on an array of factors, including expected workload, system component characteristics, and desired levels of availability and security. Unfortunately, most system designs appear to involve an *ad hoc* choice, often resulting in a substantial performance loss due to missed opportunities and over-engineering.

This paper promotes a better approach to selecting the data distribution scheme. At a high level, this new approach consists of three steps: enumerating possible data distribution schemes ($<algorithm, parameters>$ pairs), modeling the consequences of each scheme, and identifying the best-performing scheme for any given set of availability and security requirements. The surface shown in Figure 1 illustrates one result of the approach. Generating such a surface requires codifying each dimension of the trade-off space such that all data distribution schemes fall into a total order. The surface serves two functions: (1) it enables informed trade-offs among security, availability, and performance, and (2) it identifies the best-performing scheme for each point in the trade-off space. Specifically, the surface shown represents the performance of the best-performing scheme that provides at least the corresponding levels of availability and security. Many schemes are not best at any of the points in the space and as such are not represented on the surface.

This paper demonstrates the feasibility and importance of careful data distribution scheme choice. The results show that the optimal choice varies as a function of workload, system characteristics, and the desired levels of availability and security. The results show that minor ($\approx 2\times$) changes in these determinants have little effect, which means that the models need not be exact to be useful. Importantly, the results also show that large changes, which would correspond to distinct systems, create substantially different trade-off spaces and best choices. Thus, failing to examine the trade-off space in the context of one's system can yield both poor performance and unfulfilled requirements. With sensitivity studies and survivable storage system examples from the literature, we identify interesting trends and design points.

The remainder of this paper is organized as follows. Section 2 discusses survivable storage system design and data distribution scheme options. Section 3 describes how we codify the trade-off space. Section 4 describes PASIS [40, 48], our prototype survivable storage system, and its role in validating our performance model. Section 5 explores the trade-off space and its sensitivity to model inaccuracies, system components, and expected workloads. Section 6 describes current and past survivable storage system research, identifying insights from our explorations that could enhance their designs. Section 7 summarizes this papers contributions.

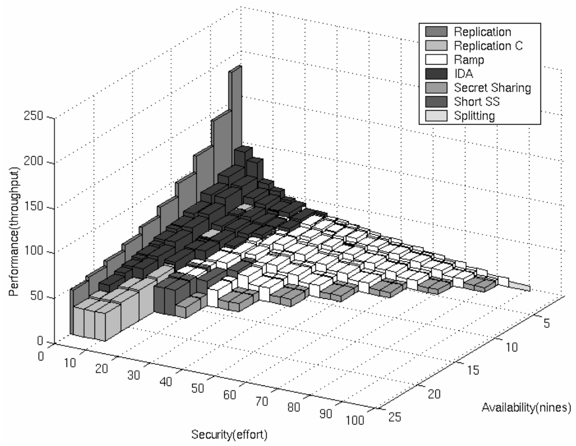

Figure 1: **Data distribution scheme selection surface plotted in trade-off space.** The trade-off space has performance, availability, and security axes. Performance is quantified as the number of 32 KB requests per second that can be satisfied for a single client; only the best-performing scheme that provides at least the given security and availability levels is shown, resulting in a monotonic decrease along those axes. Availability is the probability that stored data can be accessed. Security is the effort required to compromise either the confidentiality or integrity of stored data. Section 2 describes the data distribution algorithms listed in the legend. Section 3 describes the axes in detail.

2 Survivable Storage Systems

Survivable systems operate from the fundamental design thesis that no individual service, node, or user can be fully trusted; having some compromised entities must be viewed as the common case rather than the exception. Survivable storage systems must encode and distribute data across independent storage nodes, entrusting the data's persistence to sets of nodes rather than to individual nodes. If confidentiality is required, unencoded data should not be stored directly on storage nodes; otherwise, compromising a single storage node enables an attacker to bypass access-control policies.

Prior work in cluster storage systems [1, 3, 8, 20, 33, 34] provides much insight into how to efficiently decentralize storage services while providing a single, unified view to applications. Figure 2 illustrates the basic decentralized storage architecture. In most cases, some intermediary software is responsible for translating between the unified view and the decentralized reality. This piece of software may execute directly on each client system [1, 3, 20, 33], on storage nodes identified as leaders [8, 34], or at intermediary nodes [1, 3].

In most cluster storage systems, the data and some form of redundancy information are striped across storage nodes. Survivable storage systems differ from decentralized storage systems mainly in the encoding mechanism used (i.e., the data distribution scheme). The data distribution scheme enables the storage system to survive both failures and compromises of storage nodes. Each scheme (a $<$*algorithm, parameters*$>$ pair) offers a different level of performance, availability and security. Little understanding exists of the large trade-off space that the schemes occupy in practice.

This paper takes a step towards understanding the relative merits of different data distribution schemes by examining them in the context of the trade-off space. The remainder of this section describes various data distribution algorithms and the parameters that specify each algorithm. There are several issues involved with constructing a complete survivable storage system that are not central to the results and insights of this paper. Specifically, this paper argues for and provides an approach for making mindful selections of the data distribution scheme, but explicitly tries to do so while remaining impartial on issues that are largely orthogonal. Examples of such issues include metadata and naming mechanisms, client node authentication, and consistency given concurrent writers.

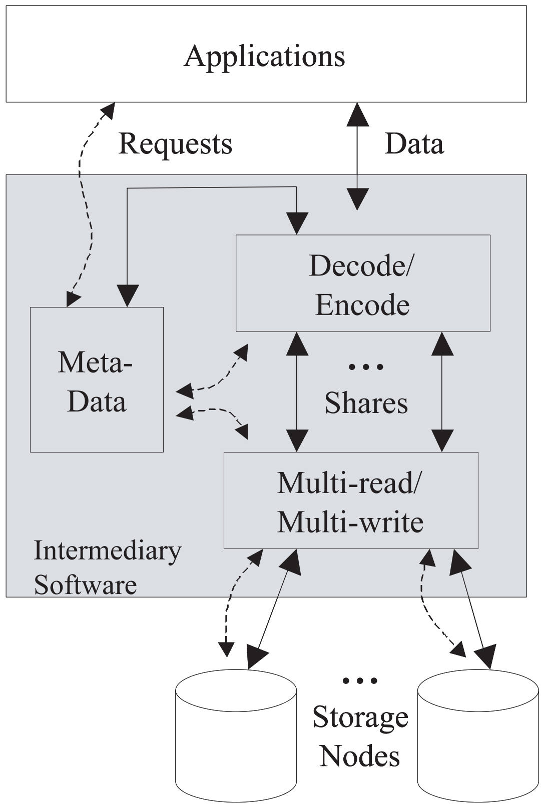

Figure 2: **Generic decentralized storage architecture.** Intermediary software translates the applications' unified view of storage to the decentralized reality. Solid lines trace the data path. Dashed lines trace the meta-data path. Applications read and write blocks to the intermediary software. Encoding transforms blocks into shares (decoding does the reverse). Sets of shares are read from (written to) storage nodes. Intermediary software may run on clients, leader storage nodes, or at some point in between.

2.1 Threshold algorithms

There is a wide array of data distribution algorithms, including encryption, replication, striping, erasure-resilient coding, information dispersal, and secret sharing. *Threshold algorithms*, characterized by three parameters (p, m, and n), represent a large set of these algorithms. In a p-m-n threshold scheme, data is encoded into n shares such that any m of the shares can reconstruct the data and less than p reveal no information about the encoded data. Thus, a stored value is available if at least m of the n shares can be retrieved. Attackers must compromise at least p storage nodes before it is even theoretically possible to determine any part of the encoded data.

Table 1 lists some p-m-n threshold schemes that have more familiar names. Perhaps the simplest example is n-way replication, which is a 1-1-n threshold scheme. That is, out of the n replicas that are stored, any single replica provides the original data ($m = 1$), and each replica reveals information about the encoded data ($p = 1$). Another simple example is decimation (or striping, as in disk arrays), wherein a large block of data is partitioned into n sub-blocks, each containing $1/n$ of the data (so, $p = 1$ and $m = n$). At the other end of the spectrum is "splitting", an n-n-n threshold scheme that consists of storing $n-1$ random values and one value that is the exclusive-or of the original value and those $n-1$ values; $p = m = n$ for splitting, since all n shares are needed to recover the original data. Replication, decimation, and splitting schemes have a single tunable parameter, n, which affects their place in the trade-off space.

With more CPU-intensive mathematics, the full range of p-m-n threshold schemes becomes available. For example, secret sharing schemes [4, 27, 43] are m-m-n threshold schemes. Shamir's implementation of secret sharing is based on interpolating points on a polynomial in a finite field [43]. The secret value along with $m-1$ randomly generated values uniquely determine the encoding polynomial of order $m-1$. Each

Parameters	Description
1-1-n	Replication
1-n-n	Decimation (Striping)
n-n-n	Splitting (XORing)
1-m-n	Information Dispersal
m-m-n	Secret Sharing
p-m-n	Ramp Schemes

Table 1: **Example p-m-n threshold schemes.**

share is generated by evaluating the polynomial at distinct points (distinct from other shares and the point containing the secret value).

Rabin's information dispersal algorithm, a 1-m-n threshold scheme, uses the same polynomial-based math as Shamir's secret sharing, but no random numbers [42]; m secret values are used to determine the unique encoding polynomial. Thus, each share reveals partial information about the m simultaneously encoded values, but the encoding is much more space-efficient. Ramp schemes [5] are p-m-n threshold schemes, and they can also be implemented with the same polynomial-based math. The points used to uniquely determine the encoding polynomial are $p-1$ random values and $m-(p-1)$ secret values. Ramp schemes thus offer information-theoretic confidentiality of up to $p-1$ shares. They are also more space-efficient than secret sharing (so long as $m>p$). For $p = 1$, ramp schemes are equivalent to information dispersal; for $p = m$, they are equivalent to secret sharing. Threshold algorithms can be implemented in ways other than the polynomial method summarized in this section. For example, Blakley proposed implementing secret sharing with intersecting hyper-planes [4], and Luby proposed implementing information dispersal with Tornado codes [35].

Parameter options for p-m-n threshold schemes expose a large space of possible data distribution schemes. In fact, there are on the order of N^3 different options to consider given N storage nodes. Each scheme in this space offers different levels of availability, confidentiality, CPU costs, and storage requirements (which also translates into network bandwidth requirements). For example, as n increases, information availability increases (it is more probable that m shares are available), but the amount of storage required increases (more shares are stored) and confidentiality decreases (more shares are available to steal). As m increases, the storage space required decreases (a share's size is proportional to $1/(m-(p-1))$), but so does its availability (more shares are required to reconstruct the original object). Also, as m increases, each share contains less information; this may increase the number of shares that must be captured before an intruder can reconstruct a useful portion of the original object. As p increases, the information system's confidentiality increases, but the storage space required also increases. With such a wide array of options, selecting the most appropriate data distribution scheme for a given environment is far from trivial.

2.2 Other algorithms

There are also important data distribution algorithms outside the class of p-m-n threshold schemes. Notably, encryption is a common approach to protecting the confidentiality of information. Symmetric key encryption (e.g., triple-DES, AES) is a data distribution algorithm characterized by a single parameter—key length. Hybrid data distribution algorithms can be constructed by combining the algorithms already discussed. For example, many survivable storage systems combine replication with encryption to address availability and security, respectively. Security in such a system hinges both upon how well the encryption keys are protected and upon the difficulty of cryptanalysis. As another example, *short secret sharing* encrypts the original value with a random key, stores the encryption key using secret sharing, and stores the encrypted information using information dispersal [30]. Short secret sharing algorithms have three parameters: m, n, and k (key length). Public key cryptography (e.g., RSA) can be used instead of symmetric key cryptography to protect information confidentiality. The management of cryptographic keys must be addressed in the design of a system that uses cryptography; symmetric key and public key cryptography require different key management strategies. Finally, compression algorithms (e.g., Huffman coding) can be used before other data distribution algorithms to reduce the size of the data that must be encoded.

Another important type of data distribution algorithm provides integrity verification. Cryptographic hash algorithms (e.g., MD5, SHA-1) can be used to add digests to data before it is encoded with another algorithm. The hash of the decoded data can be compared with the digest to verify the decoded data's integrity. Digests can also be generated for shares resulting from an encoding (e.g., distributed fingerprints [29]), allowing integrity to be verified prior to decoding the data. Hash algorithms are parameterized by the hash size, and they must either be encoded with or stored separately from the data in order to be effective. Digital signatures (e.g., DSA) can provide similar integrity guarantees as hash algorithms.

Two classes of integrity algorithms are often used to build authentication and directory services that protect the integrity of meta-data. The first class are agreement algorithms [32] such as the Byzantine fault tolerant library [9]. Quorum systems [19, 36], which are a superset of voting algorithms [21, 45], are the second class. Also, there are integrity algorithms that work exclusively with threshold algorithms. In schemes where m is less than n, excess shares can be used during decode (different permutations of m shares are used for validation). Secret sharing schemes can also be modified directly to offer a probabilistic guarantee of cheater detection [46].

3 The Trade-off Space

The algorithms described in Section 2, together with the degrees of freedom given by their parameters, provide thousands of data distribution schemes for survivable storage systems. Thoughtfully selecting one scheme from this set requires the ability to evaluate their relative merits and, further, to do so in the context of the target environment. This section describes our approach to comparing the performance, availability, and security of various schemes.

3.1 Evaluating availability

The substantial body of prior work in building highly-available systems [23, 44] provides a clear metric for evaluating information availability: the probability that a desired piece of information can be accessed at any given point in time. Assuming uncorrelated failures, this probability can be computed from the probabilities that required system components are available. For example, a p-m-n threshold scheme requires at least m of the n storage nodes containing shares to be operational to perform a read. If f_{node} is the probability that a storage node has failed or is otherwise unavailable, then the availability of the stored information is

$$Availability_{read} = \sum_{i=0}^{n-m} \binom{n}{i} (f_{node})^i (1 - f_{node})^{n-i}$$

For writes, the computation of availability depends upon the system's design. A system could require that all of n specific nodes be operational for a write to succeed. This approach clearly has poor availability characteristics. To improve write availability in a system with $N > n$ storage nodes, the write operation can attempt to write shares to different storage nodes until it has completed n writes. Alternatively, a system could allow a write to complete when fewer than n shares have been written. This requires more work during failure recovery or reduces read availability of the stored data (because n is effectively lower for that data). We calculate write availability using the last approach. Thus, m storage nodes of the N storage nodes in the system must be available for a write to be performed, and write availability is

$$Availability_{write} = \sum_{i=0}^{N-m} \binom{N}{i} (f_{node})^i (1 - f_{node})^{N-i}$$

Availability requirements for storage systems tend to be quite high. A popular manner for discussing high-availability values is in terms of "nines," referring to the number of nines after the decimal point in the availability value before a digit other than nine. For example, a low availability value of 0.993 has just two nines, whereas a high availability value of 0.99999996 has seven nines. It is worth noting that failures are not always uncorrelated, as is generally assumed in availability computations. For survivable systems in particular, denial-of-service (DoS) attacks can induce highly-correlated failures, bringing into question the value of this metric. We believe that the relative availability levels computed for different schemes provide insight even with DoS attacks. Although the absolute values may not always be meaningful, relative values certainly are when comparing data distribution schemes for a given set of system components. In particular, availability increases only when more options are available for servicing requests. Thus, a higher availability value means that a DoS attack must eliminate a larger set of storage nodes.

3.2 Evaluating security

By far, security is the hardest of the three dimensions to evaluate, as there are no proven metrics or even a mature body of research from which to draw. Our initial plan was to reuse the mathematics of fault tolerance, counting how many storage nodes must be compromised to bypass confidentiality or integrity. However, this approach raised significant problems: First and foremost, it relies upon having a "probability of being compromised" with which to compute security values. Unfortunately, we are aware of no reliable way to obtain or even estimate such a value. Further, using such a value would be questionable, because security problems experienced by nodes in a distributed system are expected to be highly correlated; when the system is under attack, the probability value would go up. Another problem with evaluating security in terms of a probability of a node being compromised is that it is a useful measure for only a subset of the data distribution algorithms (threshold algorithms). For example, it ignores the additional confidentiality provided by encryption.

Our current approach to evaluating security focuses on the *effort* (E) required for an active foe to compromise the security of the system. For example, for a n-n-n threshold scheme, confidentiality can be compromised by breaking into all n heterogenous storage nodes or by compromising the authentication system:

$$E_{Conf} = \min\left[E_{Auth}, (n \times E_{BreakIn})\right]$$

Extending this example, assume that the data is also encrypted and that the names of shares reveal no information about the encoded data. The attacker has many paths to the data—attempt cryptanalysis or steal the encryption key, attempt combining shares in many permutations or compromise the directory service. Assuming that an attacker is going to take the easiest path to the data:

$$\begin{aligned} E_{Conf} &= \min\left[E_{Auth}, (n \times E_{BreakIn})\right. \\ &\quad + \min\left[E_{Cryptanalysis}, E_{StealKey}\right] \\ &\quad \left.+ \min\left[E_{IdentifyShares}, E_{StealNames}\right]\right] \end{aligned}$$

In this paper, we focus strictly on the security of the storage system; attacks on the authentication service and directory service are not considered further. Thus, we use only two values in our security model: $E_{BreakIn}$ and $E_{CircumventCryptography}$. These values are dimensionless. Thus, the unit of the security axis is "effort units," and security values across all runs have been normalized on a scale that ranges from 0 to 100.

The first term, $E_{BreakIn}$, is the effort required to steal a piece of data from a storage node. The second term, $E_{CircumventCryptography}$, is the effort required to circumvent cryptographic confidentiality. It is a coarse metric that includes cryptanalysis, key guessing, and the theft of decryption keys. The confidentiality of an encrypted replica is thus the summation of these two terms: one encrypted replica must be stolen and then the encryption must be broken. Alternatively, the confidentiality of secret sharing is solely a function of the first term (i.e., $m \times E_{BreakIn}$)—a total of m shares must be stolen to compromise the confidentiality of data, thus the effort is m times the effort to steal a piece of data. We calculate the effort to compromise the confidentiality of information dispersal as a weighted sum of the information content of the shares, where the weight is proportional to the number of shares that must be stolen to decode. This heuristic sets the confidentiality of information dispersal above that of replication, below that of secret sharing, and higher as m increases. We assume that $E_{BreakIn}$ is constant for all storage nodes (i.e., distinct attacks of equivalent difficulty are necessary to compromise each storage node). For this assumption to be true storage nodes must be heterogenous. Beyond this, we hypothesize that, as n increases, the likelihood that an attacker can find a storage node that they can compromise increases; as such, confidentiality should decrease as n increases. We do not currently consider n in the calculation of security. A more detailed security model can be implemented, however these simple models capture the major features of the algorithms we are currently investigating.

Security is often defined as availability, confidentiality, and integrity. In our analysis, we have distinguished availability from the two other security characteristics because there is a trade-off between it and them. We focus our security axis on confidentiality, since there appears to be agreement in the community that cryptographic hashes are the right way to protect integrity, and we have found no contradictory evidence. The hashes can be generated from the cleartext or the ciphertext, and they can be stored with data shares, encoded into the name, or stored in the directory service. All of these increase integrity significantly, usually well beyond other effort levels.

Although effort terms are difficult to quantify, we believe that effort-based evaluation focuses the designer's attention on the right thing—raising the security bar "high enough." As with availability, the relative merits of schemes can be compared usefully even if the absolute effort quantities are inaccurate. Further, other security engineering research projects are now addressing the problem of measuring security and doing so in terms of effort [39]. As better effort models become available, they can be modularly inserted into the trade-off exploration approach.

3.3 Evaluating performance

As with availability, performance metrics and evaluation techniques of distributed systems are part of a mature body of prior work [26, 28]. The main issue faced in creating a general tool is balancing detail, which should yield greater accuracy, with generality in the performance models. We use a simple, abstract system model to predict the relative performance of different data distribution options. Like the general decentralized-storage model described in Section 2, we intend for it to represent a wide array of survivable storage system designs. The model includes three parts: CPU time for encode or decode in the intermediary software, network bandwidth for delivering the shares, and storage node response times.

CPU Time. The encode and decode operations involved with any data distribution scheme require CPU time. There are orders of magnitude differences between the CPU times for different schemes. For example, Figure 3 shows the CPU cost for encoding a 32 KB block for four p-m-n threshold schemes for all possible parameter selections with $n \leq 25$. Notice the large differences between the shapes of the performance curves and the times at the same m and n for different schemes.

We have constructed and calibrated simple models for the following data distribution algorithms: ramp schemes, secret sharing, information dispersal, replication, decimation, splitting, short secret sharing, encryption, and hash algorithms. The models require CPU measurements of a few key primitives: polynomial interpolation of order $m-1$, random number generation, triple-DES encryption, and MD5 hash generation. Given the requisite measurements, the models can predict the CPU time required for an encode or decode operation for any of the data distribution schemes with at least 90% accuracy.

Network Bandwidth. The amount of data that must pass between the client and the storage nodes depends directly on the data distribution scheme. For any of the p-m-n threshold schemes, the network bandwidth required depends on the read-write ratio and the values of p, m, and n. Each share's size can be computed as the original data size multiplied by $1/(m-(p-1))$. For a write, we assume that all n shares must be updated. For a read, we assume that only m shares are fetched by default.

We model the network bandwidth with a distribution, indicating the probability of a given bandwidth during any period of time, assuming that a single bottleneck link determines the aggregate bandwidth. Although imprecise, we believe that this allows the first-order effects of concern to be captured, when combined with the storage node latencies discussed below. For a local-area network or dial-up client, we expect this bottleneck link to be at the client's network interface card. For a wide-area system, we expect this link to be at the edge router through which the client interacts with the wide-area network. Network congestion would appear in this very simple model as a reduction of available bandwidth (if consistent) or an increase in variability (if sporadic).

Storage Node Latency. A read or write request to a storage node involves work at the storage node in addition to moving data over the network. This is observed at the client as a response time latency that includes both delays, but accurate modeling requires separating these delays and recombining them appropriately. In particular, the time for an operation that simultaneously writes data to n nodes must capture both the shared bandwidth and concurrent latency aspects.

We model storage node latency as a random variable whose mean and variance depend on the operation type (read or write) and the request size. Dependence on the former is expected, and we found dependence

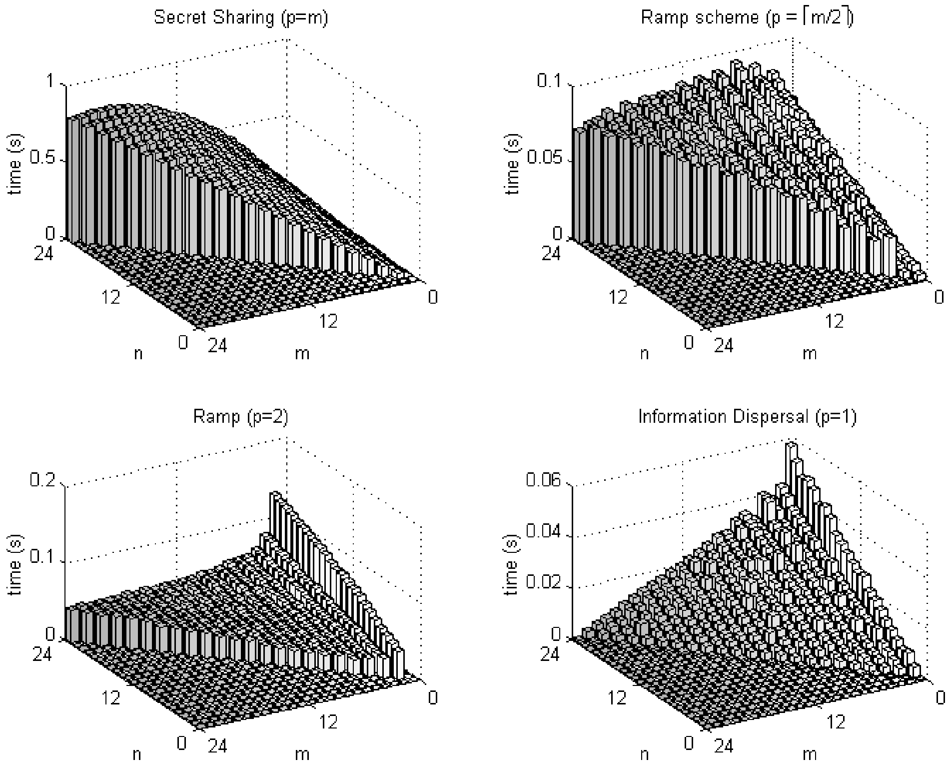

Figure 3: **Measured encode times for 32 KB blocks on a 600 MHz Pentium III.** All m and n combinations up to $n = 25$ are shown for four threshold schemes: secret sharing (p=m), ramp scheme with $p = \lceil m/2 \rceil$, ramp scheme with $p = 2$, and information dispersal ($p = 1$).

on the latter necessary to model storage protocols (e.g., FTP or NT4's CIFS implementation) that open a new TCP/IP connection with slow-start for each file transfered. Competing loads on a storage node would appear in this simple model as increases in the mean and/or the variance.

Overall performance model. To predict the access latency for a given request, we combine these partial models as follows. For a write, ignoring variance and assuming homogeneous servers, we sum the modeled CPU time for encode, the storage node latency for writing one share to one server, and n times the network transmission time required per share. Thus, in sequence, the data is encoded, each share is sent over the network to its appropriate storage node, and all responses are awaited; the n^{th} request is sent after the first $n-1$ and it finishes after the average storage node latency. For a read, the same steps occur in reverse. With variance, the last request sent is not necessarily the last to finish, so a predicted latency for each request must be computed. Our tools use simulation to compute the expected performance for any given scheme. In our experience, the simulation is fast enough for our purposes, with each single-scheme prediction requiring only 10–30ms. It is also accurate enough, yielding predictions that are within 15% of values measured on the prototype described in Section 4 for a variety of validation experiments.

4 Prototype and Validation

We have implemented a prototype survivable storage system, PASIS [40, 48], which we use to validate our performance models of data distribution schemes. Our system fits the architecture illustrated in Figure 2, with the intermediary software executing on the client machines. The encode/decode and multi-read/multi-write aspects of the architecture are implemented in separate libraries. PASIS allows us to select from many data distribution algorithms and specify a wide range of parameters; this enables us to explore many schemes within the context of a single survivable storage system.

4.1 Prototype Components

The encode/decode library supports all of the basic schemes described in Section 2. The library interface exports two main functions: (1) a `decode` call that takes a scheme description and a set of shares as input and returns the original data as output, and (2) an `encode` call that does the reverse. The implementation builds on version 4.1 of the Crypto++ library [11], which provides code for encryption (triple-DES), hashing (MD5), and polynomial operations in a finite field. To enhance its performance for encoding/decoding bulk data (as opposed to key-like secrets), we replaced its stream-based internal functions with block-based versions and reduced the field size. The latter change enables the use of lookup tables to find multiplicative inverses and the product of two elements. Together, these changes yield 3–5× faster encode and decode times for the information dispersal, secret sharing, and ramp algorithms when compared to the original Crypto++ implementations.

The multi-read/multi-write library manages parallel communication with a set of storage nodes. The library interface exports two main functions: (1) a `fetch` call that takes a set of n share descriptions and a number of shares required (m), and (2) a `store` call that does the same for writes. Each share description includes a storage node description and a name for the share within that storage node. For the experiments described in this paper, these libraries were combined in a benchmark application that itself keeps track of per-block metadata.

The library currently supports the use of NFS storage nodes, CIFS storage nodes, and FTP storage nodes. By using these standard protocols, we are able to use independently-implemented, off-the-shelf storage nodes. This is an important practical design consideration that allows us to enhance storage node heterogeneity, which in turn enhances security; for example, the likelihood is low of finding a single attack that can compromise a Sun NFS server, a Network Appliance NFS server, and a SNAP NFS server. Without this feature, $\mathit{Effort}_{BreakIn}$ for the second through n^{th} storage nodes could be zero.

We have also implemented an NFS proxy server that exports an NFSv2 interface to clients and uses the libraries above to encode and distribute the files across a set of storage nodes. To store the necessary metadata, each exported file or directory is represented by two actual sets of storage objects: one set that stores share descriptions for each file block and one set that stores the actual data. The share descriptions for the former are stored in the corresponding directory entries. We generally allow only the local machine to mount the NFS proxy server (over the `loopback` interface), thus allowing us to use survivable storage without changing the kernel. The downside, as expected, is lower performance relative to an in-kernel implementation.

4.2 Model Verification

We have used our prototype, PASIS, to verify that the simple performance model of Section 3 can capture the key performance trends of at least one real survivable storage system. To do so, we measured the necessary model parameters, configured the model accordingly, and then compared the model predictions to measured system performance. The model predictions of read and write throughput were compared to this testbed for all threshold scheme options up to n=6. In almost all cases, they are within 10% and the largest observed error was 15%. Section 5 shows that this level of accuracy is more than sufficient for proper data distribution scheme selection.

The testbed consisted of seven PC systems inter-connected via a dedicated 100 Mbps switch. Each system contains a 600 MHz Intel Pentium III, 256 MB of RAM, a 3Com Fast Etherlink XL network interface card, and a Quantum Atlas 10K disk connected via an Adaptec ULTRA-2 SCSI controller. One system, acting as the client, runs Windows NT 4.0 with Service Pack 5. The other six systems, acting as CIFS storage nodes, run a SAMBA server on Linux RedHat 6.2 with kernel version 2.2.14. Time measurements are based on the Pentium cycle counter, and all values are the mean of 20 measurements.

In addition to the overall validation, we have validated the libraries in isolation. The encode/decode model matches measured library performance to within 10% for all supported data distribution schemes up to $n = 25$. This was also verified on a 300 MHz Pentium II and a 500 MHz Pentium III, and the measured CPU times were observed to scale linearly with CPU performance. Likewise, the model predicts multi-read/multi-write library performance to within 10%.

5 Trade-off Space Exploration

We have built a model with a good balance between accuracy and sensitivity. This allows us to capture the large-scale effects in the trade-off space. Comprehension of such features in the trade-off space is necessary to make system-level design decisions. For large variations in system characteristics (i.e., distinct system configurations), large differences in the selection surface are observed. This supports our claim that systems with different characteristics require different schemes. Also, a specific system should, for performance reasons, use different schemes when operating in very different regions of the space—a region being an area of the availability-security plane. For small variations in the system configuration (i.e., slight inaccuracies in models and/or measurements), only small differences in the selection surface are observed. This means that the scheme selection surface is stable. Although it is inherently difficult to accurately characterize real systems, we are able to determine a close-to-optimal scheme choice for a given system because of the stability of the selection surface.

In the remainder of this section, the default configuration is defined, and our method of measurement is explained. A series of experiments are performed to investigate the sensitivity of the selection surface to small configuration changes, to large configuration changes, and to workload changes. Finally, the availability and security models are considered.

Default configuration. For small systems (e.g., with $n \leq 10$), there are around 1000 schemes to consider. In the remainder of this section, we concentrate on a system with 10 storage nodes. All scheme selection graphs in this section are plotted with performance measured in 32 KB blocks per second, security measured in effort, and availability measured in "number of nines." Performance is calculated based on the CPU, network, and storage nodes as described in Section 4.2. The default value for the network parameter is 100 Mbps. Because no clear metric has emerged from research into estimating effort required to compromise security, we set the values for $E_{BreakIn}$ and $E_{CircumventCryptography}$ equal in the default model. The default value for the system workload is a read/write probability of 0.5 (an equal number of reads and writes). The default value for a single storage node's failure probability is $f_{node} = 0.005$.

The *default configuration* is the base case against which all other configurations are compared. The scheme selection surface for the default configuration is shown in Figure 1.

Measurements. To compare scheme selection surfaces, we use a metric with two components. The first component quantifies what percentage of the scheme choices change. The second component quantifies the magnitude of the performance cost associated with using the other surface's scheme at a given point. For example, we state such results as, "The selection surface differs over X% of the area with an average difference of Y%." When performing sensitivity analysis, we reverse the direction of the comparison (i.e., we examine the impact of inaccuracies in the assumed system configuration). This allows us to determine how accurate the configuration representation must be for the trade-off analysis to be useful. In some regions of the trade-off space, many schemes have similar performance over a range of configurations. In such regions, different schemes may be selected for different configurations, but with minimal performance impact. In other regions, significant performance costs are incurred if the wrong scheme is selected. To capture this interesting effect, we sometimes perform a comparison between two selection surfaces and only list the area and difference for regions that have changed by some minimum amount (i.e., we ignore differences of less than 10% to get a better measure of large scale effects).

Another method of analyzing the trade-off space is used to understand the consumption of resources by the system. Specifically, we examine the ratio of time spent performing CPU operations to the time spent performing network I/O in order to ascertain what regions of the selection surface are bound by each resource. A scheme is considered to be bound by a resource if the resource accounts for greater than 80% of the overall performance time. The resource consumption for the default configuration is presented in Figure 4.

5.1 Sensitivity to the model

If the selection surface is highly sensitive to small changes in the configuration, then it might not be a useful design tool because it would require exact information about how a system will behave. To ensure that the selection surface is relatively stable, we examined its sensitivity to the system characteristics and workload aspects of the default configuration. For the system characteristics, we varied the default network speed by a

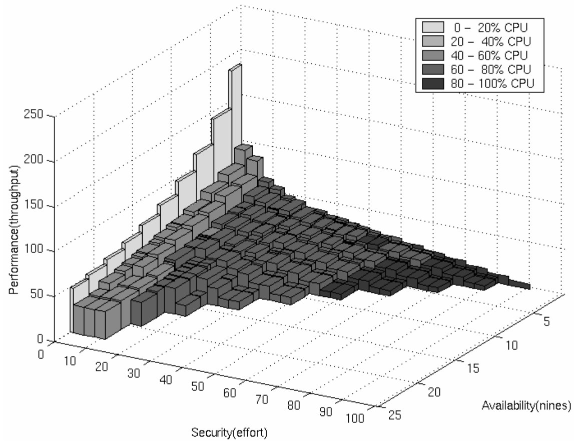

Figure 4: **Balance of resources in default configuration.** Light grey indicates a region of the surface in which the selected scheme is bound by the available network bandwidth. Dark grey indicates a region in which the selected scheme is bound by the CPU. In the default configuration, replication (which appears along the zero security line) is network bound. Farther down the security axis, schemes are CPU bound. There is a large region, comprised of information dispersal, replication with cryptography, ramp schemes, and short secret sharing, with balanced resource consumption.

factor of two in each direction. For the workload, we considered workloads of 40% reads as well as 60% reads. For the security model we varied the value of $E_{BreakIn}$ by 10% in either direction relative to the value of $E_{CircumventCryptography}$. Considering these cases, the selection surface differs by less than 9% of the area with a difference of less than 9%. From this, we conclude that the selection surface is relatively stable. As well, this means that as long as system characteristics are modeled with moderate accuracy and the workload is roughly understood, the scheme selection surface can provide significant insight into the real configuration.

5.2 System characteristics

To investigate the impact that system characteristics have on proper scheme selection, we fix the CPU speed and vary the network speed. This explores system configurations representative of a wide range of distributed systems such as peer-based computing on a LAN or client-server architectures.

We begin by varying the default network speed one order of magnitude (i.e., to 10 Mbps and 1 Gbps). Speeding up the network has little effect on the selection surface. The selection surface differs over less than 10% of the area with an average difference of 3%. Changes resulted from replication with cryptography, a computationally cheap and space-inefficient algorithm, becoming a better selection than information dispersal. Figure 5 shows the selection surface for the fast network. When the network bandwidth is reduced, the selection surface differs over 26% of the area with an average difference of 17%. Clearly, this change is substantial. However, some of this region is comprised of a "checkerboard," where the right choice bounces between two schemes with little performance difference. In this checkerboard region, the selection surface differs over 11% of the area with an average difference of 3%. In one part of the checkerboard region, short secret sharing and ramp schemes have similar performance. In another part, information dispersal schemes with parameters that make them more space-efficient are selected over less space-efficient ones. The remainder of the difference between the surfaces is due to 14% of the area with an average difference of 29%. In this region, which is along the availability axis, information dispersal dominates replication. Figure 6 shows the selection surface for the slow network.

An analysis of the resource consumption over the selection surfaces for the various network speeds considered shows that, for the fast network most of the selection surface is CPU bound, for the default model most of the surface is balanced, and for slow networks most of the surface is network bound. Many conclusions can be drawn from the results of these experiments. First, scheme selection depends on the balance between the speed of the processor and the speed of the network. Second, in system configurations that include a slow network, the space-efficiency of a scheme is a significant indicator of its overall performance. Third, the default configuration can be used to select schemes for faster networks since scheme performance is generally

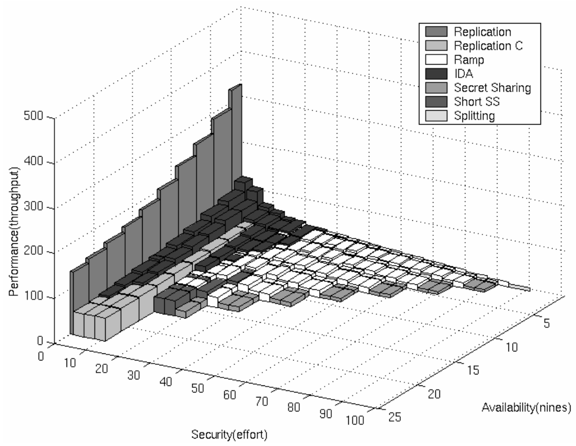

Figure 5: **Fast network (1 Gbps).** Replication with cryptography dominates the low security, high availability region of the graph, because the network consumption of replication has a low performance cost on a fast network. Note: The fast network required doubling the performance axis scale.

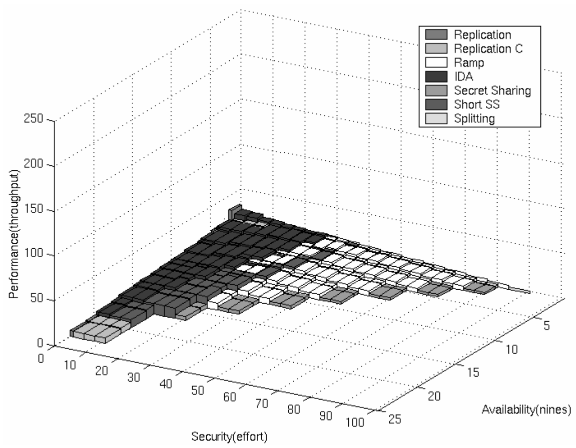

Figure 6: **Slow network (10 Mbps).** Information dispersal dominates along the availability axis. Short secret sharing has similar performance to ramp schemes in the foreground of the graph.

CPU bound in the default configuration.

5.3 Impact of workload

To investigate the impact that workload has on scheme selection, the ratio of reads to writes was varied from a 100% read workload to a 100% write workload. Recall that the default workload is equal parts reads and writes. As shown in Table 2, this experiment found that scheme selection is insensitive to workload over a large portion of this range, 10% reads to 90% reads. However, at the end points of the range, there is substantial change in the selection surface. In particular, changing the workload can change the operating region of an algorithm (i.e., it changes the availability guarantee that can be made)—a clear difference from modifying system characteristics.

The reason the selection surface is insensitive to the workload, except at extremes, is because the workload mainly changes system performance. As the read workload increases/decreases the selection surface expands/contracts along the performance axis. The reason for this is that the read (decode) costs are lower than the write (encode) costs for most algorithms. Only at the ends of the workload range do the different rates at which schemes are able to scale result in new schemes outperforming the scheme selected for a mid-

Read Workload (% reads)	Surface Change	Average Performance Cost
0%	97%	33%
5%	67%	6%
20%	22%	5%
40%	6%	4%
60%	4%	2%
80%	26%	7%
95%	50%	16%
100%	61%	20%

Table 2: **Impact of workload on selection surface.** Scheme selection is insensitive to changes in the mid-range of workloads. Significant changes occur near the endpoints of the possible workloads.

range workload. The conclusion of this experiment is that, for most read-write workloads, scheme selection can be done assuming a workload of 50% reads because the selection surface is stable over a large range. However, for write-once/read-many storage systems, the trade-off space differs radically.

The selection surfaces for the 100% read and 100% write workloads have some specific features that provide insight into the trade-off space. Figure 7 and Figure 8 show the selection surfaces for these workloads. Splitting dominates the low availability-high security region (the back plane of the graph) for the 100% read workload. Whereas, splitting dominates the diagonal that demarcates the edge of the operational region for the 100% write workload. For other workloads, splitting only occurs at the maximum security point. Splitting's encode operation is faster than secret sharing's; however, for any other workload secret sharing dominates splitting because splitting has extremely poor read availability. Splitting's decode operation is extremely fast compared to its encode. This is why it is selected for the read workload—its encode performance does not penalize it.

This insight about splitting is applicable to the rest of the threshold schemes. For the 100% read workload, schemes are not penalized for poor write performance. The result of this is that ramp schemes dominate the region held by information dispersal for write dominated workloads—ramp schemes with large n and low m are competitive with information dispersal schemes with mid-range n and similar m values because their poor write performance is counted. Also, short secret sharing is shown to have little value for a read-only workload. Short secret sharing offers good security for a low encode cost. Again, the encode cost does not matter in a read-only workload—ramp schemes offer better read-only performance than short secret sharing.

Considering the resource consumption of the 100% read and 100% write workload is also instructive. Figure 9 and Figure 10 demonstrate that the best performing schemes have a balanced utilization of resources when performing reads. Whereas, the CPU is the scarcest resource when encoding data for high security. The fact that security is CPU bound matches intuition—security is expensive and system designers loathe paying the price. However, the fact that reading secure data is not as expensive, in terms of CPU cycles, is significant. Indeed, for workloads with $> 60\%$ reads the utilization of resources in the system is balanced deep into the security region.

5.4 Failure model

The failure probability for storage nodes, f_{node}, is an interesting parameter in our model. A total ordering of all schemes based on their availability can be performed without substituting a specific value for this parameter. Thus, the parameter only contracts or expands the surface in the availability dimension. For example, the surface in Figure 11, which is based on a failure probability $10\times$ larger than the default value, has the exact same shape as the surface in Figure 1, except for the scaling along the availability axis. Thus, in some sense the selection surface does not depend on the failure probability—ordering along the availability axis is solely a function of n and m. Clearly though, an accurate understanding of the failure model of storage nodes is necessary to correctly engineer the system to meet its requirements.

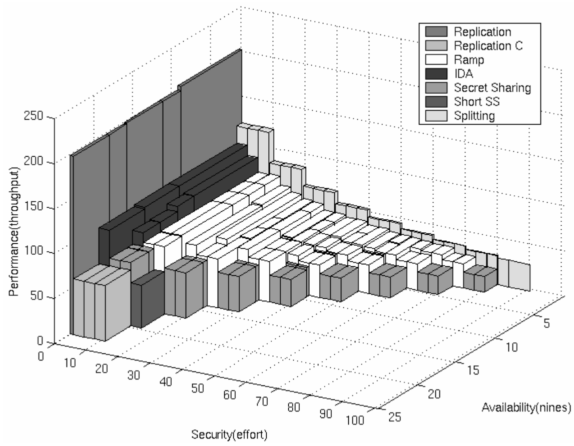

Figure 7: **100% Read workload.**

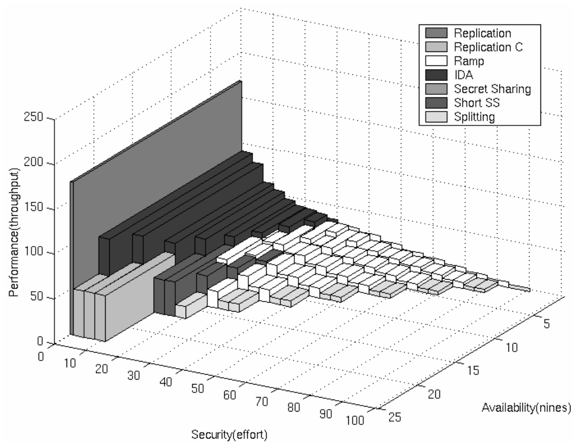

Figure 8: **100% Write workload.**

5.5 Security model

In this section, we investigate what happens if one assumes circumventing cryptographic protection and compromising storage nodes require different amounts of effort. It is important to only consider relative changes in these experiments—the effort metric only provides a relative ordering of schemes.

We performed experiments to see what happens when $E_{CircumventCryptography}$ is varied relative to $E_{BreakIn}$. Reducing $E_{CircumventCryptography}$ by an order of magnitude relative to $E_{BreakIn}$ has little effect on the selection surface. It differs over less than 4% of the area with an average difference of 9%. Thus, the effort valuation used to generate a total order for schemes along the security axis gives less weight to cryptographic confidentiality than to information-theoretic confidentiality. The results are much different when $E_{CircumventCryptography}$ is increased by an order of magnitude relative to $E_{BreakIn}$. Replication with cryptography dominates almost the entirety of the security-availability plane. This is because the maximum possible value of n is 10 in our default model. A threshold scheme can only achieve security of $10\times E_{BreakIn}$—one order of magnitude. To gain more insight into how the selection surface changes, we increased $E_{CircumventCryptography}$ by a factor of 2.5 relative to $E_{BreakIn}$. Figure 12 presents the resulting selection surface. Replication with cryptography dominates a very large region of the scheme selection surface. As well, the region in which short secret sharing dominates ramp schemes grows significantly. Indeed, the selection surface differs by 45% of the area with an average difference of 60%.

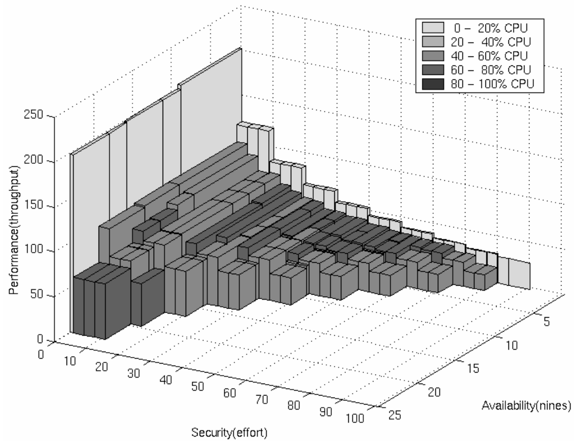

Figure 9: **Balance of resources for a 100% read workload.** The entire availability-security plane has balanced consumption of network and CPU resources when performing reads.

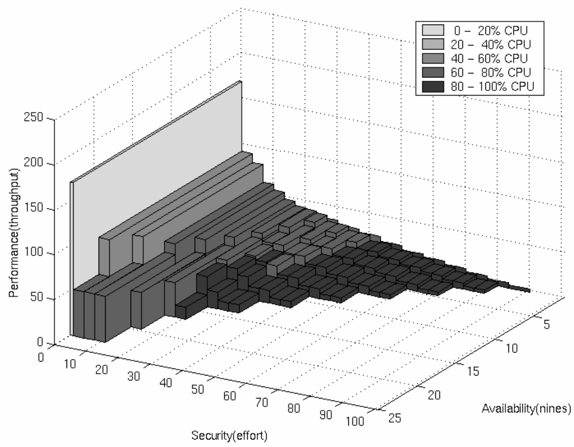

Figure 10: **Balance of resources for a 100% write workload.** The cost of security is entirely in the write (encode) operation. As well, the level of security is bound by the CPU.

Even though the numbers assigned to the security costs have an arbitrary nature, it is interesting to consider how the value placed in cryptographic techniques can change depending on the amount of time confidentiality must be maintained. A goal of security engineering is to expend just enough effort (i.e., minimize performance degradation) to achieve the security goal. For systems with a short confidentiality window, the weighting of $E_{CircumventCryptography}$ relative to $E_{BreakIn}$ should be higher. Effectively, this assumes a bound on the time that an attacker can utilize to crack the cipher, steal keys, or guess keys. On the other hand, information whose confidentiality is a long-term priority must carefully consider the valuation of cryptographic techniques.

5.6 Analysis

Although the trade-off space is complex, we expected specific algorithms to dominate certain regions of the availability-security plane. Within a region we expected the parameters to have a large impact on the performance achieved. Our investigation shows that our original insights were basically correct. However, some transitions between regions are abrupt, indicating that a large performance difference occurs for a small change in security or availability guarantees. For example, the transition from replication to information dispersal and the first transition between schemes within the region where information dispersal dominates

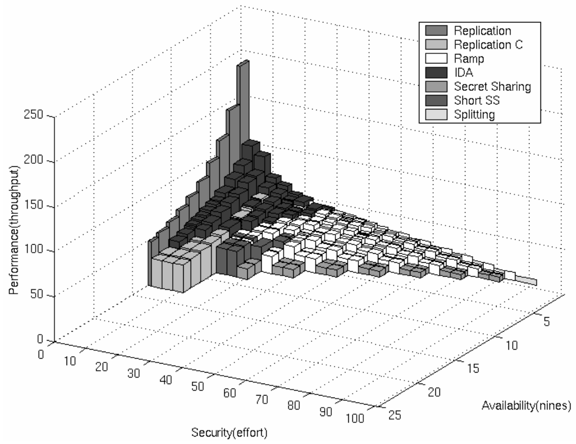

Figure 11: **Order of magnitude increase in node failure probability.** The scheme selection surface contracts by a factor of two along the availability axis. Remember, the scale of the availability axis is the number of nines. An order of magnitude increase in the probability of failure results halves the number of nines of availability.

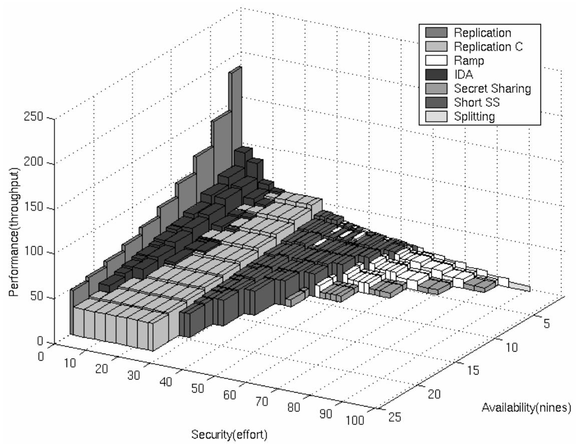

Figure 12: **Placing higher value on cryptography than node security.** If cryptography provides better confidentiality than the security of the storage nodes, replication with cryptography is a very effective scheme. As well, short secret sharing, which combines cryptography with information dispersal, dominates ramp schemes deep into the region of high security.

have this property. This contradicts our original intuition that the large number of schemes would result in a selection surface that is smooth. If a system is operating near a sharp performance transition, the system requirements must be considered and an engineering decision must be made as to whether to err in favor of performance, availability, or security.

Sharp transitions tend to be localized in the back corner of the trade-off space (low availability and low security). The sharpness is due to the data redundancy of threshold schemes. Share size is calculated as $1/(m-(p-1))$, and n shares are generated by a threshold scheme. Consider a 1-1-4 threshold scheme (4-fold replication) and a 1-2-4 threshold scheme (an information dispersal scheme). By increasing m from 1 to 2, the data redundancy is halved (i.e., storage space consumption is the same as 2-fold replication). On a read, both schemes require the same amount of data (one replica versus two shares, each half the size of the replica). The abrupt performance difference on the selection surface is due to write operations, which depend on the amount of data that must be sent across the network.

Another feature of the selection surface occurs when two algorithms perform similarly; the result is a "checkerboard" on the selection surface. In such cases, it is informative to consider the resource consumption of each of the similarly performing schemes. Since the predicted difference is within the error of our model, a system designer can use secondary criteria to select schemes.

Another aspect of the space that interests us is the performance difference between algorithms. Our intuition was that limiting the set of potential algorithms would often result in poor performance. Our intuition was correct: each algorithm tends to be good for only some region of the availability-security plane. If the system being considered does not operate within the algorithm's "good" region, the system has incurred an unnecessary performance penalty. Indeed, limiting the set of algorithms reduces the region of the availability-security plane in which the system can operate. Constraining an algorithm to a specific set of parameters (i.e., building a system around a single scheme) fixes the availability and security that can be offered. Moreover, unless thorough analysis of the trade-off space was done as part of the system design, it is unlikely that the scheme selected is well matched to the system being built. This is only a reasonable design decision if the system is to operate in a single region and the scheme used is on the selection surface at the region.

The highest degree of availability is always achieved by a replication algorithm (because only replication can handle $n-1$ failures) and the highest degree of security is always achieved by splitting (because only splitting can handle $n-1$ compromises).

After some level of security, secret sharing dominates only the edge region that demarcates the reachable portion of the availability-security plane. This is because for a m_1-m_1-n_1 secret sharing scheme, there is usually a p_2-m_2-n_2 ramp scheme with $p_2 = m_1$, $m_2 > m_1$, and $n_2 > n_1$ which provides, by definition, the same security guarantee, provides adequate availability, but has much better performance. The reason for this is that the space savings of ramp schemes with $m > p$ results in fewer computations being required, and thus better performance (i.e., since shares are smaller in size, fewer calculations are required to generate each share). Thus, if secret sharing is the only algorithm that offers the security and availability required, increasing the number of storage nodes, (i.e., the maximum value n may take), will produce a better performing ramp scheme ($> 2\times$ improvement) that meets the availability and security requirements. Clearly, this can only be done if there is not a hard design constraint on the number of storage nodes in the system.

6 Survivable Storage Projects

Survivable storage systems are an active area of current research. This section describes six current and past survivable storage projects, focusing on their system models, target workloads, and data distribution schemes. For each, design insights resulting from our exploration of the trade-off space are noted.

Delta-4. The Delta-4 system [12, 16] is comprised of clients, security servers and data storage servers. The data distribution scheme employed is referred to as fragmentation, redundancy and scattering. Data is decimated into fragments, each fragment is encrypted (with a chained cipher so that fragments must be decoded in a certain order), and the encrypted fragments are replicated. Each data storage server is sent all of the fragments and uses a pseudo-random algorithm to decide which fragments to store. The names of fragments are self-verifying but give no information about the fragment's contents. To access a piece of data, a client must get authorization from a set of storage servers. The storage servers run an agreement protocol to provide integrity. The hash, which enables a client to determine the names of the fragments it wants to read, is stored using secret sharing on the security servers. Both the integrity and confidentiality of the data stored in Delta-4 hinges on the security servers adequately protecting fragments' meta-data. Since CPU cycles are not as scarce as they were fifteen years ago, cryptography coupled with information dispersal should be used in a Delta-4 type system rather than replication. Indeed, since all shares are sent to all storage nodes, space-efficiency of encoding is paramount to reducing bandwidth consumption.

Publius. Publius [47] is strongly influenced by Anderson's Eternity Service proposal [2], which argues for survivable storage plus anonymity. Publius uses encrypted replicas for data distribution. The key used for encryption is encoded with secret sharing, and a single share of the key is stored with each replica. This algorithm is very similar to short secret sharing, except that replication rather than information dispersal is used to encode the cipher text. No indication is given as to the specific parameters selected for the data distribution scheme. However, the authors do identify parameter selection as a difficult task, citing the desire to balance resistance to censorship and performance. The availability of data in a Publius system is limited by the secret sharing of the encryption key. Thus, the extra space consumed by replication, over information dispersal is an inefficiency. Because decode times for information dispersal schemes is so low and Publius is so heavily weighted towards reads, 2–4× space reduction could be realized at very little incremental cost by utilizing a better scheme.

Intermemory. The Intermemory project [10, 22] focuses on archival storage of public data (i.e., write-once, read-many data with high availability and integrity requirements over time). Intermemory's goal is to provide extraordinarily high availability while minimizing storage consumption and the expected cost of reading data. Deep encoding is the data distribution scheme used in Intermemory. Deep encoding involves storing a single copy of a piece of data, then applying a threshold scheme (Tornado schemes in this case [35]) to the piece of data, distributing the shares, and then having the storage nodes recursively apply the same threshold scheme to each share they receive. Deep encoding is performed to some depth level (e.g., deep encoding of depth three stores a replica, n shares, and n^2 subsequent shares of shares). Intermemory uses a 1-16-32 threshold scheme applied to depth level three. Public-key cryptography is discussed as a means to provide long-term integrity of stored data in the Intermemory system. The Intermemory project has selected an excellent scheme that provides high availability with reasonable expected read performance and very low storage space overhead. However, integrity is explicitly stated as a goal of the system. An attacker need only compromise a single replica to defeat the integrity of the stored data. Adding a hash to the encoding algorithm and removing the first depth level of deep encoding would greatly increase the integrity of the Intermemory system at the expense of increasing the expected cost of performing a read.

Oceanstore. Oceanstore [31] is envisioned as a wide-area data utility. It intends to leverage excess storage capacity over a wide area to provide survivable storage of data accessible from anywhere. Oceanstore stores two types of data: active data (which can be read and written) and archival data (which is write-once, read-many). Active data is stored using encrypted replicas. Replicas have self-verifying names to provide integrity. Agreement algorithms are used to manage authorization to data. Archival data is stored using deep encoding as in Intermemory. Oceanstore is addressing many other issues pertaining to building a data storage utility of global scale, including client mobility, robust naming, data migration, and replica discovery. Analyzing the trade-offs amongst data distribution schemes would enable Oceanstore to clearly understand the different characteristics of the two distinct data distribution schemes it employs. Indeed, such an analysis is necessary for them to determine at which workload data should switch from being active to archival.

Farsite. The Farsite project [7] is exploring peer-based storage in a local-area setting. Farsite is attempting to leverage the idle cycles and unused storage capacity of desktop workstations, which inherently have poor availability characteristics, to provide a survivable storage system. Farsite uses encrypted replication with self-verifying names. A Byzantine agreement protocol amongst client machines storing meta-data is used to guarantee the integrity of the directory service (and subsequently provide the self-verifying nature of the replica names). To limit space consumption, single instance storage [6], in conjunction with convergent encryption, is used to ensure that only n replicas of any similar piece of data is stored system wide. Since Farsite explicitly states concerns about storage space consumption, information dispersal rather then replication should probably be used. With relatively small CPU costs, 2–4× reductions in space and bandwidth utilization are available. Further, both single instance storage and convergent encryption can still be used with the shares generated by information dispersal.

e-Vault. The e-Vault system [18, 25] is a survivable storage system that uses either information dispersal or short secret sharing, depending on whether or not strong confidentiality is desired. Distributed fingerprints [29] are used for integrity. The system is designed for a local-area setting in which clients interact with storage servers via a "gateway" (a designated storage server). Performance numbers are given for the 1-2-3 information dispersal scheme [25], since the test platform is limited to three storage servers. This is a very limited system in which to consider information dispersal schemes. A system with just a few more storage nodes has significantly better performance and availability characteristics.

PASIS. Our prototype system, PASIS [40, 48], is described in Section 4. PASIS provides an implementation of a survivable storage system against which we validate our performance models. The main design goal of PASIS is to be flexible enough for us to investigate many different approaches to building survivable storage systems. Our implementation of encode/decode functionality is separate from our multi-read/multi-write functionality. The encode/decode library provides threshold algorithms, cryptographic algorithms, and hybrid algorithms, all of which can be instantiated over a wide range of parameters. The library is structured so as to facilitate the addition of data distribution algorithms and the composition of hybrid algorithms. The multi-read/multi-write library supports NFS, CIFS, and FTP storage nodes and can be extended to support additional types of storage nodes. The flexibility of the PASIS design allows us to explore the complex engineering trade-offs of survivable storage system design within the context of a real system.

7 Summary

This paper illustrates the complex trade-off space associated with selecting the right data distribution scheme for a survivable storage system. A reasoned approach to exploring this trade-off space is developed and used to explore the space's sensitivity to various system and workload characteristics. Although further work is needed to refine the models, we believe that this paper takes a big step in the right direction—away from *ad hoc* selections and towards informed engineering decisions.

This work was done in the context of the PASIS survivable storage project at Carnegie Mellon University. Additional information on our work and results can be found on the project web site [40].

8 Acknowledgements

The authors would like to thank John D. Strunk, Garth R. Goodson, and Theodore M. Wong for extensive discussion and feedback on the directions and goals of this research. We would also like to thank the many other graduate students in the Parallel Data Lab that provided feedback on rough drafts of this paper. Finally, we would like to thank Matt Vestal for administering the cluster of machines on which we ran experiments.

References

[1] Darrell C. Anderson, Jeffrey S. Chase, and Amin M. Vahdat. Interposed request routing for scalable network storage. *Symposium on Operating Systems Design and Implementation* (San Diego, CA, 22–25 October 2000), 2000.

[2] Ross J. Anderson. The Eternity Service. *PRAGOCRYPT*, pages 242–253. CTU Publishing, 1996.

[3] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. Serverless network file systems. *ACM Transactions on Computer Systems*, **14**(1):41–79, February 1996.

[4] G. R. Blakley. Safeguarding cryptographic keys. *AFIPS National Computer Conference* (New York, NY, 4–7 June 1979), pages 313–317. AFIPS, 1979.

[5] G. R. Blakley and Catherine Meadows. Security of ramp schemes. *Advances in Cryptology - CRYPTO*, pages 242–268. Springer-Verlag, 1985.

[6] William J. Bolosky, Scott Corbin, David Goebel, and John R. Douceur. Single Instance Storage in Windows 2000. *USENIX Windows Systems Symposium* (Seattle, WA, 3–4 August 2000), pages 13–24. USENIX Association, 2000.

[7] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasability of a serverless distributed file system deployed on an existing set of desktop PCs. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Santa Clara, CA, 17–21 June 2000). Published as *Performance Evaluation Review*, **28**(1):34–43. ACM, 2000.

[8] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. *Symposium on Operating Systems Design and Implementation* (New Orleans, LA, 22–25 February 1999), pages 173–186. ACM, 1998.

[9] Miguel Castro and Barbara Liskov. Proactive recovery in a Byzantine-fault-tolerant system. *Symposium on Operating Systems Design and Implementation* (San Diego, CA, 23–25 October 2000), pages 273–287. USENIX Association, 2000.

[10] Yuan Chen, Jan Edler, Andrew Goldberg, Allan Gottlieb, Sumeet Sobti, and Peter Yianilos. A prototype implementation of archival Intermemory. *ACM Conference on Digital Libraries* (Berkeley, CA, 11–14 August 1999), pages 28–37. ACM, 1999.

[11] Wei Dai. *Crypto++ reference manual*. http://cryptopp.sourceforge.net/docs/ref/.

[12] Yves Deswarte, L. Blain, and Jean-Charles Fabre. Intrusion tolerance in distributed computing systems. *IEEE Symposium on Security and Privacy* (Oakland, CA, 20–22 May 1991), pages 110–121, 1991.

[13] eVault. http://www.bell-labs.com/user/garay/#projects.

[14] Farsite. http://www.research.microsoft.com/sn/Farsite/.

[15] J. Fraga and D. Powell. A fault and intrusion-tolerant file system. *IFIP International Conference on Computer Security*, pages 203–218. Elsevier, 1985.

[16] Jean-Michel Fray, Yves Deswarte, and David Powell. Intrusion-tolerance using fine-grain fragmentation-scattering. *IEEE Symposium on Security and Privacy* (Oakland, CA, 7–9 April 1986), pages 194–201. IEEE, 1986.

[17] FreeHaven. http://www.freehaven.net/.

[18] Juan A. Garay, Rosario Gennaro, Charanjit Jutla, and Tal Rabin. Secure distributed storage and retrieval. *Theoretical Computer Science*, **243**(1–2):363–389. Elsevier, September 2000.

[19] Hector Garcia-Molina and Daniel Barbara. How to assign votes in a distributed system. *Journal of the ACM*, **32**(4):841–855, October 1985.

[20] Garth A. Gibson, David F. Nagle, Khalil Amiri, Jeff Butler, Fay W. Chang, Howard Gobioff, Charles Hardin, Erik Riedel, David Rochberg, and Jim Zelenka. A cost-effective, high-bandwidth storage architecture. *Architectural Support for Programming Languages and Operating Systems* (San Jose, CA, 3–7 October 1998). Published as *SIGPLAN Notices*, **33**(11):92–103, November 1998.

[21] D. K. Gifford. *Violet: an experimental decentralized system.* Technical report CSL–79–12. Xerox Palo Alto Research Center, CA, November 1979.

[22] Andrew V. Goldberg and Peter N. Yianilos. Towards an archival intermemory. *IEEE International Forum on Research and Technology Advances in Digital Libraries* (Santa Barbara, CA, 22–24 April 1998), pages 147–156. IEEE, 1998.

[23] Jim Gray and Daniel P. Siewiorek. High-availability computer systems. *IEEE Computer*, **24**(9):39–48, September 1991.

[24] Intermemory. http://www.intermemory.org/.

[25] A. Iyengar, R. Cahn, C. Jutla, and J. A. Garay. Design and implementation of a secure distributed data repository. *IFIP International Information Security Conference* (Vienna, Austria, and Budapest, Hungary, 31 August–4 September 1998), pages 123–135. ACM, 1998.

[26] Raj Jain. *The art of computer systems performance analysis.* John Wiley & Sons, 1991.

[27] Ehud D. Karnin, Jonathan W. Greene, and Martin E. Hellman. On secret sharing systems. *IEEE Transactions on Information Theory*, **IT-29**(1):35–41. IEEE, January 1983.

[28] Leonard Kleinrock. *Queueing systems.* John Wiley and Sons, 1973.

[29] Hugo Krawczyk. Distributed fingerprints and secure information dispersal. *ACM Symposium on Principles of Distributed Computing* (Ithaca, NY, 15–18 August 1993), pages 207–218, 1993.

[30] Hugo Krawczyk. Secret sharing made short. *Advances in Cryptology - CRYPTO* (Santa Barbara, CA, 22–26 August 1993), pages 136–146. Springer-Verlag, 1994.

[31] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaten, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: an architecture for global-scale persistent storage. *Architectural Support for Programming Languages and Operating Systems* (Cambridge, MA, 12–15 November 2000). Published as *Operating Systems Review*, **34**(5):190–201, 2000.

[32] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, **4**(3):382–401. ACM, July 1982.

[33] Edward K. Lee and Chandramohan A. Thekkath. Petal: distributed virtual disks. *Architectural Support for Programming Languages and Operating Systems* (Cambridge, MA, 1–5 October 1996). Published as *SIGPLAN Notices*, **31**(9):84–92, 1996.

[34] Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, Liuba Shrira, and Michael Williams. Replication in the Harp file system. *ACM Symposium on Operating System Principles* (Pacific Grove, CA, 13–16 October 1991). Published as *Operating Systems Review*, **25**(5):226–238, 1991.

[35] Michael Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A. Spielman. Analysis of low density codes and improved designs using irregular graphs. *ACM Symposium on Theory of Computing* (Dallas, TX, 23–26 May 1998), pages 249–258. ACM, 1998.

[36] Dahlia Malkhi and Michael K. Reiter. Secure and scalable replication in Phalanx. *IEEE Symposium on Reliable Distributed Networks* (West Lafayette, IN, 20–23 October 1998), 1998.

[37] MojoNation. http://www.mojonation.net/.

[38] Oceanstore. http://oceanstore.cs.berkeley.edu/.

[39] Rodolphe Ortalo, Yves Deswarte, and Mohamed Kaaniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Transactions on Software Engineering*, **25**(5):633–650. IEEE, September 1999.

[40] PASIS. http://www.ices.cmu.edu/pasis/.

[41] Publius. http://cs1.cs.nyu.edu/waldman/publius/.

[42] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, **36**(2):335–348. ACM, April 1989.

[43] Adi Shamir. How to share a secret. *Communications of the ACM*, **22**(11):612–613. ACM, November 1979.

[44] Daniel P. Siewiorek and Robert S. Swarz. *Reliable computer systems: design and evaluation.* Digital Press, Second edition, 1992.

[45] R. H. Thomas. A majority consensus approach to concurrency control. *ACM Transactions on Database Systems*, **4**:180–209, 1979.

[46] Martin Tompa and Heather Woll. How to share a secret with cheaters. *Journal of Cryptography*, **1**(2):133–138, 1988.

[47] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: a robust, tamper-evident, censorship-resistant, web publishing system. *USENIX Security Symposium* (Denver, CO, 14–17 August 2000), pages 59–72. USENIX Association, 2000.

[48] Jay J. Wylie, Michael W. Bigrigg, John D. Strunk, Gregory R. Ganger, Han Kiliccote, and Pradeep K. Khosla. Survivable information storage systems. *IEEE Computer*, **33**(8):61–68. IEEE, August 2000.