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 distributed data management systems. However, these objectives tend to compete with one another. This paper presents a model for describing distributed data management systems, along with a framework for measuring the privacy, preservation and performance offered by such systems. The framework enables a system designer to quantitatively explore and optimize the tradeoffs between the 3 P's.

1 Introduction

There are several attributes to consider when designing a distributed data management system. For example, we would like a system that enforces "privacy" by not divulging data to unauthorized entities. At the same time, we want to "preserve" the data effectively i.e., protect it from hardware failures, natural disasters, and so on. And, of course, we do not want to sacrifice "performance" – a system that runs slowly may not be very useful. There are many such desirable attributes: confidentiality, integrity, reliability, availability, throughput, and so on.

While each of these attributes has been studied extensively, there is not much work that considers the *tradeoffs* between them. We believe that today the real challenge in designing a system is in achieving a balance between several conflicting attributes. For example, a system that simply deletes all input data would be very "secure" – no data will ever leak out! – but would be unattractive in other dimensions. Similarly, one can build a highly "reliable" system that proliferates many copies of its data. This system would excel along the preservation dimension, but each extra copy would increase the chances of unauthorized break-ins. If we encrypt a large collection of records as a single "blob", 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 to design systems that strike a balance among them. We refer to these attributes as the "3 P's" of distributed data management – *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).

1.1 Overview

The main contributions of this paper are:

1. A unified framework for measuring the privacy, preservation and performance offered by a system.
2. An algorithm for finding systems that achieve a desired balance between the 3 P's.

Our work represents a bridge between *system design* and *risk management*. Risk management is the science of identifying, measuring and mitigating the uncertainty related to threats.

We identify privacy violations, data loss and poor performance as threats faced by a system in its daily operation. We then measure the potential harm caused by these threats: a) Privacy is measured by the *damage*, D, caused by leakage of information to unauthorized parties, b) Preservation is measured by the *loss*, L, incurred due to lost or corrupted data, and c) Performance is measured by the *running time*, T, of read and write commands issued to the system.

We mitigate risk not by providing "guarantees" – instead, we accept that bad things can happen, and try to minimize the harm when bad things do happen. How can we ensure that we do well "on average", and how can we reduce the likelihood of "catastrophes"? We treat D, L and T as random variables, and solve optimization problems involving their means and variances.

To effectively manage risk, it is crucial to account for the *heterogeneity* within a collection of data objects. For example, there may be a great deal of damage done if an internal e-mail discussing corporate strategy is leaked to outsiders. But, nobody may care if the e-mail was accidentally deleted. On the other hand, nobody would mind if marketing materials were "leaked" to outsiders (it's probably a good thing!). But if these marketing materials were lost, the time and money spent developing them would be lost. Moreover, e-mail is mostly text whereas marketing flyers are filled with large images, so the performance implications are vastly different between the two. In our framework, we *explicitly* model data objects in a heterogeneous collection as having differing values, sensitivities, sizes and usage patterns.

The outcome of our work is a solution strategy for the following problem: *Given a set of resources (e.g., data centers, servers, storage devices), and a collection of data objects (e.g., documents, photos, MP3s, 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., [2, 4, 8, 12]). 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 [13] to provide fault tolerance guarantees in a distributed system. In [10] and [11], the properties of threshold schemes are achieved by alternative, more efficient means. In [5] and [6], it is shown how to optimally tradeoff between privacy and preservation. However, these approaches are incomplete because:

- The performance dimension is not considered, and
- All data is treated equally – heterogeneity is not modeled.

Adding these two elements makes the problem substantially more complex, and our solution substantially more useful.

2 Model

In this section, we define 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. Distributed data management systems are then built up as *compositions* of these operators. In Sections 3 and 4, we will discuss how to quantify 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 $\mathbf{C}$, produces as its output k identical copies of its input data. A *k-way Split operator*, denoted by $\mathbf{S}$, applies an *encoding* to its input and produces k outputs, or *shares*, such that all k shares are required to fully reconstruct the input.

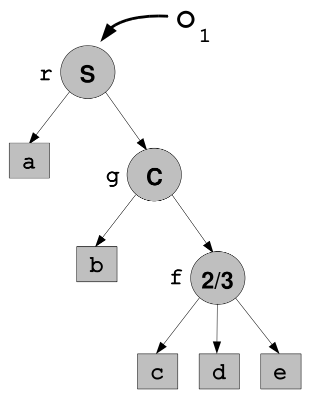

Figure 1. $F_\Theta = \mathbf{a}(\mathbf{b} + f_{2/3}(\mathbf{c}, \mathbf{d}, \mathbf{e}))$

Copy and Split are special cases of a more general class of operators, known as Threshold operators. A *k-of-n Threshold operator*, denoted $\mathbf{k}/\mathbf{n}$, 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 $\mathbf{r}$. The input[1] to $\mathbf{r}$ is then split into two shares, $\mathbf{a}$ and $\mathbf{g}$, using a 2-way Split operator so that both $\mathbf{a}$ and $\mathbf{g}$ are needed to reconstruct $\mathbf{r}$. Two identical copies of $\mathbf{g}$ are then made, labelled $\mathbf{b}$ and $\mathbf{f}$, using the 2-way Copy operator at $\mathbf{g}$. Finally, $\mathbf{f}$ is divided again into three shares $\mathbf{c}$, $\mathbf{d}$ and $\mathbf{e}$ such that any two are sufficient to reconstruct $\mathbf{f}$.

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

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

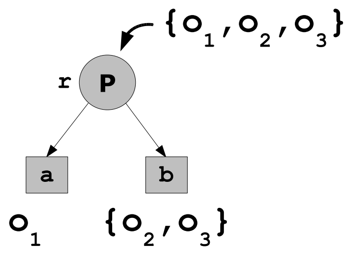

Figure 2. Partition operator.

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 $\mathbf{o}_1$.

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 (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 input object). 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 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.

2.4 Partition Operator

A *k-way Partition operator*, denoted $\mathbf{P}$, simply breaks its input into k disjoint *subcollections*, and outputs these k subcollections without any further encoding. Figure 2 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 2, for example, vertex $\mathbf{b}$ is valuable even without $\mathbf{a}$, since the subcollection $\{\mathbf{o}_2, \mathbf{o}_3\}$ is stored there. If $\mathbf{r}$ 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 [1]. The objects, in this case, would be individual records.

Similarly, there are several implementations of Threshold operators (e.g., [3, 9]). Threshold operators in practice tend to suffer from poor performance, so [11] proposes a faster alternative that composes replication and XOR operations. For a Partition operator, choosing an implementation means deciding to which output to send each object in the input collection (see Figure 2 for an example).

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 $\mathbf{r}$, with the ciphertext at $\mathbf{a}$ and encryption key at $\mathbf{g}$. The $\mathbf{k}/\mathbf{n}$ at $\mathbf{f}$ could use Shamir's scheme. Thus, if $\mathbf{o}_1$ was of size s, the data at $\mathbf{a}$ would also be size s, whereas $\mathbf{b}, \mathbf{c}$, $\mathbf{d}$ and $\mathbf{e}$ would be 128 bits each. Instead, if XOR were used at $\mathbf{r}$, the outcome would be quite different – $\mathbf{a}, \mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$, $\mathbf{e}$ and the input to $\mathbf{f}$ would all be of 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 $\mathbf{r}$.

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 $\mathbf{P}$ behaves like $\mathbf{S}$ 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 example, $F_\Theta = \mathbf{a}(\mathbf{b} + f_{2/3}(\mathbf{c}, \mathbf{d}, \mathbf{e}))$ is not the same as $F_\Theta = \mathbf{ab} + \mathbf{acd} + \mathbf{ade} + \mathbf{ace}$, though they are *logically equivalent*[2].

Our goal 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. In Section 5, we present a detailed case study illustrating an application of our techniques.

3.1 Privacy and Preservation

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 $\mathbf{b}$ in Figure 1 was a DVD kept in somebody's desk, a "break-in" might mean that the DVD is physically stolen. Alternatively, if $\mathbf{b}$ 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 permanent (e.g., media failures, bit rot). Returning to Figure 1, if server $\mathbf{b}$ was a DVD, "data loss" might mean that the DVD is damaged or misplaced. We can think of $q_\mathbf{v}$ as the "probability of preservation failure" at $\mathbf{v}$.

Together, the probabilities of break-in and data loss are referred to as *failure probabilities*. 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 (causing 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 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 $\mathbf{o}_i$[3]. 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 to us if an unauthorized party gained access to $\mathbf{o}_i$? 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, $\{A_i : \Omega_p \rightarrow \{0,1\}, i \in 1, \ldots, |\mathcal{O}|\}$, 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*, $D : \Omega_p \rightarrow [0, d_0]$ 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 $\mathbf{o}_i$ is lost. Intuitively, l_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 $l_2 \gg l_1$. Alternatively, l_i might refer to the time needed to restore $\mathbf{o}_i$ from a backup.

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 $\{B_i : \Omega_q \rightarrow \{0,1\}, i \in 1, \ldots, |\mathcal{O}|\}$, 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*, $L : \Omega_q \rightarrow [0, l_0]$ 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

[2]See [7] for a full discussion of logical equivalence.

[3]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.

$\mathbb{P}$, whereas $\bar{L}$ is computed under the data loss measure $\mathbb{Q}$. Intuitively, $\bar{D}$ and $\bar{L}$ are the "costs"[4] we will 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 – if Partition operators are allowed, then a configuration that minimizes $\bar{D}$ and $\bar{L}$ does not necessarily minimize $p_{\mathbf{r}}$ and $q_{\mathbf{r}}$, or vice versa. We will use this observation in Section 4 when we search for "good" configurations.

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[5]. 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 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 3, 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 show how to compute $\bar{T}$ – we begin at the terminals, and proceed "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 $T^R_{\mathbf{a}}(s)$ be the time needed to read an object of size s from vertex $\mathbf{a}$. Then $T^R_{\mathbf{a}}(s) = st_{\mathbf{a}}$, 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 $s_{\text{key}} = 128$ bits of data from server $\mathbf{a}$. Assume that $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$ are nodes from which we can transfer at 10 MB per second. Thus, $t_{\mathbf{a}} = \frac{1}{10}$, so $T^R_{\mathbf{a}}(s_{\text{key}}) = \frac{1}{10}\cdot\frac{1}{64} = 1.56$ milliseconds. Similarly, $T^R_{\mathbf{b}}(s) = T^R_{\mathbf{c}}(s) = T^R_{\mathbf{d}}(s) = \frac{s}{10}$ seconds.

[4]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.

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

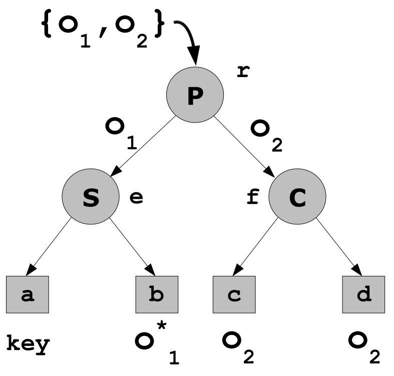

Figure 3. $F_\Theta = (\mathbf{ab})(\mathbf{c}+\mathbf{d})$

Let $T^R_{\mathbf{e}}(s)$ be the time needed to reconstruct an object of size s from the non-terminal vertex $\mathbf{e}$. $T^R_{\mathbf{e}}(s)$ 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, $t_{\mathbf{e}} = t_{\text{AES}}$, where t_{AES} is the processing time for AES. $T^R_{\mathbf{e}}(s)$ 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 $T^R_{\mathbf{e}}(s) = st_{\mathbf{e}} + \max\{T^R_{\mathbf{a}}(s_{\text{key}}), T^R_{\mathbf{b}}(s)\}$. In a *serial* model, we can only read one of $\mathbf{a}$ and $\mathbf{b}$ at a time, so $T^R_{\mathbf{e}}(s) = st_{\mathbf{e}} + T^R_{\mathbf{a}}(s_{\text{key}}) + T^R_{\mathbf{b}}(s)$. 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 $t_{\mathbf{e}} = t_{\text{AES}} = \frac{1}{20}$, then $T^R_{\mathbf{e}}(s_1) = \frac{1}{20}\cdot 1 + \max\{\frac{1}{10}, \frac{1}{640}\} = 150$ 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. 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 $\bar{T} = f_1\lambda_1 T^R_{\mathbf{e}}(s_1) + f_2\lambda_2 T^R_{\mathbf{f}}(s_2) + (1-$

$\lambda_1)f_1T^W_{\mathbf{e}}(s_1) + (1-\lambda_2)f_2T^W_{\mathbf{f}}(s_2)$. Using $f_1 = f_2 = 0.5$ and $\lambda_1 = \lambda_2 = 0.8$, we get $\bar{T} = 140$ milliseconds per command.

4 Optimization

Now that we have 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 ($\mathbf{S}$ operators) tend to worsen preservation, and vice versa ($\mathbf{C}$). Moreover, using $\mathbf{P}$ operators to improve performance can adversely impact both privacy and preservation, whereas $\mathbf{k/n}$ can do the exact opposite. Therefore, the correct approach is to solve constrained optimization problems such as (1):

$$\min_{\mathcal{S}} \bar{T} \text{ subject to } \bar{D} \leq D_0,\ \bar{L} \leq L_0 \qquad (1)$$

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 can 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 [6] – 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:

$$\min_{\Theta} p_{\mathbf{r}} \text{ subject to } q_{\mathbf{r}} \leq q_0 \qquad (2)$$

Since there was only a single data object, and performance was not being modeled, it was sufficient to find the best configuration Θ[6]. 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 4. The subsystems are comprised exclusively of $\mathbf{S}$, $\mathbf{C}$ and $\mathbf{k/n}$ operators (i.e., no $\mathbf{P}$), and they are mutually disjoint in that they have no 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 $\bar{T}$ 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 P3*, for finding[7] approximate solutions to (1):

[6]In [6], the symbols $P(\Theta)$ and $Q(\Theta)$ are used instead of $p_{\mathbf{r}}$ and $q_{\mathbf{r}}$.

[7]The name "P3" alludes to an optimization over the 3 P's.

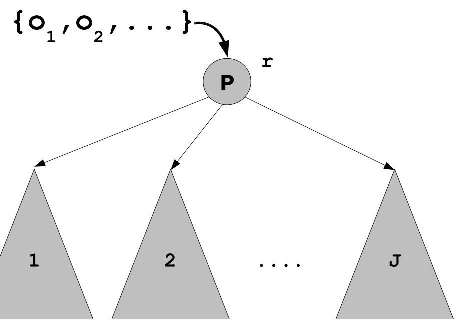

Figure 4. Partition at r, with J subsystems.

Algorithm 1 (Algorithm P3)

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

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 P3 a few times, and keep the best result.

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 [6], an efficient algorithm, which we refer to as *Algorithm P2*, is given[8] for computing approximate solutions to (2). Algorithm P2 finds a configuration comprised of $\mathbf{C}$, $\mathbf{S}$ and $\mathbf{k/n}$ 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 $\mathbf{P}$ 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 separate 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.

[8]The name "P2" alludes to an optimization over privacy and preservation.

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.

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 4.

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:

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

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

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

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

$$\begin{aligned} \min_{\{z_{i,j}\}} \quad & \sum_{i,j} h_{i,j} z_{i,j} \\ \text{subject to} \quad & \sum_{i,j} p^{(j)}_{\mathbf{r}} d_i z_{i,j} \le D_0, \;\; \sum_{i,j} q^{(j)}_{\mathbf{r}} l_i z_{i,j} \le L_0 \\ & \sum_j z_{i,j} = 1 \; \forall \; i, \;\; z_{i,j} \in \{0,1\} \; \forall \; (i,j) \end{aligned} \tag{6}$$

Problem (6) 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 just integers:

$$0 \le z_{i,j} \le 1 \; \forall \; i,j \tag{7}$$

Problem (6) with the *relaxed* constraints (7) is a *linear program*, and can be solved easily and efficiently.

At the optimum, the solution $\{z^*_{i,j}\}$ obtained under the relaxed constraints (7) is a good approximation to the solution of the integer program (6). 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 (6). The constraint relaxation in (7) 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 efficient solution algorithms exist as well. The case study in Section 5 considers one such instance.

To summarize, using the solution $\{z^*_{i,j}\}$ obtained from the constraint relaxation (7) is a viable, efficient technique for solving problems such as (6). 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 the 3 P's.

5 Case Study

We present a case study to illustrate how the techniques in this paper (i.e., Algorithm P3) 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 in the USA[9], 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[10] (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

[9]Similar laws exist in the EU (95/46/EC) and Japan (HPB 517).

[10]There is a ceiling on the total fine, which we ignore for our case study.

Table 1. Candidate system designs.

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

unreliable and may go offline. Our service-level agreement with the healthcare facility 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 access a record from our system.

5.1 Steps 1 and 2

In Steps 1 and 2 of Algorithm P3, 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 1. The first four correspond to four separate executions of Algorithm P3, whereas the final two systems in Table 1, *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. Each proposed design has its strengths and weaknesses – in particular we will observe a tradeoff between privacy ($\bar{D}$ and σ_D) and performance ($\bar{T}$), for a fixed level of preservation, $\bar{L}$.

5.2 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 2.

Using Algorithm P2, we solve (2) for the sub-clusters of various sizes listed in Table 1. 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 P2, 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 P2 are listed in Table 3. As an example, System *2x(6+4)* in Table 1 will have two sub-clusters of size $m_j = 6$ and two of size $m_j = 4$.

Table 2. Parameters used in case study.

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. Sub-clusters output by Step 3.

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 $G = f_{5/6}(\mathbf{x_5}, \ldots, \mathbf{x_{10}}) + \mathbf{x_4}(f_{4/5}(\mathbf{x_6}, \ldots, \mathbf{x_{10}}) +$ $\mathbf{x_5}(f_{3/4}(\mathbf{x_7}, \ldots, \mathbf{x_{10}}) + \mathbf{x_6}(f_{2/3}(\mathbf{x_8}, \mathbf{x_9}, \mathbf{x_{10}}) +$ $\mathbf{x_7}(\mathbf{x_8} + \mathbf{x_9}))))$

Figure 5 is a plot of $p_{\mathbf{r}}$ and $q_{\mathbf{r}}$ for the configurations in Table 3, 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 6 we illustrate the configuration output use for a subcluster of size $m_j = 5$.

5.3 Step 4

The next step is to select implementations for the operators in each of the configurations listed in Table 3. 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 2. While the exact processing times

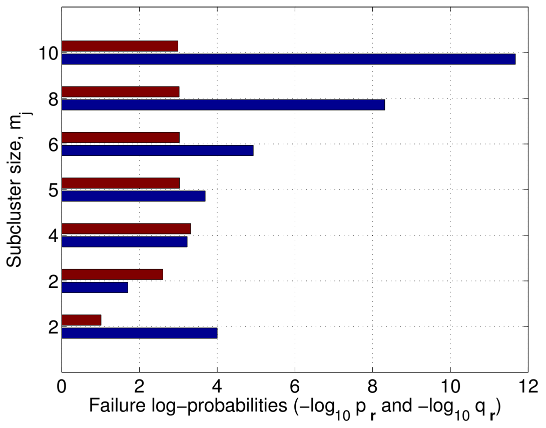

Figure 5. Log-probabilities of failure for configurations in Table 3. Longer bars correspond to lower probabilities of failure.

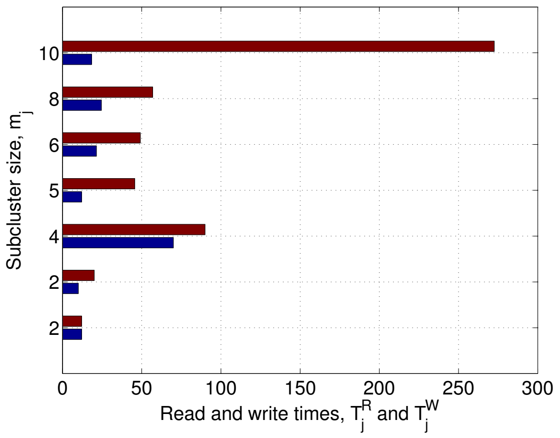

Figure 7. Read and write times in milliseconds, assuming $s = 100$ kB.

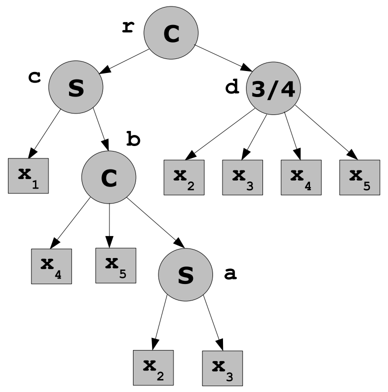

Figure 6. Subcluster with $m_j = 5$ servers.

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 3. As a concrete example, consider the configuration $F_\Theta = \mathbf{x_1}(\mathbf{x_4} + \mathbf{x_5} + \mathbf{x_2}\mathbf{x_3}) + f_{3/4}(\mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5})$, illustrated in Figure 6. 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}) = $ 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 2). Therefore, to write the data object to $\mathbf{a}$ would require $s(t_{\text{AES}} + t_{\mathbf{x}}) + s_{\text{key}} t_{\mathbf{x}}$ seconds. On the other hand, if $\mathbf{a}$ was XOR, a write would require $s(t_{\text{XOR}} + 2t_{\mathbf{x}})$ since XOR outputs two data objects of size equal to the input. Therefore, the time required to write an object of size s is $T_{\mathbf{a}}^W(s) = st_{\mathbf{x}} + \min\{st_{\text{AES}} + s_{\text{key}} t_{\mathbf{x}}, st_{\text{XOR}} + st_{\mathbf{x}}\}$. A similar calculation shows that, in this case, the read and write times for $\mathbf{a}$ are equal i.e., $T_{\mathbf{a}}^R(s) = T_{\mathbf{a}}^W(s) = \bar{T}_{\mathbf{a}}$. Therefore, we prefer $I(\mathbf{a}) = $ AES if $s(t_{\text{XOR}} - t_{\text{AES}}) + (s - s_{\text{key}})t_{\mathbf{x}} > 0$, and $I(\mathbf{a}) = $ XOR otherwise.

We use the functions $T_{\mathbf{a}}^R(s)$ and $T_{\mathbf{a}}^W(s)$ in selecting an implementation for $\mathbf{c}$. For example, if $I(\mathbf{c}) = $ AES and the key is sent to vertex $\mathbf{b}$, then $T_{\mathbf{c}}^W(s) = s(t_{\text{AES}} + t_{\mathbf{x}}) + 2s_{\text{key}} t_{\mathbf{x}} + T_{\mathbf{a}}^W(s_{\text{key}})$ and $T_{\mathbf{c}}^R(s) = s(t_{\text{AES}} + t_{\mathbf{x}}) + s_{\text{key}} t_{\mathbf{x}}$. On the other hand, if $I(\mathbf{c}) = $ XOR, then $T_{\mathbf{c}}^W(s) = s(t_{\text{XOR}} + t_{\mathbf{x}}) + 2st_{\mathbf{x}} + T_{\mathbf{a}}^W(s)$ and $T_{\mathbf{c}}^R(s) = s(t_{\text{XOR}} + 2t_{\mathbf{x}})$. 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 $\bar{T}_{\mathbf{c}} = \lambda T_{\mathbf{c}}^R(s_{\mathbf{c}}) + (1 - \lambda) T_{\mathbf{c}}^W(s_{\mathbf{c}})$ 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 the dominant factor.

In our case study, we assume the medical records are each roughly $s_{\text{rec}} = 100$ kB in size, which is much larger than s_{key}. We therefore choose $I(\mathbf{c}) = $ AES. This causes objects of size s_{key} to be sent to $\mathbf{a}$. Consequently, we choose $I(\mathbf{c}) = $ XOR.

In this manner, we select implementations and calculate the resulting execution times for each sub-cluster i.e., the amount of time to read and write a data object of size s. The results are plotted in Figure 7, for $s = s_{\text{rec}}$. The subclusters of various sizes have comparable read speeds, whereas write speeds vary widely, since multiple servers typically are updated in a write.

5.4 Step 5

The final step in Algorithm P3 is to combine the J subclusters using a single J-way Partition operator at the root, as described in Table 1. It only remains to decide how the implement the Partition i.e., how to distribute the medical records 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. 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.

Let n_i be the number of records in block $\mathbf{o}_i$. Due to the square-root law distribution of access frequencies, n_i is (roughly) linear in i. The size of block i is, in turn, linear in n_i i.e., $s_i = n_i s_{\text{rec}}$. 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, we get $f_i = \frac{1}{N}$. Since our notion of loss is downtime (i.e., the access frequency of unavailable data), we get $l_i = f_i = \frac{1}{N}$.

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

$$\min_{\mathcal{S}} \ \sigma_D \ \text{ subject to } \ \bar{L} \leq 0.1\% \qquad (8)$$

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

$$\min_{\mathcal{S}} \ \alpha\bar{D} + \beta\bar{T} \ \text{ subject to } \ \sigma_D \leq \sigma_0, \ \bar{L} \leq 0.1\% \qquad (9)$$

In this case study, we solve (8) for each design proposed in Table 1. The results of the optimization for each proposed design are given in Table 4. 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 – low running times due to their simplicity, but weak on privacy and preservation.

Arguably, System *2x(8+2)* is a good 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.

6 Conclusion

In this paper, we have presented an integrated framework for balancing the 3 P's of distributed data management: privacy, preservation and performance. It is important to consider the 3P's jointly. As demonstrated by our case study (see Table 4), some systems that are attractive in one or two dimensions may be much weaker in another.

Table 4. Summary of results.

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

Algorithm P3 is significant because it allows us to *quantitatively* compare various candidate designs. By simply varying some parameters, we can explore an entire multi-dimensional tradeoff space. Depending on where we want to be in the tradeoff space, we can select a final design.

References

[1] 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.

[2] 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.

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

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

[5] 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.

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

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.