Privacy, Preservation and Performance:

The 3 P's of Distributed Data Management

Bobji Mungamuru
Stanford University
bobji@i.stanford.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Abstract

Privacy, preservation and performance ("3 P's") are central design objectives for secure distributed data management systems. However, these objectives tend to compete with one another. This paper introduces a model for describing distributed data management systems, along with a framework for measuring privacy, preservation and performance. The framework enables a system designer to quantitatively explore the tradeoffs between the 3 P's.

1 Introduction

There are many conflicting goals when designing a distributed data management system. For example, one would like a system that is enforces "privacy" by not divulging data to unauthorized entities. At the same time, one would prefer a system that "preserves" the data, i.e., protects the data from hardware failures, natural disasters, and so on. And of course one does not want to sacrifice "performance" – a system that takes a year to give us data would not be very useful. There are many such conflicting desired attributes, often closely related to the ones we illustrated: confidentiality, security, integrity, reliability, throughput, and so on.

While all of these attributes have been studied extensively, there is not much work that considers the tradeoffs among attributes. We believe that today the real challenge in designing a system is in achieving a balance between such conflicting attributes. For example, one can build a highly secure system that simply deletes all received data. Such a system excels on the security and privacy dimension (no data will ever leak out!), but is not very attractive in other dimensions. Similarly, one can build a highly reliable system that makes many copies of its data. This system would excel along the preservation dimension, but each extra copy may increase the chances of unauthorized break-ins. If we encrypt a large collection of records as a single unit, performance for reading individual records

suffers. If we encrypt each record individually, reads will be faster, but overall security may not be as strong.

In this paper we study, *in a unified way*, three conflicting attributes of a distributed system, and show how one can design a system that strikes a balance among these attributes. We call these attributes the "3 P's" of distributed data management – *privacy*, *preservation* and *performance*. Precise definitions of what we mean by these terms will be given later when we propose metrics for privacy, preservation and performance. Informally,

- *Privacy* refers to protecting data from unauthorized access (related to security and confidentiality).
- *Preservation* ensures that data is still available and uncorrupted far into the future (related to integrity, availability, and reliability).
- *Performance* refers to the quick, timely access to our distributed data (related to response time and throughput).

We probably do not have to argue very hard that the 3 P's are critical in many of today's large-scale applications. Search engines, medical systems, social networking sites and banking systems are just a few systems that require a combination of privacy, preservation and performance. Such systems are often configured to extract maximal performance out of a limited set of physical resources. However, extracting maximal performance may come at the cost of privacy and preservation. We may wonder what is the highest level of privacy, preservation or performance achievable using a given set of resources? What benefits could be derived from investing in additional resources? Thus, being able to model and reason about the 3 P's in an integrated way is of both theoretical and practical interest.

1.1 Overview

We begin by defining a declarative and unified model that encodes the choices we can make regarding the 3 P's. The model is based upon four classes of operators – Split, Copy, Threshold and Partition. The first three use replication and data decomposition techniques to improve privacy and preservation. Partition operators break up collections of data objects into smaller sets for performance and risk reasons. Each class of operators has several concrete implementations, and each implementation has different performance characteristics.

Distributed data management systems are then built up as compositions of these operators. We define metrics for the privacy, preservation and performance offered by a given system, and formulate constrained optimization problems over these metrics. Roughly speaking, choosing a particular composition of operators will impact privacy and preservation, whereas choosing concrete implementations for each operator will impact performance. Unfortunately, these optimization problems are intractable for even moderately-sized problem instances. Thus, we will also

propose a tractable technique for finding approximate solutions. We conclude with a detailed case study illustrating an application of our model and techniques.

The end result will be a solution strategy for the following problem: *Given a set of physical resources (e.g., data centers, servers, storage devices), and a collection of data objects (e.g., documents, photos, music files, e-mail) with known usage characteristics, sensitivities and sizes, how should we distribute them across our resources to achieve a desired balance between the 3 P's – privacy, preservation and performance?*

1.2 Related Work

Most existing research in distributed data management shows how to design systems with either good privacy, good preservation or good performance properties (e.g., [3, 5, 9, 13, 14]). However, there is relatively little literature discussing the tradeoffs between these three objectives. There is some work on how to *safeguard* data – that is, managing the privacy and preservation aspects together. For example, threshold schemes are used in [15] as a primitive to provide fault tolerance guarantees in a distributed system. In [11] and [12], the properties of threshold schemes are achieved by alternative, more efficient means. In [6] and [7], it is shown how to quantify and optimally manage the tradeoff between privacy and preservation. However, in these works performance is not considered, and they discuss only how to distribute a single (monolithic) data object.

In this paper, we extend the techniques in [7] in two important ways. First, we include performance as a dimension in our optimization. We will answer such questions as: What is the best performing system we can build given some minimum requirements on privacy and preservation? Second, we show how to optimally manage a collection of heterogeneous data objects. For example, consider a collection comprised of some documents, photos, music files and some e-mail. The value, sensitivity, size and access patterns of each of these data objects varies widely, so it seems prudent to use different strategies to distribute each of them. Our work prescribes good strategies for distributing such heterogeneous collections across a set of physical storage resources.

2 Model

Our first task is to develop a framework to describe the choices that affect privacy, preservation and performance in a distributed data management system. In Sections 3 and 4, we will discuss how to measure the 3 P's, and search for "good" systems within this framework.

2.1 Split, Copy and Threshold Operators

We begin by defining a pair of *operators*, which take a collection of data objects as input and produce one or more collections of data objects as output. A *k-way Copy operator*, denoted by **C**, produces as its output k identical

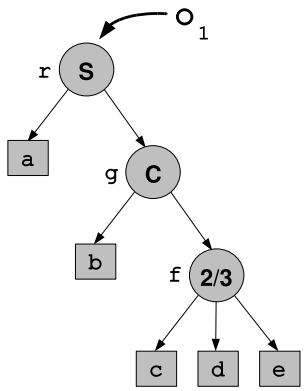

Figure 1: The configuration $F_\Theta = \mathbf{a}(\mathbf{b} + f_{2/3}(\mathbf{c}, \mathbf{d}, \mathbf{e}))$.

copies of its input data. A *k-way Split operator*, denoted by **S**, applies an *encoding* to its input and produces k outputs, or *shares*, such that all k shares are required to fully reconstruct the input.

Copy and Split are special cases of a more general class of operators, known as Threshold operators. A *k-of-n Threshold operator*, denoted $\mathbf{k/n}$, applies a transformation to its input and produces n shares such that any k are sufficient to reconstruct the input. A Copy operator corresponds to $k = 1$ and a Split corresponds to $k = n$.

There can be several possible implementations for a given operator. For example, as we discuss in Section 2.5, there are several ways to implement a k-way Split operator, such as XOR'ing the input data with $k - 1$ randomly generated bit sequences or using repeated AES encryptions with $k - 1$ different encryption keys (giving k outputs total in each case).

2.2 Configurations

Operators can be recursively composed in interesting ways in order to *safeguard* data objects i.e., provide privacy and preservation. We refer to such compositions of operators as *configurations*. For example, consider the configuration shown in Figure 1, which we use to safeguard a single sensitive data object, $\mathbf{o}_1$ (say, a document). The data object $\mathbf{o}_1$ is input to the Split operator at the root, labelled **r**. The input[1] to **r** is then split into two shares, **a** and **g**, using a 2-way Split operator so that both **a** and **g** are needed to reconstruct **r**. Two identical copies of **g** are then made, labelled **b** and **f**, using the 2-way Copy operator at **g**. Finally, **f** is divided once again into three shares **c**, **d** and **e** such that any two are sufficient to reconstruct **f**.

The configuration in Figure 1 tells us how to decompose the data at **r** and distribute it across the *servers* (i.e., physical storage locations) $\mathbf{a}, \mathbf{b}, \mathbf{c}$, **d** and **e**. The data objects $\mathbf{r}, \mathbf{g}$ and **f** are *transient*, in the sense that they are not

[1]When we refer to a vertex **v**, we are sometimes referring to the data that is input to the operator or storage server at vertex **v** (e.g., "reconstructing **v**"), while at other times we are referring to the operator or server itself (e.g., "choosing an implementation for **v**"). The meaning will always be clear from the context.

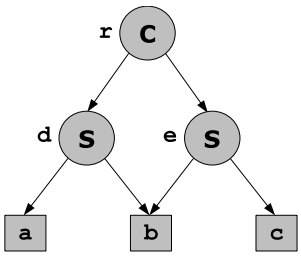

Figure 2: The configuration $F_\Theta = \mathbf{ab} + \mathbf{bc}$.

materialized. To reconstruct $\mathbf{r}$, we need either objects $\{\mathbf{a}, \mathbf{b}\}$ or $\{\mathbf{a}, \mathbf{c}, \mathbf{d}\}$ or $\{\mathbf{a}, \mathbf{c}, \mathbf{e}\}$ or $\{\mathbf{a}, \mathbf{d}, \mathbf{e}\}$. Thus, we can represent this configuration by the *access formula* $\mathbf{a}(\mathbf{b} + f_{2/3}(\mathbf{c}, \mathbf{d}, \mathbf{e}))$, where $f_{2/3}(\mathbf{c}, \mathbf{d}, \mathbf{e})$ is true if two or more of $\mathbf{c}$, $\mathbf{d}$ and $\mathbf{e}$ are true, and false otherwise.

Using the configuration in Figure 1, we have managed to safeguard the data at $\mathbf{r}$ in the following sense. Suppose the data object at $\mathbf{a}$ was leaked, say, to an attacker. Without also obtaining $\mathbf{g}$, the attacker is unable to reconstruct $\mathbf{o}_1$. An attacker will always need at least two of $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ and $\mathbf{e}$ in order to reconstruct $\mathbf{o}_1$. Alternatively, suppose the data object at $\mathbf{b}$ was lost, say, due to a disk failure. Using $\mathbf{c}, \mathbf{d}$ and $\mathbf{e}$, we can still reconstruct $\mathbf{g}$, $\mathbf{r}$, and therefore $\mathbf{o}_1$. Thus, we can lose one of $\mathbf{b}, \mathbf{c}, \mathbf{d}$ or $\mathbf{e}$ and still recover $\mathbf{o}_1$.

A configuration is, in general, a directed acyclic graph (DAG) where terminal vertices are servers[2] and non-terminal vertices are operators. Shared vertices are permitted, such as vertex $\mathbf{b}$ in Figure 2. A shared vertex represents a *union* of input data[3]. In our example, if an attacker breaks into $\mathbf{b}$, he obtains both $\mathbf{d}$ and $\mathbf{e}$. If server $\mathbf{b}$ fails, then both $\mathbf{d}$ and $\mathbf{e}$ are lost.

2.3 Data Objects

A *data object* can be thought of as a "blob of bits" that exists as a logical unit. Documents, music files, email, source code and images are all examples. Section 2.2 gave an example involving a single object, $\mathbf{o}_1$. In general, we are interested in distributing *collections* of one or more (possibly heterogeneous) data objects.

When a collection of data objects is input to an operator, each object in the collection is processed *individually*. For example, suppose a collection $\{\mathbf{o}_1, \mathbf{o}_2\}$ is input to vertex $\mathbf{r}$ in Figure 1. If $\mathbf{r}$ is implemented using encryption, $\mathbf{a}$ would be the collection $\{\mathbf{o}_1^*, \mathbf{o}_2^*\}$ of (individually) encrypted objects and $\mathbf{g}$ would be a single key (alternatively, we can use a collection of keys, one for each object in the input). To reconstruct $\mathbf{o}_1$, we would only need to download $\mathbf{o}_1^*$ from $\mathbf{a}$, rather than the entire encrypted collection.

A key point is that Split, Copy and Threshold do not "break up" a collection of data objects – if a given output

[2]We use the term "server" generically to include any type of storage location, including (say) a USB disk or even an entire data center.

[3]An alternate set of semantics is discussed in [8].

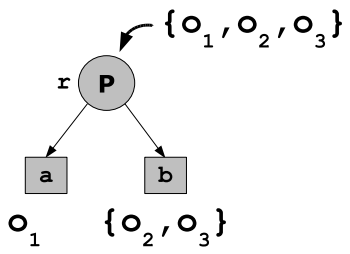

Figure 3: A Partition operator.

is a collection, then there is a one-to-one correspondence between objects in that output and objects in the input.

If a server is broken-into by an attacker, all of the data objects at that server are obtained by the attacker. Similarly, if a server is lost or destroyed, all of the data objects on that server are lost. When data accesses (e.g., reads and writes) occur, however, individual data objects are accessed. Thus, the collections of data objects stored on servers are the *units of failure* in our model, whereas individual data objects are the *units of access*.

2.4 Partition Operator

A *k-way Partition operator*, denoted **P**, simply breaks its input into k disjoint *subcollections*, and outputs these k subcollections without any further encoding. Figure 3 illustrates a 2-way Partition of a collection of three data objects, $\{\mathbf{o}_1, \mathbf{o}_2, \mathbf{o}_3\}$.

The key difference between a Partition and a Split is that individual outputs of a Partition can be valuable on their own. In Figure 3, for example, vertex **b** is valuable even without **a**, since the subcollection $\{\mathbf{o}_2, \mathbf{o}_3\}$ is stored there. If **r** were a Split, this would not be the case. In this sense, we can think of a Partition as an "insecure Split".

2.5 Operator Implementations

A configuration does not specify how to actually implement each operator. To fully specify a distribution strategy, we must select an implementation for each operator, and decide where to send each output of the operator.

There are several ways to Split data into k shares such that all k are required to reconstruct the input. Earlier, we gave the example of AES and XOR'ing with random bit sequences. Encryption algorithms offer a whole family of widely-used implementation options. Another possible implementation would be vertically partitioning (or normalizing) the columns of a database relation, such that no subset of columns in sensitive [2]. The objects in the collection, in this case, would be individual records.

Similarly, there are several implementations of Threshold operators (e.g., [4, 10]). Threshold operators in practice tend to suffer from poor performance, so [12] proposes a faster alternative that uses a composition of replication and XOR operations.

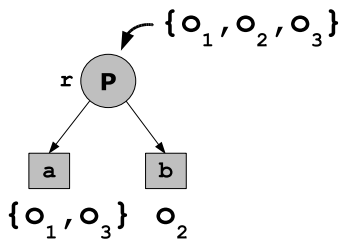

Figure 4: Alternate implementation of **P**.

For a Partition operator, choosing an implementation means deciding to which output to send each object in the input collection. Figure 3 depicts one example, whereas Figure 4 shows an alternate implementation.

Copy operators, unlike Split, Threshold and Partition, do not have a variety of different implementation options. Most methods of replication perform and behave virtually the same for our purposes, so we will assume a unique implementation option for Copy operators.

The *processing time* per MB of input data, and the *sizes* of the outputs per MB of input are the most important characteristics of an operator implementation. For example, in Figure 1, perhaps AES with 128-bit keys is used at **r**, with the ciphertext at **a** and encryption key at **g**. The **k**/**n** at **f** could use Shamir's scheme. Thus, if $\mathbf{o}_1$ was of size s, the data at **a** would also be size s, whereas **b**, **c**, **d** and **e** would be 128 bits each. Instead, if XOR were used at **r**, the outcome would be quite different – **a**, **b**, **c**, **d**, **e** and the input to **f** would all be size s.

2.6 Summary

We now give a brief summary of Section 2, and define some notation for use throughout the remainder of this paper. We are given a collection of data objects, $\mathcal{O} \equiv \{\mathbf{o}_1, \mathbf{o}_2, \dots\}$, and a set of *servers*, $\mathcal{X} \equiv \{\mathbf{x}_1, \mathbf{x}_2, \dots\}$. We wish to *distribute* $\mathcal{O}$ across $\mathcal{X}$ in such a way that we can read and write quickly (i.e., we want good performance), and the objects are safeguarded from unauthorized access (privacy) and loss (preservation). We are also given a set $\mathcal{I}$ of *implementation options* from which we can select an implementation for each operator.

A *data distribution strategy* (or *strategy* for short), $\mathcal{S}$, is specified by the triple $\mathcal{S} \equiv (\mathcal{O}, \Theta, I)$, where Θ is the configuration we choose and I gives the implementation for each operator. $\Theta \equiv \Theta(\mathcal{X}, \mathcal{Y}, \mathcal{E})$, where $\mathcal{X}$ is the set of servers defined above, $\mathcal{Y} \equiv \{\mathbf{y}_1, \mathbf{y}_2, \dots\}$ is the set of non-terminals vertices (i.e., operators), and $\mathcal{E}$ is the set of directed edges in Θ. The function $I : \mathcal{Y} \to \mathcal{I}$ specifies which operator implementation in $\mathcal{I}$ we are using at each non-terminal vertex $\mathbf{y} \in \mathcal{Y}$. We often assume that vertices are labelled $\mathbf{a}, \mathbf{b}, \mathbf{c}, \dots$ and that the root is labelled **r**.

We will often represent a configuration Θ by its *access formula*, $F_\Theta : \{0,1\}^{|\mathcal{X}|} \to \{0,1\}$, which is a factored monotone boolean expression over the elements of $\mathcal{X}$. F_Θ captures the *reconstruction semantics* of Θ. Copy operators are represented by boolean addition, whereas Split and Partition operators are represented by boolean multiplication

(since **P** behaves like **S** with respect to reconstruction semantics). A k-of-n Threshold operator is represented by the function $f_{k/n} : \{0,1\}^n \rightarrow \{0,1\}$, which returns true if and only if at least k inputs are true. *Satisfying assignments* of F_Θ indicate which terminals are sufficient for an attacker to reconstruct the data at the root, whereas *falsifying assignments* indicate which must be lost for reconstruction of the root to be rendered impossible. For example, the configuration given in Figure 1 is $F_\Theta = \mathbf{a}(\mathbf{b} + f_{2/3}(\mathbf{c}, \mathbf{d}, \mathbf{e}))$. The particular factorization of F_Θ is important, for our purposes. For example, $F_\Theta = \mathbf{a}(\mathbf{b} + f_{2/3}(\mathbf{c}, \mathbf{d}, \mathbf{e}))$ is not the same as $F_\Theta = \mathbf{ab} + \mathbf{acd} + \mathbf{ade} + \mathbf{ace}$, although they are *logically equivalent*[4].

Our goal in this paper is to output the "best" data distribution strategy $\mathcal{S}$, given data objects $\mathcal{O}$, servers $\mathcal{X}$, and implementations $\mathcal{I}$.

3 Evaluating Strategies

In this Section, we discuss how to quantify privacy, preservation and performance for a given strategy, $\mathcal{S}$. In Section 4, we will develop a procedure for finding good data distribution strategies under these measures.

3.1 Privacy and Preservation

Privacy and preservation are closely related, in that they can both be measured using the notion of "harm". Privacy is measured using the expected damage – the harm we expect suffer due to leaks of sensitive data – which we try to minimize. Preservation is measured by expected loss – the harm due to loss of valuable data – which we will also try to minimize. The expectations are taken with respect to the probabilities of failure, which we define next.

3.1.1 Probabilities of Failure

The *probability of break-in*, $p_{\mathbf{v}}$, for a vertex $\mathbf{v}$ is the probability that an attacker breaks into enough servers to reconstruct all of the data at vertex $\mathbf{v}$. We use the term "break-in" generically to refer to an unauthorized party gaining access to our data. For example, if server **b** in Figure 1 was a DVD kept in somebody's desk, a "break-in" might mean that the DVD is physically stolen. Alternatively, if **b** is a password-protected network node, a "break-in" might refer to a cracked or leaked password. We can equivalently think of $p_{\mathbf{v}}$ as the "probability of privacy failure" at vertex $\mathbf{v}$.

Similarly, the *probability of data loss*, $q_{\mathbf{v}}$, for a vertex $\mathbf{v}$ is the probability that enough servers are lost or destroyed (or unavailable for some other reason) that we are no longer able to reconstruct any of the data at vertex $\mathbf{v}$. The phrase "data loss' generically refers to any situation where where we cannot access data from a server – depending on the application, we may be concerned about temporary data loss (e.g., misplaced data, network outages), or

[4]See [8] for a full discussion of logical equivalence.

permanent (e.g., media failures, bit rot). Returning to Figure 1, if server **b** was a DVD, "data loss" might mean that the DVD is damaged or simply misplaced. We can think of $q_{\mathbf{v}}$ as the "probability of preservation failure" at **v**.

Together, the probabilities of break-in and data loss are referred to as *failure probabilities*. To gain an intuition, consider Figure 2, which shows the configuration $F_\Theta = ab+bc$. Assume that each terminal vertex is broken into with probability $p_{\mathbf{a}} = p_{\mathbf{b}} = p_{\mathbf{c}} = \frac{1}{4}$, mutually independent. An attacker wishing to reconstruct **r** must do one of three things. He must either break into terminals $\{\mathbf{a}, \mathbf{b}\}$ only, terminals $\{\mathbf{b}, \mathbf{c}\}$ only or all three of $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$. Therefore, $p_{\mathbf{r}}$, the probability of break-in at the root will be the sum of the probabilities of these three mutually exclusive events. That is, $p_{\mathbf{r}} = 2\left(\frac{1}{4}\right)^2 \frac{3}{4} + \left(\frac{1}{4}\right)^3 = \frac{7}{64}$. Using similar reasoning, we also conclude that $p_{\mathbf{d}} = p_{\mathbf{e}} = \frac{1}{16}$. Now, assume further that the probability of data loss at each terminal is also $q_{\mathbf{a}} = q_{\mathbf{b}} = q_{\mathbf{c}} = \frac{1}{4}$. If any of the following sets of terminals are lost, the data at **r** cannot be reconstructed: $\{\mathbf{b}\}$, $\{\mathbf{a}, \mathbf{b}\}$, $\{\mathbf{b}, \mathbf{c}\}$, $\{\mathbf{a}, \mathbf{c}\}$ or $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$. Thus, the probability of data loss at the root, $q_{\mathbf{r}} = \frac{1}{4}\left(\frac{3}{4}\right)^2 + 3\left(\frac{1}{4}\right)^2 \frac{3}{4} + \left(\frac{1}{4}\right)^3 = \frac{19}{64}$. We also conclude that $q_{\mathbf{d}} = q_{\mathbf{e}} = \frac{7}{16}$.

Formally, we define a pair of independent probability spaces $(\Omega_p, \mathbb{P})$ and $(\Omega_q, \mathbb{Q})$, which represent an adversary's attempts to break-into and destroy our data, respectively. Ω_p and Ω_q are the *sample spaces*, where each elementary outcome $\omega \in \Omega_p$ represents a specific subset of $\mathcal{X}$ that the attacker manages to break into whereas each $\omega \in \Omega_q$ represents a subset of $\mathcal{X}$ that the attacker manages to destroy (resulting in data loss). Thus, $\Omega_p = \Omega_q = 2^{\mathcal{X}}$. $\mathbb{P}$ and $\mathbb{Q}$ are *discrete probability measures* over Ω_p and Ω_q. Therefore, if $\mathcal{T}_\Theta$ and $\mathcal{F}_\Theta$ are, respectively, the sets of satisfying and falsifying assignments[5] of F_Θ, then $p_{\mathbf{r}} = \mathbb{P}(\mathcal{T}_\Theta)$ and $q_{\mathbf{r}} = \mathbb{Q}(\mathcal{F}_\Theta)$.

3.1.2 Damage and Loss

For each data object $\mathbf{o}_i \in \mathcal{O}$, we define the *sensitivity*, $d_i \equiv d(\mathbf{o}_i)$, as the damage or "harm" to us if an attacker gains (unauthorized) access to $\mathbf{o}_i$[6]. It is most natural to think of d_i as a measure of how "sensitive" data object $\mathbf{o}_i$ is – how much damage would be done if an unauthorized party gained access to $\mathbf{o}_i$, or, how "valuable" is $\mathbf{o}_i$ to an attacker? For example, if $\mathbf{o}_1$ was a secret document and $\mathbf{o}_2$ was a music file, then we might expect that $d_1 \gg d_2$.

Suppose we use a strategy $\mathcal{S}$ to distribute an object $\mathbf{o}_1$. An attacker can cause damage d_1 only by breaking into enough servers to reconstruct $\mathbf{o}_1$. Formally, we define indicator random variables, $\{A_i : \Omega_p \to \{0, 1\}, i \in 1, \ldots, |\mathcal{O}|\}$, where $A_i(\omega) = 1$ if and only if the data stored at servers $\omega \in \Omega_p$ is sufficient to reconstruct $\mathbf{o}_i$. The *realized damage*, $D : \Omega_p \to [0, d_0]$ is a random variable on Ω_p, where $D(\omega) \equiv \sum_i d_i A_i(\omega)$ for each $\omega \in \Omega_p$, and $d_0 \equiv \sum_i d_i$ is the sum-total sensitivity of all the objects in $\mathcal{O}$. In words, if an attacker breaks into servers $\omega \subseteq \mathcal{X}$, the total damage caused is simply the sum of the sensitivities of the data objects that he can reconstruct using ω.

Completely analogous to d_i, we can define the *value*, $l_i \equiv l(\mathbf{o}_i)$, as the cost or "harm" to us if data object $\mathbf{o}_i$

[5] Clearly, $\mathcal{T}_\Theta \subseteq 2^{\mathcal{X}}$ and $\mathcal{F}_\Theta \subseteq 2^{\mathcal{X}}$.

[6] Notationally, we choose the symbol d_i to represent *sensitivity* so that the reader may associate it with with the damage, D. For similar reasons, we will use l_i to denote *value*, to connect it with the loss, L.

is lost. Intuitively, d_i is a measure of how useful or valuable $\mathbf{o}_i$ is to us. Suppose, in our example, that we had a backup copy of document $\mathbf{o}_1$ in our e-mail, but no backup of the music file $\mathbf{o}_2$. In that case, probably $l_2 \gg l_1$.

The value l_i is lost only if so many servers are lost (or destroyed, or otherwise unavailable) that reconstruction of $\mathbf{o}_i$ is rendered impossible. Define indicators $\{B_i : \Omega_q \to \{0,1\}, i \in 1,\ldots,|\mathcal{O}|\}$, where $B_i(\omega) = 1$ if and only if losing access to $\omega \in \Omega_q$ would mean that we could no longer reconstruct $\mathbf{o}_i$. The *realized loss*, $L : \Omega_q \to [0, l_0]$ is a random variable on Ω_q, where $L(\omega) \equiv \sum_i l_i B_i(\omega)$ for each $\omega \in \Omega_q$, and $l_0 \equiv \sum_i l_i$ is the sum-total of values across all the data objects being distributed using $\mathcal{S}$.

3.1.3 Expected Damage and Expected Loss

One way to measure the privacy offered by a strategy $\mathcal{S}$ is the *expected damage*, $\bar{D} \equiv \mathbb{E}^{\mathbb{P}}[D]$. Similarly, preservation can be measured by the *expected loss*, $\bar{L} \equiv \mathbb{E}^{\mathbb{Q}}[L]$. We want both $\bar{D}$ and $\bar{L}$ to be as small as possible. Observe that the expectation $\bar{D}$ is computed with respect to the break-in probability measure $\mathbb{P}$, whereas $\bar{L}$ is computed under the data loss probability measure $\mathbb{Q}$. Intuitively, $\bar{D}$ and $\bar{L}$ are the "costs"[7] we expect to incur, on average, for using $\mathcal{S}$ to distribute our collection of data objects, $\mathcal{O}$.

It can be shown that, if Partition operators are disallowed, selecting the configuration that minimizes $\bar{D}$ and $\bar{L}$ is equivalent to minimizing $p_{\mathbf{r}}$ and $q_{\mathbf{r}}$, respectively. However, the converse is untrue – that is, if Partition operators are allowed, then a configuration that minimizes $\bar{D}$ and $\bar{L}$ does not necessarily minimize $p_{\mathbf{r}}$ and $q_{\mathbf{r}}$, and vice versa. We will leverage this observation when we search for "good" configurations, in Section 4.

3.1.4 Second Moments

We may also wish to control σ_D and σ_L, the standard deviations of D and L, respectively. Similar to the means $\bar{D}$ and $\bar{L}$, the standard deviations are also measures of privacy and preservation provided by a system. However, σ_D and σ_L capture the risk of "catastrophic" privacy breaches or data loss where *all* our sensitive data is leaked or lost[8]. Partition operators are useful for reducing σ_D and σ_L since they distribute data across storage locations whose failures are (hopefully) weakly correlated, or independent.

3.2 Performance

A natural metric for performance is the amount of time it takes to access our data. To make this idea concrete, we define a *command* as either a read or a write of a single data object. A command is the basic *unit of work* in our model. For a given strategy $\mathcal{S}$, our measure of performance is the *expected running time*, $\bar{T} \equiv \mathbb{E}^{\mathbb{F}}[T]$, which is the average execution time of a single command (we will define our notation shortly). There are a number of

[7] The economic notion of *expected utility* is closely related. Expected damage can be thought of as the expected utility of an attacker, whereas expected loss is the expected disutility of a defender.

[8] σ_D and σ_L are reminiscent of "risk" in portfolio optimization.

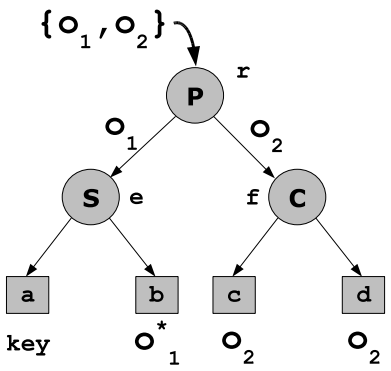

Figure 5: The configuration $F_\Theta = (\mathbf{ab})(\mathbf{c} + \mathbf{d})$.

factors that affect $\bar{T}$. For brevity, we will illustrate a few of these factors using a small example. We will notice immediately that the chosen operator implementations have a large impact on performance.

Suppose we are using the strategy $\mathcal{S}$ shown in Figure 5, with $F_\Theta = (\mathbf{ab})(\mathbf{c} + \mathbf{d})$, to distribute two documents $\mathbf{o}_1$ and $\mathbf{o}_2$, each of size $s_1 = s_2 = 1$ MB. The Partition $\mathbf{r}$ sends $\mathbf{o}_1$ to $\mathbf{e}$ and $\mathbf{o}_2$ to $\mathbf{f}$. The Split $\mathbf{e}$ is implemented using AES with 128-bit keys, where the key is stored at $\mathbf{a}$ and the ciphertext $\mathbf{o}_1^*$ is stored at $\mathbf{b}$. We will demonstrate how to compute $\bar{T}$ – we begin at the terminals, and work our way "up" the configuration to $\mathbf{r}$.

Suppose we issue a read command involving $\mathbf{o}_1$. We must read data from both $\mathbf{a}$ and $\mathbf{b}$ to compute $\mathbf{e}$. Observe that we do not need to reconstruct $\mathbf{f}$, however. Let $T_{\mathbf{a}}^R(s)$ be the time needed to read an object of size s from vertex $\mathbf{a}$. Then $T_{\mathbf{a}}^R(s) = st_{\mathbf{a}}$, where $t_{\mathbf{a}}$ is the *access time* in seconds per MB for the storage device $\mathbf{a}$. Since a key is stored at $\mathbf{a}$, we require $s_{key} = 128$ bits of data from server $\mathbf{a}$. Assume that $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$ are network nodes from which we can transfer data at 10 MB per second. Thus, $t_{\mathbf{a}} = \frac{1}{10}$, so $T_{\mathbf{a}}^R(s_{key}) = \frac{1}{10} \cdot \frac{1}{64} = 1.56$ milliseconds. Similarly, $T_{\mathbf{b}}^R(s) = T_{\mathbf{c}}^R(s) = T_{\mathbf{d}}^R(s) = \frac{s}{10}$ seconds.

Let $T_{\mathbf{e}}^R(s)$ be the time needed to reconstruct an object of size s from the non-terminal vertex $\mathbf{e}$. $T_{\mathbf{e}}^R(s)$ depends on $t_{\mathbf{e}}$, the *processing time* per MB of input data at $\mathbf{e}$. Since the Split at $\mathbf{e}$ is implemented using AES, $t_{\mathbf{e}} = t_{AES}$, where t_{AES} is the processing time for AES.

$T_{\mathbf{e}}^R(s)$ will also depend on our choice of *computation model*. That is, are the accesses of $\mathbf{a}$ and $\mathbf{b}$ done in parallel or serial? In a *parallel* model, we can read in $\mathbf{a}$ and $\mathbf{b}$ in at the same time, so $T_{\mathbf{e}}^R(s) = st_{\mathbf{e}} + \max\{T_{\mathbf{a}}^R(s_{key}), T_{\mathbf{b}}^R(s)\}$. In a *serial* model, we can only read one of $\mathbf{a}$ and $\mathbf{b}$ at a time, so $T_{\mathbf{e}}^R(s) = st_{\mathbf{e}} + T_{\mathbf{a}}^R(s_{key}) + T_{\mathbf{b}}^R(s)$. We will assume a parallel model for the rest of this example. The input to $\mathbf{e}$ is $s_1 = 1$ MB in size. Assuming $t_{\mathbf{e}} = t_{AES} = \frac{1}{20}$, we get $T_{\mathbf{e}}^R(s_1) = \frac{1}{20} \cdot 1 + \max\{\frac{1}{10}, \frac{1}{640}\} = 150$ milliseconds.

Finally, the Partition at $\mathbf{r}$ requires little computational effort, since no encoding is done (a similar statement applies for Copy). So, in our model, we simply set $t_{\mathbf{r}} = 0$ i.e., $\mathbf{P}$ and $\mathbf{C}$ operators come "for free". We conclude that the time required to read $\mathbf{o}_1$ is 150 milliseconds.

Description	Symbol
Probability of break-in at vertex $\mathbf{v}$	$p_{\mathbf{v}}$
Break-in probability measure	$\mathbb{P}$
Expected damage	$\bar{D}$
Standard deviation of damage	σ_D
Sensitivity of object $\mathbf{o}_i$	d_i
Probability of data loss at vertex $\mathbf{v}$	$q_{\mathbf{v}}$
Data loss probability measure	$\mathbb{Q}$
Expected loss	$\bar{L}$
Standard deviation of loss	σ_L
Value of object $\mathbf{o}_i$	l_i
Access time per MB of server $\mathbf{x} \in \mathcal{X}$	$t_{\mathbf{x}}$
Processing time per MB at vertex $\mathbf{y} \in \mathcal{Y}$	$t_{\mathbf{y}}$
Read time for an object of size s from vertex $\mathbf{v}$	$T^R_{\mathbf{v}}(s)$
Write time for an object of size s from vertex $\mathbf{v}$	$T^W_{\mathbf{v}}(s)$
Frequency of access for object $\mathbf{o}_i$	f_i
Read workload for object $\mathbf{o}_i$	λ_i
Data access probability measure	$\mathbb{F}$
Expected running time	$\bar{T}$
Size of object $\mathbf{o}_i$	s_i

Table 1: Parameters used in measuring the 3 P's.

Now, suppose we want to read $\mathbf{o}_2$ instead. We need $\mathbf{f}$, and consequently either $\mathbf{c}$ or $\mathbf{d}$, since they are both identical copies of $\mathbf{f}$. Thus, a read of $\mathbf{o}_2$ requires $T^R_{\mathbf{f}}(s_2) = t_{\mathbf{f}} s_2 + \min\{T^R_{\mathbf{c}}(s_2), T^R_{\mathbf{d}}(s_2)\} = 100$ milliseconds.

Whereas a read of $\mathbf{o}_2$ requires accessing either one of $\mathbf{c}$ or $\mathbf{d}$, a write or update of $\mathbf{o}_2$ requires both $\mathbf{c}$ and $\mathbf{d}$ to be updated (i.e., to keep the copies consistent). Thus, we expect $T^W_{\mathbf{f}}(s) \geq T^R_{\mathbf{f}}(s)$, where $T^W_{\mathbf{f}}(s)$ is the time needed to write (or update) an object of size s to $\mathbf{f}$. Writing or updating $\mathbf{o}_1$, on the other hand, requires accessing vertices $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{e}$, which is the same as if we were reading $\mathbf{o}_1$. However, the access times and processing times at each vertex may be different for reads and writes e.g., decryptions may be faster than encryptions. As such, we might also expect $T^W_{\mathbf{e}}(s) \geq T^R_{\mathbf{e}}(s)$. For this example, we assume that $T^W_{\mathbf{e}}(s) = T^W_{\mathbf{f}}(s) = 200s$ milliseconds.

We are now in a position to compute $\bar{T}$. Let f_1 and f_2 be the *access frequencies* of $\mathbf{o}_1$ and $\mathbf{o}_2$, respectively – we have $\sum_i f_i = 1$, so the access frequency f_i can be thought of as the probability that a command will access object $\mathbf{o}_i$. Let λ_i be the *read workload*, which is the fraction of commands involving $\mathbf{o}_i$ that are reads. Thus, $1 - \lambda_i$ is the *write workload* for $\mathbf{o}_i$. Combining these quantities, we can conclude that expected running time for a command is $\bar{T} = f_1\lambda_1 T^R_{\mathbf{e}}(s_1) + f_2\lambda_2 T^R_{\mathbf{f}}(s_2) + (1-\lambda_1) f_1 T^W_{\mathbf{e}}(s_1) + (1-\lambda_2) f_2 T^W_{\mathbf{f}}(s_2)$. Using $f_1 = f_2 = 0.5$ and $\lambda_1 = \lambda_2 = 0.8$, we get $\bar{T} = 140$ milliseconds per command.

3.3 Summary

Table 1 summarizes the parameters involved in quantifying privacy (i.e., damage), preservation (i.e., loss) and performance (i.e., running time) for a strategy, $\mathcal{S}$.

4 Optimization

Now that we have well-defined metrics of privacy, preservation and performance for any given strategy $\mathcal{S}$ (i.e., $\bar{U}, \bar{D}, \sigma_U, \sigma_D$ and $\bar{T}$), we are in a position to search for the "best" strategy under these metrics. However, it is not so simple, because of the tradeoffs involved. Steps taken to improve privacy (**S** operators) tend to worsen preservation, and vice versa (**C**). Moreover, using **P** operators to improve performance can adversely impact both privacy and preservation, whereas **k**/**n** can do the exact opposite. Therefore, the correct approach is to solve constrained optimization problems such as (1):

$$\begin{aligned} \min_{\mathcal{S}} \quad & \bar{T} \\ \text{s.t.} \quad & \bar{D} \leq D_0 \\ & \bar{L} \leq L_0 \end{aligned} \tag{1}$$

In words (1) says: *Given a collection of data objects $\mathcal{O}$, servers $\mathcal{X}$, and minimal requirements D_0 and L_0 for privacy and preservation, find the strategy $\mathcal{S}$ that offers maximal performance.* Alternatively, we may want to optimize or constrain σ_D or σ_L, or some weighted combination of all of $\bar{D}, \bar{L}, \sigma_D, \sigma_L$ and $\bar{T}$.

Problems such as (1), though well-posed, are difficult to solve. In fact, a much simpler version is considered in [7] – given a single data object $\mathbf{o}_1$, some servers $\mathcal{X}$ and a lower bound on probability of data loss, find the configuration that minimizes the probability of break-in:

$$\begin{aligned} \min_{\Theta} \quad & p_{\mathbf{r}} \\ \text{s.t.} \quad & q_{\mathbf{r}} \leq q_0 \end{aligned} \tag{2}$$

Since there was only a single data object, and performance was not being modeled, it was sufficient to find the best configuration Θ[9]. Unfortunately, even solving (2) exactly is intractable for modestly-sized problem instances. As such, it is unlikely that a tractable method for finding the global optimum in problems such as (1) can be found.

Instead, suppose we restrict our search space to strategies where the configuration is a J-way Partition at the root, and the children of the Partition are *subsystems* $\{\mathcal{S}_j, j \in 1, \ldots, J\}$, as illustrated in Figure 6. The subsystems are comprised exclusively of **S**, **C** and **k**/**n** operators (i.e., no **P**), and they are mutually disjoint in that they do not have any vertices in common.

It does not seem that we lose much by restricting the search space in this way. We studied a number of examples that were outside this restricted space. In each case, we found a system within our search space whose $\bar{D}$, $\bar{L}$ and

[9] In [7], the symbols $P(\Theta)$ and $Q(\Theta)$ are used instead of $p_{\mathbf{r}}$ and $q_{\mathbf{r}}$. Also, $q_{\mathbf{r}}$ could be minimized while constraining $p_{\mathbf{r}}$.

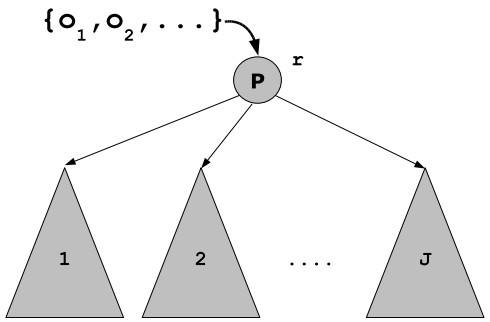

Figure 6: Partition at $\mathbf{r}$, with J subsystems.

$\bar{T}$ were roughly the same. Therefore, although such a restriction implies that the resulting strategy $\mathcal{S}$ will most likely be suboptimal, we conjecture that $\mathcal{S}$ may not be far from optimal. Each step involved in searching through this restricted space can be done near-optimally (finding the global optimum in some cases). Moreover, each step of the search is actually tractable. Therefore, we propose the following technique, referred to as *Algorithm DLT*, for finding approximate solutions to (1):

1. Select J and integers $\{m_j, j \in 1, \ldots, J\}$ such that $\sum_j m_j = |\mathcal{X}|$.
2. Divide servers $\mathcal{X}$ into J subsets $\{\mathcal{X}_j, j \in 1, \ldots, J\}$ such that $|\mathcal{X}_j| = m_j$, and failures across subsets are mutually independent.
3. For $j \in 1, \ldots, J$, solve (2) using $\mathcal{X}_j$ – find the configuration Θ_j with the lowest $p_{\mathbf{r}}$, given servers $\mathcal{X}_j$ and a lower bound on $q_{\mathbf{r}}$ (or, vice versa). In each Θ_j, ensure that there are no **P** operators.
4. Choose operator implementations I_j for each Θ_j such that $\bar{T}_j$ is minimized for each subsystem[10] $\mathcal{S}_j \equiv (\Theta_j, I_j)$.
5. We now have J subsystems $\{\mathcal{S}_j, j \in 1, \ldots, J\}$. Use a **P** at the root to allocate $\mathcal{O}$ across the subsystems, such that $\bar{D}$, $\bar{L}$ and $\bar{T}$ are optimized.

We will devote the remainder of this Section to discussing each step of our proposed solution technique.

4.1 Steps 1 and 2

In Steps 1 and 2, we assign each of the servers to one of J subgroups, $\mathcal{X}_j$. Although it is unclear how to choose J and $\{m_j\}$ optimally, we might try different choices, repeat Algorithm DLT a few times, and keep the best result.

A special (but quite common) case is where all the servers have identical characteristics (i.e., each $\mathbf{x} \in \mathcal{X}$ has the same failure probabilities, capacity and access time), in which case Step 2 is trivial. Alternatively, $\mathcal{X}$ might be "almost" homogeneous. For example, perhaps half the servers in $\mathcal{X}$ are fast and secure (and expensive), while the

[10]The notation $\mathcal{S}_j \equiv (\Theta_j, I_j)$ is somewhat abusive since we have not specified any data objects, $\mathcal{O}_j$.

other half are cheaper, less secure, slower machines. Heuristics for dividing up $\mathcal{X}$ may perform well here. Perhaps we should choose $J = 2$, with all the faster servers in one group all the slower servers in the other. Alternatively, each group could have equal numbers of fast and slow machines. Although such heuristics may yield good results, we are unable to say so with certainty, since we are unable to compare to the global optimum.

4.2 Step 3

Given $\{\mathcal{X}_j, j \in 1, \ldots, J\}$ from Steps 1 and 2, the next step is to find the "best" configuration Θ_j using each $\mathcal{X}_j$. That is, given $\mathcal{X}_j$, find the Θ_j that solves (2).

In [7], an efficient algorithm – which we refer to as *Algorithm PQ* – is given for computing approximate solutions to (2). Algorithm PQ finds a configuration comprised of **C**, **S** and **k**/**n** operators only. As such, optimizing over $p_{\mathbf{r}}$ and $q_{\mathbf{r}}$ in Step 3 is equivalent to optimizing over $\bar{D}_j$ and $\bar{L}_j$ for each subsystem $\mathcal{S}_j$, irrespective of the operator implementation choices. By disallowing **P** operators in the subsystems, we are greatly reducing the complexity of solving (1). We do not need to jointly optimize over configuration choice and implementation choice. Instead, we decouple the task into individual optimizations over configuration (Step 3) and operator implementation choices (Step 4). Our choices in Step 4 will not change the $p_{\mathbf{r}}$ and $q_{\mathbf{r}}$ values achieved in Step 3.

4.3 Step 4

Step 4 is to search for "optimal" implementations for each operator in Θ_j, such that $\bar{T}_j$ is minimized. We use a dynamic programming approach to reduce the complexity of the search. The idea is to compute a topological sort, G_j, of the vertices in Θ_j. First traverse the list in reverse order (i.e., terminals first, root last), computing $T^R_{\mathbf{v}}(s)$ and $T^W_{\mathbf{v}}(s)$ as a function of the input size s, at each vertex $\mathbf{v} \in G_j$. Then, traversing G_j in forward order (i.e., root first), select the implementation at each $\mathbf{v}$ that minimizes the "average" running time at $\mathbf{v}$, and compute the size of the output data objects that will be sent to $\mathbf{v}$'s children.

Choosing an implementation at each vertex $\mathbf{v} \in G_j$ is a balance between the processing time spent at $\mathbf{v}$ and the reduction achieved in output data size. It can be shown that minimizing the average running time at each $\mathbf{v} \in G_j$ will find the globally optimum implementations for Θ_j. We omit the details here.

4.4 Step 5

We now have J subsystems, $\{\mathcal{S}_j, j \in 1, \ldots, J\}$. For each $\mathcal{S}_j$, let $p^{(j)}_{\mathbf{r}}$ and $q^{(j)}_{\mathbf{r}}$ be the resulting probabilities of break-in and data loss at the root, $\mathbf{r}^{(j)}$, of Θ_j. Define $T^{R,(j)}_{\mathbf{r}}(s)$ and $T^{W,(j)}_{\mathbf{r}}(s)$ as the time required to read and write a data object of size s using $\mathcal{S}_j$. We are able to compute $p^{(j)}_{\mathbf{r}}$, $q^{(j)}_{\mathbf{r}}$, $T^{R,(j)}_{\mathbf{r}}(s)$ and $T^{W,(j)}_{\mathbf{r}}(s)$ for each $\mathcal{S}_j$. We now combine these subsystems into a single strategy $\mathcal{S}$ using a J-way Partition operator at the root, $\mathbf{r}$. The j^{th} child of $\mathbf{r}$ is

subsystem $\mathcal{S}_j$. See Figure 6.

All that remains to be decided is what implementation to choose for the Partition operator at the root. That is, we must allocate each data object $\mathbf{o}_i \in \mathcal{O}$ to a subsystem $\mathcal{S}_j$, such that the constraints in (1) are satisfied and our objective function in minimized. Allocating data objects is a combinatorial optimization problem, which in general is expensive to solve exactly. Fortunately, we can formulate good relaxations of problems such as (1) that are quite efficient to solve, as we show next.

Define indicator variables $z_{i,j} \in \{0,1\}$, where $z_{i,j} = 1$ if and only if $\mathbf{o}_i \in \mathcal{O}$ is sent to $\mathcal{S}_j$. Clearly, $\sum_j z_{i,j} = 1 \; \forall \; i$. Using $\{z_{i,j}\}$, we can express the privacy, preservation and performance for our strategy $\mathcal{S}$ as follows:

$$\bar{D} = \sum_{i,j} p_{\mathbf{r}}^{(j)} d_i z_{i,j} \tag{3}$$

$$\bar{L} = \sum_{i,j} q_{\mathbf{r}}^{(j)} l_i z_{i,j} \tag{4}$$

$$\bar{T} = \sum_{i,j} h_{i,j} z_{i,j} \tag{5}$$

$$\sigma_D^2 = \sum_j p_{\mathbf{r}}^{(j)} \left(1 - p_{\mathbf{r}}^{(j)}\right) \left(\sum_i d_i z_{i,j}\right)^2 \tag{6}$$

$$\sigma_L^2 = \sum_j q_{\mathbf{r}}^{(j)} \left(1 - q_{\mathbf{r}}^{(j)}\right) \left(\sum_i l_i z_{i,j}\right)^2 \tag{7}$$

where $h_{i,j} \equiv \lambda_i T_{\mathbf{r}}^{R,(j)}(s_i) + (1 - \lambda_i) T_{\mathbf{r}}^{W,(j)}(s_i)$. We can, therefore, rewrite (1) using these new expressions:

$$\begin{aligned} \min_{\{z_{i,j}\}} \quad & \sum_{i,j} h_{i,j} z_{i,j} \\ \text{s.t.} \quad & \sum_{i,j} p_{\mathbf{r}}^{(j)} d_i z_{i,j} \leq D_0 \\ & \sum_{i,j} q_{\mathbf{r}}^{(j)} l_i z_{i,j} \leq L_0 \\ & \sum_j z_{i,j} = 1 \; \forall \; i \\ & z_{i,j} \in \{0,1\} \; \forall \; i,j \end{aligned} \tag{8}$$

Problem (8) is an *integer program* in the variables $z_{i,j}$, which is expensive to solve. Observe, however, that the expressions (3), (4) and (5) are linear in $z_{i,j}$. Suppose we allow each $z_{i,j}$ to take on real values in $[0,1]$ instead of constraining them to be integers:

$$0 \leq z_{i,j} \leq 1 \; \forall \; i,j \tag{9}$$

Problem (8) with the *relaxed* constraints (9) is a *linear program*, and can be solved easily and efficiently.

At the optimum, the solution $\{z^*_{i,j}\}$ obtained under the relaxed constraints (9) is a good approximation to the solution of the integer program (8). In many problem instances, we find that $\{z^*_{i,j}\} \in \{0, 1\}$ for most (i, j). For the few $\{z^*_{i,j}\}$ values that are strictly between 0 and 1, we can simply round off to obtain a close approximation to the exact solution to (8).

The constraint relaxation in (9) would also allow us to solve integer programs (approximately) that include σ_D and σ_L in the objective or constraints. In such cases, the relaxed problems are, at worst, *quadratically-constrained quadratic programs*, for which very efficient solution algorithms exist as well. The case study in Section 5 considers one such problem instance.

To summarize, using the solution $\{z^*_{i,j}\}$ obtained from the constraint relaxation (9) is a viable, efficient technique for solving problems such as (8). The $\{z^*_{i,j}\}$ values tell us which subsystem $\mathcal{S}_j$ to allocate each data object $\mathbf{o}_i$ to, in order to obtain the desired tradeoff between privacy, preservation and performance.

5 Case Study

5.1 Overview

We present a case study to illustrate how the techniques in this paper (i.e., Algorithm DLT) can be applied in the design of data distribution strategies. Suppose we are managing a database of 100,000 electronic medical records on behalf of our client, a large healthcare facility. Each medical record contains sensitive information about a single patient such as their personal information, medical history and test results. We have a cluster of 20 networked storage servers across which we want to distribute the database with the aim of achieving good privacy, preservation and performance. Using our earlier notation, $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{20}\}$.

Under HIPAA regulations [1] in the USA[11], in the event that a breach of privacy occurs and patients' medical records are leaked to an unauthorized party, we would be subject to a fine of \$100 per patient[12] (i.e., per leaked record). Thus, in our case study the realized damage, D, corresponds to the total fines levied against us as a penalty for data leaked from our database. One of our aims in distributing the database will be to attain a low value for the expectation, $\bar{D}$, and the standard deviation, σ_D, of D. We are interested in the latter since it is a measure of "risk" (i.e., the likelihood of catastrophic break-ins).

Frequent tape backups are made of the entire database – thus, in our case study, we are not concerned about permanent data loss, per se. However, both the servers and the network infrastructure that connects them are prone to *outages* i.e., servers are unreliable and may go offline. Our service-level agreement with the healthcare facility

[11] Similar laws exist in the EU (95/46/EC) and Japan (HPB 517).
[12] There is a ceiling on the total fine, which we ignore here.

Name	Description
2x10	$J = 2$ sub-clusters of 10 servers
4x5	$J = 4$ sub-clusters of 5 servers
2x(6+4)	$J = 4$ sub-clusters of 6 and 4 servers, respectively
2x(8+2)	$J = 4$ sub-clusters of 8 and 2 servers, respectively
10x2s	$J = 10$ 2-way Split operators
10x2c	$J = 10$ 2-way Copy operators

Table 2: Candidate system designs.

requires us to provide an "uptime" of at least 99.9%. That is, at most 0.1% of reads and writes are allowed to fail. Thus, a second goal in distributing the database is to achieve our 99.9% uptime requirement. The challenge is to design a highly available strategy using a collection of relatively unreliable servers. In our case study, the realized loss, $\bar{L}$, corresponds to the "downtime" i.e., the fraction of reads and writes that fail due to outages. Finally, we also aim for a low $\bar{T}$, the average time required to read or write a record from our system.

5.2 Steps 1 and 2

In Steps 1 and 2 of Algorithm DLT, we divide our cluster of servers $\mathcal{X}$ into J subclusters $\mathcal{X}_1, \ldots, \mathcal{X}_J$. We will compare six possible system designs, listed in Table 2. The first four correspond to four separate executions of Algorithm DLT, whereas the final two systems in Table 2, *10x2s* and *10x2c*, are "naive" base cases that we can evaluate our technique's output against. Since the 20 servers have identical characteristics, it makes no difference exactly which servers are assigned to which sub-clusters – only the sizes of the sub-clusters are important. We will find that each proposed design has it's strengths and weaknesses – in particular we will observe a trade-off between privacy ($\bar{D}$ and σ_D) and performance ($\bar{T}$), for a fixed level of preservation, $\bar{L}$.

5.3 Step 3

In Step 3, we search for optimal configurations Θ_j using the servers in each sub-cluster $\mathcal{X}_j$. We estimate that for each server $\mathbf{x} \in \mathcal{X}$, the probability of break-in $p_{\mathbf{x}}$ (i.e., the probability that server $\mathbf{x}$'s data is leaked) is 1%, where break-ins occur mutually independently. Similarly, the probability $q_{\mathbf{x}}$ that any server $\mathbf{x}$ goes offline is estimated to be 5%, with failures occuring independently. Reads and writes of data from each server proceed at $t_{\mathbf{x}} = 10$ MB per second, including network latencies. These data are summarized in Table 3.

Using Algorithm PQ, we solve (2) for the sub-clusters of various sizes listed in Table 2. If we can achieve a 99.9% uptime for *every* sub-cluster, then no matter how we distribute the records across the J sub-clusters, we will satisfy our requirement of 99.9% overall uptime. So, in each execution of Algorithm PQ, we use $q_0 = 100\% - 99.9\% = 0.1\%$ as the upper bound on probability of data loss. The configurations (i.e., sub-clusters) output by Algorithm PQ are listed in Table 4. As an example, System *2x(6+4)* in Table 2 will have two sub-clusters of size $m_j = 6$ and two of

Description	Symbol	Value
Probability of break-in for server $\mathbf{x}$	$p_{\mathbf{x}}$	1%
Probability of data loss for server $\mathbf{x}$	$q_{\mathbf{x}}$	5%
Access time per MB for server $\mathbf{x}$	$t_{\mathbf{x}}$	10 sec.
Processing time for AES	t_{AES}	0.005 sec./MB
Processing time for XOR	t_{XOR}	0.001 sec./MB
Processing time for k-of-n	t_k	0.500 sec./MB
Size of AES encryption keys	s_{key}	128 bits
Size of each medical record	s_{rec}	100 kB
Read workload for each record	λ	80%

Table 3: Parameters used in case study.

m_j	F_{Θ_j}
2s	$\mathbf{x_1 x_2}$
2c	$\mathbf{x_1} + \mathbf{x_2}$
4	$f_{2/4}(\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4})$
5	$\mathbf{x_1}(\mathbf{x_4} + \mathbf{x_5} + \mathbf{x_2 x_3}) + f_{3/4}(\mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5})$
6	$\mathbf{x_1} f_{2/5}(\mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5}, \mathbf{x_6}) + \mathbf{x_2} f_{2/3}(\mathbf{x_4}, \mathbf{x_5}, \mathbf{x_6})$
8	$\mathbf{x_1} \cdot f_{4/7}(\mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5}, \mathbf{x_6}, \mathbf{x_7}, \mathbf{x_8}) +$ $\mathbf{x_2} \cdot f_{4/6}(\mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5}, \mathbf{x_6}, \mathbf{x_7}, \mathbf{x_8})$
10	$\mathbf{x_1}((\mathbf{x_2} + \mathbf{x_3})G + \mathbf{x_2 x_3} f_{4/7}(\mathbf{x_4}, \ldots, \mathbf{x_{10}}) +$ $\mathbf{x_8 x_9 x_{10}} f_{3/4}(\mathbf{x_4}, \mathbf{x_5}, \mathbf{x_6}, \mathbf{x_7}) +$ $(\mathbf{x_8 x_9} + \mathbf{x_{10}})\mathbf{x_4 x_5 x_6 x_7} + f_{7/9}(\mathbf{x_2}, \ldots, \mathbf{x_{10}})$ where $G = f_{5/6}(\mathbf{x_5}, \ldots, \mathbf{x_{10}}) + \mathbf{x_4}(f_{4/5}(\mathbf{x_6}, \ldots, \mathbf{x_{10}}) + \mathbf{x_5}($ $f_{3/4}(\mathbf{x_7}, \ldots, \mathbf{x_{10}}) + \mathbf{x_6}(f_{2/3}(\mathbf{x_8}, \mathbf{x_9}, \mathbf{x_{10}}) + \mathbf{x_7}(\mathbf{x_8} + \mathbf{x_9}))))$

Table 4: Configurations (i.e., sub-clusters) output by Step 3.

size $m_j = 4$.

Figure 7 is a plot of $p_{\mathbf{r}}$ and $q_{\mathbf{r}}$ for the configurations in Table 4, on a negative-log scale. For example, the sub-cluster of size $m_j = 8$ has $p_{\mathbf{r}} = 4.9 \times 10^{-9}$ and $q_{\mathbf{r}} = 0.095$. In each subcluster (other than the two base cases), the probability of data loss is less than 0.1%, as required. Observe that the subcluster with 4 servers has a lower $q_{\mathbf{r}}$ than the one with 6 servers. In the latter case, we were able to exploit slack in the constraint on $q_{\mathbf{r}}$ to achieve a much lower value for $p_{\mathbf{r}}$.

Recall that the configuration in each subcluster is comprised of Split, Copy and Threshold operators, but not Partitions. Moreover, operator implementations have not yet been chosen at each vertex – we select these in Step 4. For concreteness, in Figure 8 we illustrate the configuration output use for a subcluster of size $m_j = 5$.

5.4 Step 4

The next step is to select implementations for the operators in each of the configurations listed in Table 4. Our goal is to minimize the average running time of reads and writes within each configuration. Recall that, since there are no Partitions used in the subclusters, our choices of operator implementations will not affect the probabilities of failure achieved in Step 3.

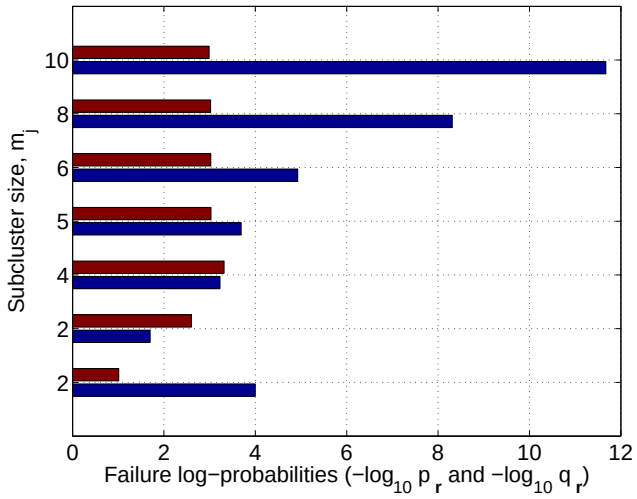

Figure 7: Log-probabilities of failure for configurations in Table 4. Longer bars correspond to lower probabilities of failure.

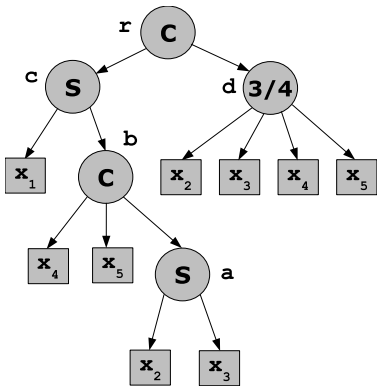

Figure 8: Subcluster with $m_j = 5$ servers.

In our case study, we consider two possible implementations for 2-way Split operators, namely AES and XOR with a randomly chosen bit sequence. For k-way Splits, $k \geq 3$, our only option is an XOR with $k-1$ bit sequences. Similarly, Threshold operators are implemented using a Shamir secret-sharing scheme. Thus, we have $\mathcal{I} = \{\text{AES}, \text{XOR}, \text{Shamir}\}$ and the only decisions we must make are how to implement 2-way Split operators. The processing times per MB of input data, for each operator implementation, are given in Table 3. While the exact processing times are not important, the key observations for our case study are that XOR runs more quickly than AES, whereas a k-of-n operator is orders of magnitude slower than either. We assume that the server access times and processing times are equal for reads and writes (this assumption is approximately true for AES and XOR), and that Copy operators are "free".

We use a dynamic programming approach to select operator implementations for each 2-way Split operator in each configuration in Table 4. As a concrete example, consider the configuration $F_\Theta = \mathbf{x_1}(\mathbf{x_4} + \mathbf{x_5} + \mathbf{x_2 x_3}) +$

$f_{3/4}(\mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5})$, illustrated in Figure 8. We assume that $\lambda = 80\%$ of commands are reads for all records.

We must select implementations $I(\mathbf{a})$ and $I(\mathbf{c})$ for vertices $\mathbf{a}$ and $\mathbf{c}$ (the other operators have only one implementation option). Vertex $\mathbf{a}$ appears after $\mathbf{c}$ in any topological sort of the 2-way Split vertices, so we consider $\mathbf{a}$ first. Suppose the input to $\mathbf{a}$ was a data object of size s. If $I(\mathbf{a}) = \text{AES}$, then s MB of ciphertext would be stored at $\mathbf{x}_2$ and a key of size s_{key} would be stored at $\mathbf{x}_3$ (see Table 3). Therefore, to write the data object to $\mathbf{a}$ would require $s(t_{AES} + t_{\mathbf{x}}) + s_{key}t_{\mathbf{x}}$ seconds. On the other hand, if $\mathbf{a}$ was XOR, a write would require $s(t_{XOR} + 2t_{\mathbf{x}})$ since XOR outputs two data objects of size equal to the input. Therefore, the time required to write an object of size s is $T^W_{\mathbf{a}}(s) = st_{\mathbf{x}} + \min\{st_{AES} + s_{key}t_{\mathbf{x}}, st_{XOR} + st_{\mathbf{x}})\}$. A similar calculation shows that, in this case, the read and write times for $\mathbf{a}$ are equal i.e., $T^R_{\mathbf{a}}(s) = T^W_{\mathbf{a}}(s) = \bar{T}_{\mathbf{a}}$. Therefore, we prefer $I(\mathbf{a}) = \text{AES}$ if $s(t_{XOR} - t_{AES}) + (s - s_{key})t_{\mathbf{x}} > 0$, and $I(\mathbf{a}) = \text{XOR}$ otherwise.

We use the functions $T^R_{\mathbf{a}}(s)$ and $T^W_{\mathbf{a}}(s)$ in selecting an implementation for $\mathbf{c}$. For example, if $I(\mathbf{c}) = \text{AES}$ and the key is sent to vertex $\mathbf{b}$, then $T^W_{\mathbf{c}}(s) = s(t_{AES} + t_{\mathbf{x}}) + 2s_{key}t_{\mathbf{x}} + T^W_{\mathbf{a}}(s_{key})$ and $T^R_{\mathbf{c}}(s) = s(t_{AES} + t_{\mathbf{x}}) + s_{key}t_{\mathbf{x}}$. On the other hand, if $I(\mathbf{c}) = \text{XOR}$, then $T^W_{\mathbf{c}}(s) = s(t_{XOR} + t_{\mathbf{x}}) + 2st_{\mathbf{x}} + T^W_{\mathbf{a}}(s)$ and $T^R_{\mathbf{c}}(s) = s(t_{XOR} + 2t_{\mathbf{x}})$.

Note that the data input to $\mathbf{c}$ is a copy of the input data at the root $\mathbf{r}$, so $s_{\mathbf{c}} = s_{\mathbf{r}}$. Evaluating $\bar{T}_{\mathbf{c}} = \lambda T^R_{\mathbf{c}}(s_{\mathbf{c}}) + (1 - \lambda)T^W_{\mathbf{c}}(s_{\mathbf{c}})$ for various $s_{\mathbf{r}}$, we find that for small $s_{\mathbf{r}}$ (relative to s_{key}) we prefer an XOR at $\mathbf{c}$ while for moderate to large $s_{\mathbf{r}}$ we prefer AES. Intuitively, the reason is that for small data objects the largest component of the running time is the processing time, while for large data objects the server access times become dominant.

In our case study, we assume the medical records are each roughly $s_{rec} = 100$ kB in size, which is much larger than s_{key}. We therefore choose $I(\mathbf{c}) = \text{AES}$. This causes objects of size s_{key} to be sent to $\mathbf{a}$. Consequently, we choose $I(\mathbf{c}) = \text{XOR}$. These choices are reflective of a "rule of thumb". Whenever server access times are not negligible, it is generally desirable to avoid using k-way XOR operations on anything but encryption keys, since the data size is multiplied by a factor k. If unencoded data is input to a 2-way Split, we will typically use encryption to output a data object of equal size to the input (the ciphertext) along with a much smaller key. Further key management can then be done efficiently using XOR operations, if needed. For similar reasons, as a rule of thumb, we "re-factor" k-way Splits with $k > 2$ into a 2-way encryption and a secondary $(k-1)$-way XOR operator to manage the resulting keys.

In this manner, we select implementations and calculate the resulting execution times for each sub-cluster in Table 4 i.e., the amount of time to read and write a data object of size s. The results are plotted in Figure 9, for $s = s_{rec}$. The subclusters of various sizes have comparable read speeds, whereas write speeds vary widely, since more servers typically have to be updated in a write.

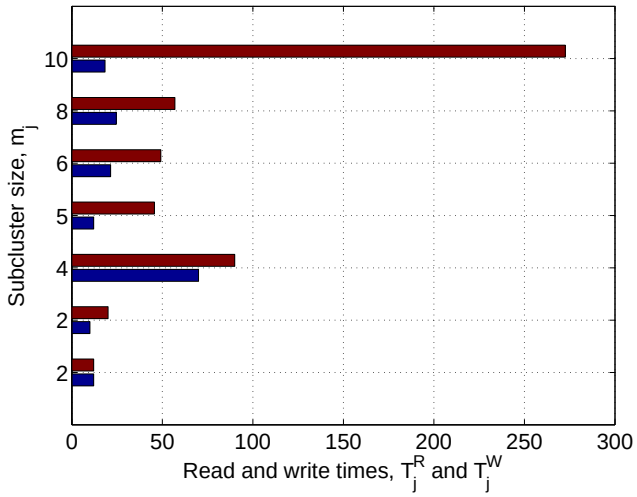

Figure 9: Read and write times in ms, assuming $s = 100$ kB.

5.5 Step 5

The final step in Algorithm DLT is to combine the J subclusters using a single J-way Partition operator at the root, as described in Table 2. It only remains to decide how the implement the Partition i.e., how to distribute the medical records database across the subclusters designed in Steps 1 to 4.

As described earlier, there are 100,000 medical records in the database. Some records are accessed more frequently than others. In particular, we observe that the distribution of access frequencies of individual medical records follows a power law with exponent $\frac{1}{2}$. One reason for a such a distribution over access times may be that the healthcare facility deals with a wide range of patients and as a patient ages, the need for visits to a physician are increased. We assume that the records are sorted by access frequency i.e., if f_k is the fraction of reads and writes that involve record k, then $f_k \propto \frac{1}{\sqrt{k}}$.

We adopt the following data distribution strategy in each proposed design. We divide the sorted list of medical records into $N = 20$ blocks, such that each block is accessed $\frac{1}{N}$ of the time. Doing so requires us to find the cumulative distribution of record access frequencies, and compute percentiles $\frac{100}{N}i$ for $i \in 1, \ldots, N$ – we omit the details here. Let $\mathcal{O} \equiv \{\mathbf{o}_i, i \in 1, \ldots, N\}$, where $\mathbf{o}_i$ is the i^{th} block of records. We will distribute the N blocks in $\mathcal{O}$ across the J subclusters that we designed in Steps 1-4. Due to the square-root law distribution of access frequencies, the number of records in each block increases (roughly) linearly in the block number. The number of records in each block is plotted in Figure 10.

Let n_i be the number of records in block $\mathbf{o}_i$, as plotted in Figure 10. Then, $s_i = n_i s_{rec}$. Recall that if there is a privacy breach, a fine of \$100 is levied per leaked record – thus, $d_i = 100 n_i$. Since each block receives an equal share of accesses by design $f_i = \frac{1}{N}$, and since our notion of loss corresponds to downtime (i.e., the access frequency

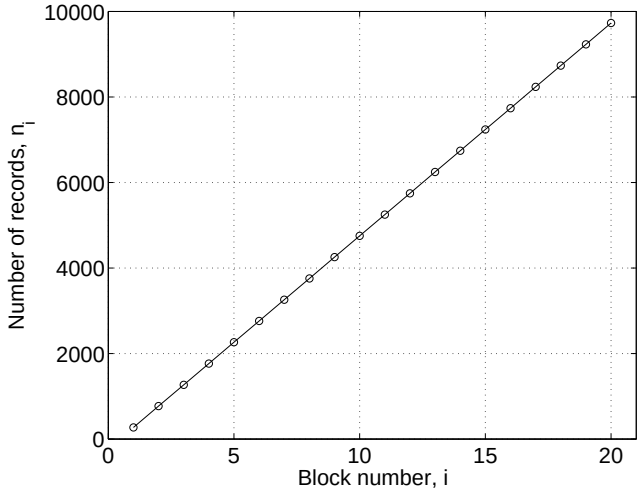

Figure 10: Number of records in block i.

of unavailable data), we get $l_i = f_i = \frac{1}{N}$.

Our goal is to minimize $\bar{D}$, σ_D, and $\bar{T}$, subject to an uptime requirement of 99.9%. There are several different problems that we might formulate. For example:

$$\begin{aligned} \min_{\mathcal{S}} \quad & \sigma_D \\ & \bar{L} \leq 0.1\% \end{aligned} \tag{10}$$

Or, we could select constants α, β and σ_0 and solve:

$$\begin{aligned} \min_{\mathcal{S}} \quad & \alpha\bar{D} + \beta\bar{T} \\ & \sigma_D \leq \sigma_0 \\ & \bar{L} \leq 0.1\% \end{aligned} \tag{11}$$

In this case study, we solve (10) for each design proposed in Table 2. The results of the optimization for each proposed design are given in Table 5. We see from the results that System *2x10* provides the lowest $\bar{D}$ and σ_D, but has a relatively high $\bar{T}$. The "base case" systems are the opposite – very low running times due to their simplicity, but weak in terms of privacy and preservation. Arguably, System *2x(8+2)* is an attractive compromise, performing relatively well under all measures.

It is interesting that System *2x(8+2)* allocates only blocks 1 and 2 (i.e., the smallest blocks containing the most frequently accessed records) to the clusters of size 2 (which are the faster but less secure). All other blocks are sent to the clusters of size 8, since the risk of leaking larger blocks of records is deemed to be unjustified.

System	σ_D ($)	$\bar{D}$ ($)	$\bar{L}$	$\bar{T}$ (ms)
2x10	10.31	$\approx 2.12 \times 10^{-5}$	0.102%	221.7
2x(8+2)	500.48	≈ 0.08	0.097%	50.5
2x(6+4)	24066.92	236.35	0.090%	44.5
4x5	71654.00	2039.08	0.093%	38.9
10x2s	31802.63	999.99	9.75%	12.1
10x2c	443268.19	198998.01	2.5%	18.0

Table 5: Summary of results.

Solving (11) instead would lead to slightly different solutions. In particular, the solution would depend on the constants α, β and σ_0, which encode an "exchange rate" between $\bar{D}$, σ_D and $\bar{T}$. Depending on the objective function, we get different allocations of objects to subclusters. Similarly, we could sequentially tighten or relax constraints to obtain tradeoff curves along two or more dimensions e.g., $\bar{D}$ and σ_D. We might also increase N, which would have the effect of reducing $\bar{T}$, $\bar{D}$ and σ_D at the cost of an increased number of keys to manage.

5.6 Summary

In this case study, we have shown how our model and optimization techniques we have proposed can be used to design data distribution strategies that achieve a desired balance of privacy, preservation and performance. As demonstrated by the results in Table 5, some systems may provide strong guarantees in one dimension, but much weaker guarantees in another – we must consider all three aspects together when selecting a strategy.

The exact numerical results in our case study are not important. Rather, our techniques are significant because they facilitate the design process, and also because they allow us to compare different system designs quantitatively, along multiple dimensions.

6 Discussion

6.1 Extensions

There are a number of straightforward extensions that can be made to our model. We can introduce storage capacity constraints for each server in $\mathcal{X}$. This would translate, in our optimization, to J extra constraints in problem (8) that limit the total size of the data objects allocated to subsystem $\mathcal{S}_j$.

We can also extend (8) to specify a *redundancy factor*, K, for each data object, by simply replacing the constraint $\sum_j z_{i,j} = 1$ with $\sum_j z_{i,j} = K$ instead. This change would mean that we require each object in $\mathcal{O}$ to be replicated in exactly K subsystems. Adding a redundancy factor is conceptually similar to adding Copy operators below the Partition at the root i.e., $\bar{D}$ may increase and $\bar{L}$ may decrease as a result. Moreover, we can use (8) to solve the *object reallocation* problem. That is, if any one subsystem $\mathcal{S}_j$ were to become unavailable (and we were using a

redundancy factor of 2 or greater), we may want reallocate the data objects from the failed subsystem to a new one. To do so optimally, we simply hold fixed the $z_{i,j}$ values for all objects not previously allocated to the failed subsystem, and re-solve (8) for the $z_{i,j}$ values corresponding to those objects we are trying to reallocate.

6.2 Algorithm DLT

We conjecture that the suboptimality of Algorithm DLT is mostly due to Steps 1 and 2, since Steps 3-5 involve solving optimization problems. Thus, it is a good idea to consider a variety of designs in Steps 1 and 2, as was done in the case study. In Step 3, it may help to use a different $q_0^{(j)}$ for each subsystem $\mathcal{S}_j$, so that we get some subsystems that have better privacy, and others with better preservation. We can generate tradeoff curves between $p_{\mathbf{r}}^{(j)}$ and $q_{\mathbf{r}}^{(j)}$ and select "sweet spots" on each curve.

6.3 Operator Strength

We stated earlier that if Partition operators are disallowed, the expected damage $\bar{D}$ will not depend on the chosen operator implementation. However, if a Split or Threshold implementation is "weak", then $\bar{D}$ may actually increase as a result. For example, if a Split were implemented using DES encryption (which is known to be weak), then a leak of the ciphertext alone can compromise privacy. Thus, a designer should always check that the implementations chosen are sufficiently strong.

7 Conclusion

In this paper, we have studied how to manage the tradeoffs between the 3 P's of distributed data management – privacy, preservation and performance. We presented a descriptive model that encodes the decisions a designer can make regarding the 3 P's, along with metrics for measuring the 3 P's for a given system. We believe that it is important to consider all three attributes jointly, because some designs that are attractive in one or two dimensions may be much weaker in another.

We also presented a technique for finding good configurations under our metrics. Our technique allows us to optimize and quantitatively compare various designs. By simply tweaking some parameters, we can explore an entire multi-dimensional tradeoff space. Depending on where we want to be in the privacy-preservation-performance tradeoff space, we can select a final design.

References

[1] 104th US Congress. Health Insurance Privacy and Portability Act, 2002. http://www.hhs.gov/ocr/hipaa/.

[2] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *Proceedings of 2nd Biennial Conference on Innovative Data Systems Research*, Asilomar, USA, January 2005.

[3] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *Proceedings of the 28th International Conference on Very Large Databases*, Hong Kong, PRC, August 2002.

[4] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, pages 313–317, Arlington, VA, USA, June 1979.

[5] E. Goh, H. Shacham, N. Modadugu, and D. Boneh. Sirius: Securing remote untrusted storage, 2003.

[6] B. Mungamuru and H. Garcia-Molina. Beyond just data privacy. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research*, Pacific Grove, CA, USA, January 2007.

[7] B. Mungamuru, H. Garcia-Molina, and S. Mitra. How to safeguard your sensitive data. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, Leeds, UK, October 2006.

[8] B. Mungamuru, H. Garcia-Molina, and C. Olston. Configurations: Understanding alternatives for safeguarding data. *Stanford InfoLab Technical Report*, October 2005.

[9] V. Reich and D. S. H. Rosenthal. LOCKSS: Lots of copies keeps stuff safe. In *Proceedings of the 2000 Preservation Conference*, York, UK, December 2000.

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

[11] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti. POTSHARDS: Secure long-term storage without encryption. In *Proceedings of the 2007 USENIX Technical Conference*, Santa Clara, CA, USA, June 2007.

[12] A. Subbiah and D. M. Blough. An approach for fault tolerant and secure data storage in collaborative work environments. In *Proceedings of the ACM Workshop on Storage Security and Survivability*, Fairfax, VA, USA, November 2005.

[13] V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Record*, 33(1):50–57, 2004.

[14] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the Seventh Symposium on Operating Systems Design and Implementation*, Seattle, WA, USA, November 2006.

[15] J. Wylie, M. Bigrigg, J. Strunk, G. Ganger, H. Kiliccote, and P. Khosla. Survivable information storage systems. *IEEE Computer*, 33(8):61–68, Aug. 2000.