Reliability Analysis of Deduplicated and Erasure-Coded Storage

Xiaozhou Li, Mark Lillibridge; Mustafal Uysal

HP Laboratories
HPL-2010-146

Keyword(s):

Deduplication, erasure coding

Abstract:

Space efficiency and data reliability are two primary concerns for modern storage systems. Chunk-based deduplication, which breaks up data objects into single-instance chunks that can be shared across objects, is an effective method for saving storage space. However, deduplication affects data reliability because an object's constituent chunks are often spread across a large number of disks, potentially decreasing the object's reliability. Therefore, an important problem in deduplicated storage is how to achieve space efficiency yet maintain each object's original reliability. In this paper, we present initial results on the reliability analysis of HP-KVS, a deduplicated key-value store that allows each object to specify its own reliability level and that uses software erasure coding for data reliability. The combination of deduplication and erasure coding gives rise to several interesting research problems. We show how to compare the reliability of erasure codes with different parameters and how to analyze the reliability of a big data object given its constituent parts' reliabilities. We also present a method for system designers to determine under what conditions deduplication will save space for erasure-coded data.

External Posting Date: October 21, 2010 [Fulltext] Approved for External Publication

Internal Posting Date: October 21, 2010 [Fulltext]

Published in HotMetrics 2010, New York City, NY, USA, June 18, 2010.

Reliability Analysis of Deduplicated and Erasure-Coded Storage

(HP Labs Technical Report HPL-2010-146)

Xiaozhou Li Mark Lillibridge
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA
firstname.lastname@hp.com

Mustafa Uysal*
VMware
3401 Hillview Avenue
Palo Alto, CA
muysal@vmware.com

ABSTRACT

Space efficiency and data reliability are two primary concerns for modern storage systems. Chunk-based deduplication, which breaks up data objects into single-instance chunks that can be shared across objects, is an effective method for saving storage space. However, deduplication affects data reliability because an object's constituent chunks are often spread across a large number of disks, potentially decreasing the object's reliability. Therefore, an important problem in deduplicated storage is how to achieve space efficiency yet maintain each object's original reliability. In this paper, we present initial results on the reliability analysis of HP-KVS, a deduplicated key-value store that allows each object to specify its own reliability level and that uses software erasure coding for data reliability. The combination of deduplication and erasure coding gives rise to several interesting research problems. We show how to compare the reliability of erasure codes with different parameters and how to analyze the reliability of a big data object given its constituent parts' reliabilities. We also present a method for system designers to determine under what conditions deduplication will save space for erasure-coded data.

1. INTRODUCTION

As the amount of important data that needs to be digitally stored continues to explode, space efficiency and data reliability are two primary concerns for modern storage systems. Several techniques can address these concerns. For example, chunk-based deduplication is a space-saving technique where identical copies of data fragments called *chunks* are identified and eliminated (e.g., [4, 6, 12]). Erasure codes (e.g., systematic Reed-Solomon codes [8]) provide data reliability by adding data redundancy. By adjusting their parameters, erasure codes can achieve different trade-offs between reliability and redundancy. For example, RAID [9] and mirroring are two special forms of erasure coding.

Chunk-based deduplication involves the following steps: 1) dividing an object into small chunks and computing a hash for each chunk; 2) for each chunk hash, determining whether it is already in the deduplicated store by looking it up in an index; 3) storing the new chunks in the data store; and 4) creating a manifest for each object that contains pointers to its chunks. Deduplication fundamentally changes the reliability of objects: since chunks are shared across multiple objects, the reliability of a given object is now determined by the reliability of its constituent chunks, including the ones shared with other objects. Furthermore, the chunks are potentially spread across a much larger number of disks as each object can share chunks that are already stored.

We consider several questions regarding the reliability of deduplicated data. First, how do we characterize the effects of deduplication on the reliability of objects in the storage system? Ideally, we want no degradation of reliability due to deduplication. Second, how do we adjust the redundancy in deduplicated storage so that object reliability is at least as good as in the non-deduplicated stores? Ideally, we should be able to construct reliable deduplicated stores based on the reliability requirements of the objects.

This paper makes two primary contributions. Firstly, we show that deduplication need not result in loss of reliability if the redundancy of the shared chunks is properly adjusted. Secondly, we also show how to calculate the minimum amount of redundancy to keep space overhead low and how to calculate the net storage space after deduplication and redundancy adjustment. We address these problems in the context of the HP Key-Value Store (HP-KVS), a deduplicated and erasure-coded key-value store.

This paper is organized as follows. We first give a brief overview of the HP-KVS system and outline how we carry out deduplication in HP-KVS in Section 2. We then give a mathematical formulation for the reliability in HP-KVS and derive the basic properties of this formulation, including how the reliability changes as the erasure code's parameters change in Section 3. We derive how the reliability of an object depends on those of its constituent chunks and how to adjust the erasure code parameters of chunk containers to make up for any potential loss of reliability during deduplication in Section 4. We present a method of calculating the net storage space after deduplication and erasure code parameter adjustment in Section 5. We discuss related work in Section 6 before concluding the paper in Section 7.

2. THE HP KEY-VALUE STORE

HP-KVS [1] is a key-value store tailored for large data objects such as pictures, audio files, VM images, and movies of moderate size ($\approx$ 100 MB to 100 GB). The high-level architecture of HP-KVS is illustrated in Figure 1. Clients use a Representational State Transfer (REST) interface to interact with a proxy server located in a data center, which performs get and put operations on behalf of the client. HP-KVS itself has two types of servers: key-lookup servers and fragment servers. Key lookup servers store a metadata list for each key, which holds the locations of its value's fragments. (Hereafter, to be consistent with key-value store terminology, we use the word *value* in place of data objects.)

HP-KVS exports two interfaces for clients: put(key, value, r-spec) and get(key). Put allows a client to associate a value with a key and get allows a client to retrieve a value associated with the

*This research was performed when the author was at HP Labs.

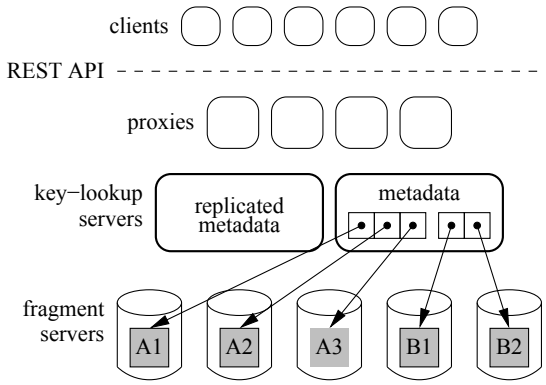

Figure 1: Architecture of HP-KVS.

```
foreach (value to be deduplicated)
    read the value's fragments;
    reconstruct the value;
    divide the value into chunks;
    foreach (chunk)
        if (chunk exists in some container)
            point to that container;
        else
            put chunk in current open container;
            if (open container full)
                close container;
                store container;
                open another container;
    store value manifest as key-lookup server metadata;
```

Figure 2: The high-level deduplication procedure.

given key. The redundancy specification (or *r-spec* for short) of a value specifies the desired level of reliability. More precisely, it specifies how to erasure encode that value. An r-spec takes the form of two integers (k, m), where k is the number of data fragments that the value should be broken into and m is the number of parity fragments that should be computed and stored. Altogether, $n = k + m$ fragments are stored such that any k fragments (data or parity) can recover the original value. For example, in Figure 1, the first value is stored with r-spec $(2, 1)$ where A1 and A2 are data fragments, and A3 is a parity fragment. The second value is stored with r-spec $(2, 0)$ where B1 and B2 are data fragments.

We deduplicate the values stored in HP-KVS as follows: We consider an offline deduplication process that runs periodically and deduplicates stored values in batches. In HP-KVS, normally a value's metadata contains a list of its erasure-coded fragments. After deduplication, this metadata is replaced with a list of pointers to the chunks making up the value. Chunks are stored in *chunk containers*. The reliability of containers may vary with different containers being encoded with different r-specs. Figure 2 outlines this procedure. We stress that chunks and fragments are different concepts. In particular, chunks (multiple KBs) are usually much smaller than fragments (multiple MBs).

3. RELIABILITY MODEL

In this section, we present the basic assumptions and definitions for the reliability analysis carried out in the rest of the paper. We first present the reliability model and define the reliability metric, and then present some useful properties of this metric. The current reliability model and metric favor simplicity and ease of understanding over precision. We plan to refine them in the future. We also include some additional analytical results in Appendix A that are useful for more refined analysis.

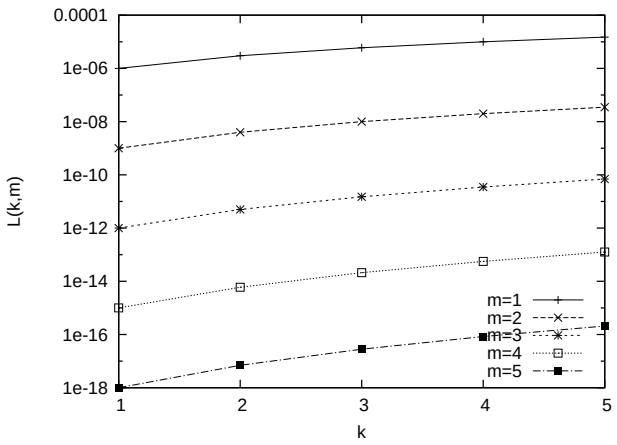

Figure 3: $L(k, m)$ vs. k for $q = 0.001$.

3.1 Reliability metric

We adopt the following simple interpretation of a (k, m) r-spec. Let $p(t)$ be the conditional probability that a disk remains functional throughout time interval $[0, t]$, given that it is functional at time 0, and let $q(t) = 1-p(t)$. Typically, $p(t)$ is close to 1 and thus $q(t)$ is close to 0 for reasonably small values of t. Let $n = k + m$ and let $R(k, m, t)$ be the conditional probability that a (k, m)-encoded value remains recoverable throughout $[0, t]$ given that all disks are functional at time 0, and let $L(k, m, t) = 1 - R(k, m, t)$. We assume that the failures of the disks in an r-spec are independent. This assumption makes reliability analysis easier, but we realize that it is a simplifying assumption. Still, in practice, the system can choose disks from failure domains that are likely to be independent of each other (e.g., disks in different servers, different racks, or even different data centers). Since n is typically not a large number (see Section 4.2), this assumption is reasonable. Since t remains the same throughout the entire analysis, we will omit explicitly writing down t hereafter. Under these assumptions, we have:

$$\begin{aligned} L(k,m) &= \sum_{i=m+1}^{n} \Pr[\text{exactly } i \text{ disks failed}] \\ &= \sum_{i=m+1}^{n} \binom{n}{i} p^{n-i} q^i. \end{aligned} \tag{1}$$

We can obtain some basic understanding of $L(k, m)$ by numerical calculation. We have plotted $L(k, m)$ for some typical values of k, m, and q in Figures 3 and 4. Figure 3 shows that as k increases, $L(k, m)$ increases, only modestly. Figure 4 shows that as m increases, $L(k, m)$ decreases, much more quickly. Careful examination of Figure 3 shows that if we fix n then $L(k, m)$ increases quickly as k increases (and m decreases): compare $L(2, 4)$, $L(3, 3)$, and $L(4, 2)$. The change of $L(k, m)$ with respect to k or m is roughly exponential (note the log-scale on the y-axis).

3.2 Basic properties of $L(k, m)$

Since the full definition of $L(k, m)$ (Equation 1) is somewhat difficult to work with, we will primarily work with an approximation of $L(k, m)$, which favors ease of understanding over preci-

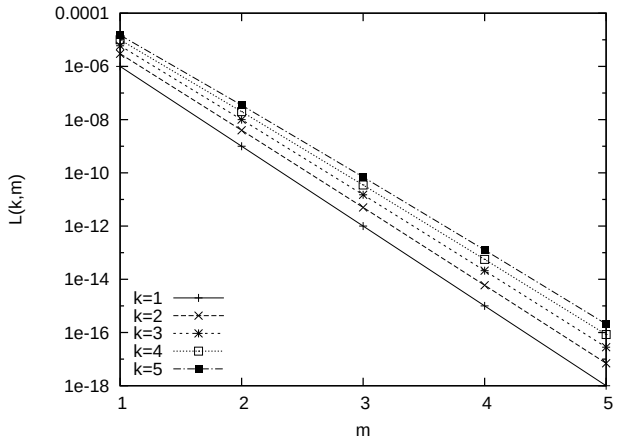

Figure 4: $L(k, m)$ **vs.** m **for** $q = 0.001$**.**

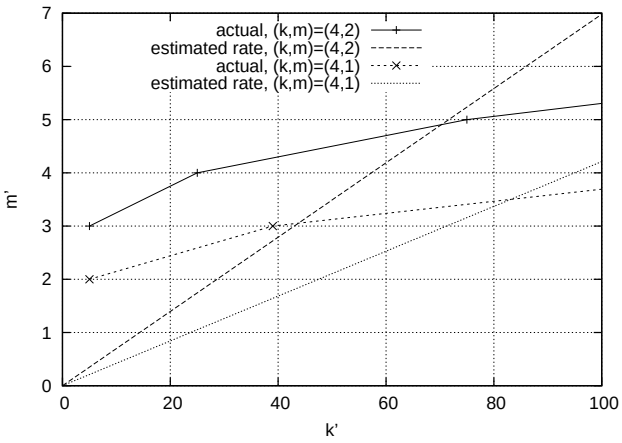

Figure 5: m' **vs.** k' **for** $q = 0.001$**.**

sion. In particular, we will ignore all but the first term of the definition of $L(k, m)$ because subsequent terms decrease by roughly a factor of q, which is close to 0. To be more precise, we observe that the ratio between two subsequent terms in $L(k, m)$ is $\frac{q}{p} \cdot \frac{n-i}{i+1} \leq \frac{q}{p} \cdot \frac{n-(m+1)}{(m+1)+1} = \frac{q}{p} \cdot \frac{k-1}{m+2}$. Since k and m are typically small integers (see Section 4.2), the above expression is largely determined by q. Therefore, an approximation of $L(k, m)$ is

$$L(k, m) \approx \binom{n}{m+1} p^{k-1} q^{m+1}. \tag{2}$$

How fast does $L(k, m)$ decrease as we increase m, and how fast does it increase as we increase k? Figures 3 and 4 have shown some numerical values. To understand the trends analytically, we use Equation 2 and obtain

$$\frac{L(k, m+1)}{L(k, m)} \approx \frac{n}{m} \cdot q, \qquad \frac{L(k+1, m)}{L(k, m)} \approx \frac{n}{k} \cdot p. \tag{3}$$

Therefore, increasing m by one decreases $L(k, m)$ signif cantly because q is close to 0. In other words, with a small increase in space redundancy, we can obtain a substantial increase in reliability. This is very important because we wish to provide good reliability without much space redundancy. On the other hand, increasing k by one only increases $L(k, m)$ modestly.

Some systems (e.g., HYDRAstor [4]) require a f xed value of n; for such systems, it is necessary to consider trading-off m for k:

$$\frac{L(k+1, m-1)}{L(k, m)} \approx \frac{m}{k} \cdot \frac{p}{q}. \tag{4}$$

Here we see that increasing k by one and decreasing m by one increases the L value signif cantly because of the $1/q$ factor. This result implies that, for reliability considerations alone, we should set $k = 1$. However, the shortcoming of doing so is that the space eff ciency of the resulting code, $r = k/n$, commonly known as the *rate* of the code, is very low.

What is the largest δ such that $L(k + \delta, m + 1) \leq L(k, m)$? The answer to this question tells us, for each additional parity fragment, how many more data fragments we can accommodate, while achieving at least the original reliability. The hope is that, the rate of the new code is better than that of the old code, i.e., $\frac{k+\delta}{n+\delta+1} > \frac{k}{n}$. We carry out the calculation as follows. Let $r = k/n$ (i.e., the rate) and $\bar{r} = m/n$ (i.e., the redundancy). Then using Equation 3, we want $\frac{L(k+\delta,m+1)}{L(k,m)} = \frac{L(k+\delta,m+1)}{L(k,m+1)} \cdot \frac{L(k,m+1)}{L(k,m)} \approx \left(\frac{p}{r}\right)^{\delta} \cdot \left(\frac{q}{\bar{r}}\right) \leq 1$. Approximating $p \approx 1$, we have

$$\delta \leq \log_{1/r}(\bar{r}/q). \tag{5}$$

To make sense of this expression, consider an example where $n = 6$, $k = 4$, $m = 2$, and $q = 0.001$. Then we have $\delta \leq \log_{1.5} 333.3 \approx 14$. In other words, a $(4 + 14, 2 + 1) = (18, 3)$ code has the same L value as a $(4, 2)$ code, with the 14 being as large as possible. We remark that this is actually a conservative estimate.

Figure 5 shows how m' grows with k'. Each data point (k', m') on an "actual" plot means that $L(k', m') \geq L(k, m)$, where m' is the smallest possible for a given k', using numerical calculations. The two "estimated" lines are drawn with slope $1/\delta$, where δ is calculated using Equation 5. As we can see, our estimated growth rates are conservative (as they grow faster than the actual plots) but still reasonably accurate.

4. THE R-SPEC OF A CONTAINER

The key question we address in this paper is how to determine what the r-spec of the chunk containers should be, as a function of the original values' r-specs so that after deduplication each value is at least as reliable as it was before. In the absence of deduplication, each value is erasure coded using some user-specif ed (k, m) r-spec, and then stored across $n = k + m$ disks. Note that n is typically much smaller than the total number of disks in the data store. When a value is deduplicated, only the unique chunks in that value are stored. Duplicate chunks are eliminated and replaced with pointers to the unique copies. This has a profound effect on the reliability of individual values: the data of a given value can potentially be spread across many drives because its chunks may be spread across many chunk containers. In other words, the reliability of individual values is potentially tied to the reliability of a large part of the entire data store due to the sharing of chunks.

We now address the f rst main question in this paper: What is a proper r-spec for a container? By proper we mean an r-spec that (1) provides at least the original reliability for all the values that have chunks in this container, and (2) the space overhead of the container is minimal. We carry out our analysis in two steps. First we analyze a special case, and then we develop the analysis for the general case.

4.1 Special case

We consider the following special case as a f rst step. Suppose all values in the HP-KVS have the same (k, m) r-spec. Let s be the

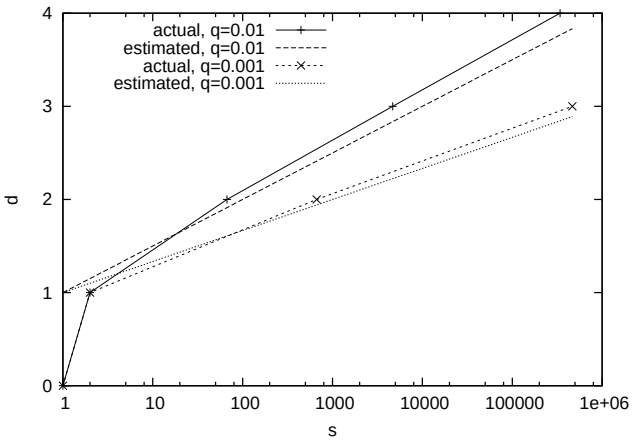

Figure 6: d vs s for $(k, m) = (4, 4)$.

maximum number of chunks that any value is composed of after deduplication. Suppose further each container is to be stored with a $(k, m+d)$ r-spec (i.e., same number of data fragments as the value pre-deduplication). How big should d be? We derive an estimate as follows. After deduplication, we have for any value v:

$$\begin{aligned}\Pr[v \text{ lost}] &= \Pr[\text{any of its chunks lost}] \\ &\leq s \cdot \Pr[\text{a particular chunk lost}] \\ &= s \cdot L(k, m+d).\end{aligned}$$

The above inequality is due to the union bound, which holds for any set of events, regardless of whether those events are mutually exclusive or independent of each other. In other words, we do not have to be concerned whether different containers are stored at overlapping set of disks. To ensure no loss of reliability for v, it suffices to choose a d such that $s \cdot L(k, m+d) \leq L(k, m)$. Using our approximation of $L(k, m)$ (Equation 2), we need

$$s \cdot \binom{n+d}{m+d+1} \cdot p^{k-1} q^{m+d+1} \leq \binom{n}{m+1} \cdot p^{k-1} q^{m+1}.$$

Since $\binom{n+d}{m+d+1} > \binom{n}{m+1}$, we need $s \cdot q^d \leq 1$, meaning that we need $d \geq \log_{1/q} s$. A more careful analysis, which we omit due to space limitations, shows that it suffices for

$$d = \log_{1/q} s + \epsilon, \tag{6}$$

where ϵ is a small constant such as 1. In essence, ϵ is a function that depends on k, m, n, but since these parameters are typically small integers, ϵ is typically at most 1. Figure 6 plots the smallest d such that $s \cdot L(4, 4+d) \leq L(4, 4)$, based on both numerical calculations and our analysis. As we can see, our analysis is fairly accurate.

4.2 General case

We now turn to the more general case: What if containers are encoded with a (k', m')-code, yet values are encoded with a (k, m)-code, but $k' \neq k$? How should we properly set k', m' in this case? To answer this question, we first make a few remarks on the impact of k and m on the reliability, space efficiency, and access speed of encoded data. The previous sections have explained how the reliability changes with different k and m. The space efficiency of a (k, m) r-spec is simply the rate of the code, $k/(k+m)$ or k/n. The influence on access speed is more subtle. For reads, since the k data fragments can be read in parallel, increasing k tends to increase read speed, but only to a certain extent because of the overhead of reading ever smaller fragments. For writes, increasing m increases the encoding overhead and number of I/O operations. Therefore, it is undesirable to have needlessly high values of m, n.

Therefore, we assume that there is a fixed system-wide value n' (cf. HYDRAstor [4]), and we need to determine k' and m' such that (1) $n' = k' + m'$, (2) the original values' reliability is matched or exceeded, and (3) containers are space efficient (i.e., maximize k'/n'). Let $k' = k + \Delta k$, $m' = m + \Delta m + d$, where d is as given in Equation 6. How do we ensure that the container's reliability is high enough so that all the values that have chunks in this container are at least as reliable as before? In Equation 5, we have shown that each additional parity fragment can accommodate δ additional data fragments. As a rough extension to that equation, Δm additional parity fragments can accommodate $\delta \cdot \Delta m$ additional data fragments. Therefore, if we require Δk and Δm satisfy $\Delta k \leq \delta \cdot \Delta m$, then for any value v, we have

$$\begin{aligned}\Pr[v \text{ lost}] &\leq s \cdot L(k + \Delta k, m + \Delta m + d) \\ &\leq L(k + \Delta k, m + \Delta m) \\ &\leq L(k, m).\end{aligned}$$

In other words, we want to maximize Δk such that $\Delta k + \Delta m + k + m + d = n'$ and $\Delta k \leq \delta \cdot \Delta m$. This task is straightforward: start with $\Delta m = 1$ and see if there is any Δk that can satisfy the two constraints; if yes, we are done; if not, increase Δm by 1 and repeat. For example, if $\epsilon = 1$, $q = 0.001$, $s = 1000$, $k = 2$, $m = 2$, $n = k + m = 4$, and $n' = 12$, then $\delta = 9$ and $d = 2$. Therefore, we have $\Delta k = 5$ and $\Delta m = 1$, namely, the containers should be encoded with a $(2 + 5, 2 + 1 + 2) = (7, 5)$ code.

We remark that the above analysis can be used to address related questions in other contexts. For example, a common question is the following: If one stores a big data object into s RAID-6 9-disk (7+2) groups, then how big can s be until the overall value's reliability is below RAID-5 9-disk (8+1) reliability? By the analysis in Section 4.2, we want to ensure $s \cdot L(7, 2) \leq L(8, 1)$. Assuming $q = 0.001$ and using Equation 4, we have $s \leq L(8, 1)/L(7, 2) \approx (2p)/(7q) \approx 300$.

5. THE NET STORAGE SPACE

In the previous section, we have seen that, in order to maintain the original reliability guarantees, we may have to add some additional parity fragments for the containers, which introduces space overhead. Therefore, a natural question is this: After the whole process of deduplication and adjusting the erasure code parameters of the containers (with additional parity fragments), do we manage to save any storage space at all? A positive answer to this question will justify the whole process, otherwise the process is not worth it. This section presents an analysis of the net storage space.

To answer this question, many parameters have to be taken into account, most of which have been discussed in the previous section. However, one parameter that has not been discussed is the *compaction factor* of deduplication, that is, the expected factor between the space used by the original data and that by the deduplicated data. Clearly, whether the entire process ultimately saves space or not depends on the compaction factor, a parameter that is highly workload dependent. For example, for backup workloads, this factor can be as high as 50–70, with 10–20 typical, but for archival workloads, this factor is often around 1.5–3. By some knowledge of the nature of the data set, the system designer often has a reasonable estimate of this factor. Using the compaction factor and other parameters, we derive equations that tell us under what conditions the net storage space will be less than the original.

5.1 Single r-spec

Let D be the original (i.e., non-deduplicated) data set size, D' be the deduplicated data set size, and $c = D/D'$ be the compaction factor. By definition, $c \geq 1$ (we ignore deduplication overheads such as extra metadata). Let S be the original storage space, S' be the net (i.e., deduplicated and then re-encoded) storage space. We assume that all values have the same r-spec (k, m), and each value is deduplicated into at most s unique chunks. It is easy to see that $S = D \cdot \frac{m+k}{k} = D \cdot \frac{n}{k}$. We want to investigate under what circumstances we can expect $S' \leq S$. We observe that $S' = \frac{D}{c} \cdot \frac{n'}{k'}$. Therefore, we need $\frac{D}{c} \cdot \frac{n'}{k'} \leq D \cdot \frac{n}{k}$, or equivalently, $c \geq \frac{k}{n} \cdot \frac{n'}{k'} = \frac{k}{n} \cdot \frac{n'}{k+\Delta k}$. In Section 4.2, we have seen how to calculate Δk. Consider the following example. Suppose $\epsilon = 1$, $q = 0.001$, $s = 1000$, $k = 2$, $m = 2$, $n = k + m = 4$, and $n' = 9$. Then by Section 4.2, we have $\Delta k = 2$. Therefore, we need $c \geq \frac{2}{4} \cdot \frac{9}{2+2} = \frac{9}{8} = 1.125$. In other words, only if we have a data set that is likely to have a compaction factor at least 1.125 can we expect the net storage space to be less than the original.

5.2 Multiple r-specs

HP-KVS allows for multiple r-specs, which will be useful for applications in which data of different importance should have different levels of reliability. Most applications are likely to only require a few different r-specs. In this section, we consider a simple yet typical case: two r-specs, (k_1, m_1) and (k_2, m_2), the first for normal data and the second for important data. We assume that the rate of the first one is larger than that of the second one, namely, $r_1 = \frac{k_1}{k_1+m_1} = \frac{k_1}{n_1} > r_2 = \frac{k_2}{k_2+m_2} = \frac{k_2}{n_2}$. Suppose the data set sizes are D_1 and D_2. The total non-deduplicated storage space is therefore $S = D_1 \cdot \frac{n_1}{k_1} + D_2 \cdot \frac{n_2}{k_2}$.

Suppose the respective compaction factors of these two data sets are c_1 and c_2. After deduplication, the deduplicated data can be divided into three parts: those present in D_1 only (part I), those present in D_2 only (part II), and those present in both D_1 and D_2 (part III). Let part I's size be $\alpha_1 \cdot \frac{D_1}{c_1}$, part II's size be $\alpha_2 \cdot \frac{D_2}{c_2}$, and part III's size be $(1 - \alpha_1) \cdot \frac{D_1}{c_1} = (1 - \alpha_2) \cdot \frac{D_2}{c_2}$, where $0 \leq \alpha_1, \alpha_2 \leq 1$.

To protect deduplicated data with multiple r-specs, we can use containers that are encoded with the same new r-spec. However, doing so may sacrifice space efficiency, because the r-spec of the container has to have sufficient redundancy for the important data, yet for normal data, such an r-spec is an overkill. Therefore, we allow containers to have two different r-specs. Furthermore, for the sake of simplicity, we assume that the containers use the same k_1 and k_2 as their data fragment numbers (i.e., the special case described in Section 4.1). We note that the calculation below can be easily extended to the general case described in Section 4.2. By this assumption, part I of the deduplicated data set will be re-encoded with r-spec $(k_1, m_1 + d)$. Part II will be re-encoded with $(k_2, m_2 + d)$, where $d = \log_{1/q} s + \epsilon$ and where s is the maximum number of chunks for any value. Part III will be re-encoded with the higher reliability r-spec $(k_2, m_2 + d)$. Then the total storage space after deduplication and re-encoding is $S' = \alpha_1 \cdot \frac{D_1}{c_1} \cdot \frac{n_1+d}{k_1} + \frac{D_2}{c_2} \cdot \frac{n_2+d}{k_2}$ (note the absence of α_2 in this expression). We want to save space, i.e., we want $S' \leq S$. Therefore, we need $D_1 \cdot \left(\frac{\alpha_1}{c_1} \cdot \frac{n_1+d}{k_1} - \frac{n_1}{k_1}\right) \leq D_2 \cdot \left(\frac{n_2}{k_2} - \frac{1}{c_2} \cdot \frac{n_2+d}{k_2}\right)$. In other words, only if the above inequality holds can we save space. By plugging in the values of the various parameters, the system designer can get an estimate whether and how much space can be saved. Consider the following example: $n_1 = 4$, $k_1 = 3$, $n_2 = 4$, $k_2 = 1$, $d = 2$, $\alpha_1 = 0.8$, $D_1 = 5D_2$. Then we have $\frac{4}{c_1} + \frac{3}{c_2} \leq \frac{16}{3}$. Suppose we have $c_1 = 1.5$ and $c_2 = 1.25$ then the above expression is satisfied because $\frac{8}{3} + \frac{12}{5} \leq \frac{16}{3}$. However, in practice, some parameters such as α_1 can be difficult to estimate.

6. RELATED WORK

To our knowledge, Bhagwat et al. [3] are the first to address reliability concerns in deduplicated storage. They observe that deduplication alters the reliability of stored data because of the sharing of common chunks. Using a metric they call "robustness"—the expected amount of data lost due to device failures (i.e., severity)—and targeting replication-based storage, they argue that the number of copies of a chunk should be logarithmic to the popularity of that chunk. In contrast, our metric is the likelihood of data loss, not severity, and our findings indicate that the additional parity fragments grow logarithmically with the maximum number of chunks into which any value can be broken.

HYDRAstor [4] is a deduplicated secondary storage system that allows chunks to be placed in different resilience classes, each of which has a different level of reliability. However, the choice of which resilience class to use for each chunk is left to the user and has no relationship to the degree of sharing of the chunks. We believe that our analysis can help HYDRAstor automatically decide the proper resilience class for each chunk based on the importance of the *documents* containing that chunk. No reliability metric was used in that paper or proposed, nor is it clear how the reliability of a chunk affects that of any given document.

Liu et al. [7] suggest that variable-length chunks should be preferred over fixed-size chunks because the former has proved to yield more space savings. The variable-length chunks are first packed into bigger fixed-size objects (i.e., chunk containers), which are then erasure coded and placed on multiple storage nodes. However, all chunks are considered equally important and they are all erasure coded using the same erasure code with the same redundancy configuration. In short, the overall approach is "deduplicate then RAID." As we have shown, this means that the minimum reliability of a document under this scheme grows worse as the document size increases (i.e., has more chunks) and as more data is stored overall (i.e., more storage nodes are required).

Our reliability analysis for per-value r-specs is largely along the lines of that by Thomasian and Blaum [11]. Our current analysis is combinatorial, and we plan to extend it to Markov analysis as well.

Disk failures in the field have been studied by Schroeder and Gibson [10]. Their study will be useful for determining proper values of q in the empirical validation of our analysis.

7. CONCLUDING REMARKS

In this paper, we have addressed a few reliability analysis problems that arise from the deduplication of a erasure-coded key-value store. In particular, we showed how to adjust the erasure code parameters to make up for the potential loss of reliability after deduplication, and we showed how to determine whether space can be saved by deduplicating erasure-coded data. Although these analysis problems arise in this particular context, we believe that they are of independent interest and may shed light on other related problems. We have made simplifying assumptions on our reliability model. In future work, we plan to refine our analysis to include non-independent disk failures and latent sector errors [2].

We conclude by noting that the reliability metric used in this paper has only considered the *likelihood* of data loss, but not the *severity* of it, yet deduplication has impact on both. Existing reliability metrics may be inadequate to address both and a new metric may be needed. To see this, consider a simple example. Suppose

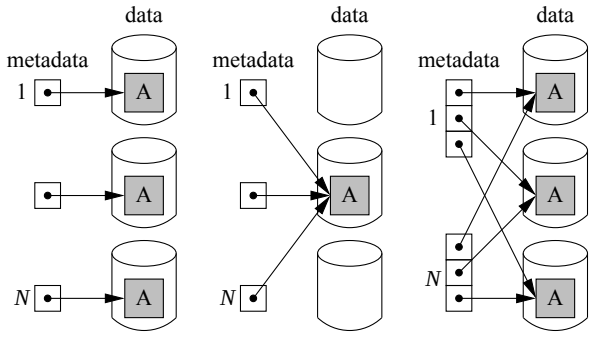

Figure 7: Comparing N copies with one copy.

metric	original	deduped	note
PDL	$1-p^N$	$1-p$	deduped wins
expected blocks lost	Nq	Nq	tie
MTTDL	$\frac{1}{N\lambda}$	$\frac{1}{\lambda}$	deduped wins
average data loss rate	$N\lambda$	$N\lambda$	tie

Figure 8: Comparing the original and the deduped.

we have N data blocks, and the blocks all have the same content but are stored on different disks (Figure 7(a)). We run deduplication on these N blocks, resulting in all metadata pointing to the same block (Figure 7(b)). Which layout is more reliable? Assume that each disk fails with probability q and has a failure rate of λ. Figure 8 summarizes the reliability for this example using several common metrics. In all cases, the deduped case either wins or ties with the original in spite of the fact that the chance of total data loss is far greater in the deduped case. This simple example illustrates that deduplication increases the *severity* of data loss but decreases the *likelihood* of it. Existing metrics either only measure the likelihood (e.g., probability of data loss or PDL, mean time to data loss or MTTDL) or use a simple combination of the two (e.g., expected amount of data loss, average data loss rate). Figure 7(c) suggests that one can increase the reliability at the cost of storing more metadata by pointing to multiple copies of the same data. The search for a meaningful reliability metric is an ongoing research topic, and we refer the interested readers to recent proposals such as the one by Greenan et al. [5].

8. REFERENCES

[1] E. Anderson et al. Eff cient eventual consistency in Pahoehoe, an erasure-coded key-blob archive. In *Proceedings of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010)*, pages 181–190, June 2010.

[2] L. N. Bairavasundaram et al. An analysis of latent sector errors in disk drives. In *Proceedings of the 2007 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 289–300, June 2007.

[3] D. Bhagwat et al. Providing high reliability in a minimum redundancy archival storage system. In *Proceedings of the 14th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '06)*, pages 413–421, September 2006.

[4] C. Dubnicki et al. HYDRAstor: A scalable secondary storage. In *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST)*, pages 197–210, February 2009.

[5] K. Greenan, J. S. Plank, and J. J. Wylie. Mean time to meaningless: MTTDL, markov models, and storage system reliability. In *Proceedings of the Second Workshop on Hot Topics in Storage and File Systems*, June 2010.

[6] M. Lillibridge et al. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST)*, pages 111–123, February 2009.

[7] C. Liu et al. R-ADMAD: High reliability provision for large-scale de-duplication archival storage systems. In *Proceedings of the 23rd international conference on Supercomputing*, pages 370–379, June 2009.

[8] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North Holland, Amsterdam, 1978.

[9] D. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 109–116, June 1988.

[10] B. Schroeder and G. A. Gibson. Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you? *ACM Transactions on Storage*, 3(3):Article 8, October 2007.

[11] A. Thomasian and M. Blaum. Mirrored disk organization reliability analysis. *IEEE Transactions on Computers*, 55(12):1640–1644, December 2006.

[12] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the Data Domain deduplication f le system. In *Proceedings of the Seventh USENIX Conference on File and Storage Technologies (FAST)*, pages 269–282, February 2008.

APPENDIX

A. ADDITIONAL PROPERTIES OF L

We mention some additional mathematical properties of $L(k,m)$ that may be of independent interest. The following property of $L(k,m)$ is straightforward: For all k, m, $L(k,0) = 1-p^k$ and $L(0,m) = 0$. The next property of $L(k,m)$ is somewhat surprising: For all k and m, $L(k+1,m+1) = p\cdot L(k,m+1)+q\cdot L(k+1,m)$. The proof of this property uses the well-known property of binomial coeff cients: $\binom{i+1}{j} = \binom{i}{j} + \binom{i}{j-1}$. The next property of $L(k,m)$ is intuitive: For all k and m, $L(k,m) < L(k+1,m)$ and $L(k,m) > L(k,m+1)$. This property basically states that increasing data fragments decreases the reliability, and increasing parity fragments increases the reliability. One can prove this property using the previous property combined with an inductive argument.

If one wishes to obtain a more precise estimate of $L(k,m)$ than Equation 2, one way to go about it is to take the f rst two or three terms (rather than one) in the def nition of $L(k,m)$ and ignore the rest. Another way is to treat $L(k,m)$ as a power series where the ratio between two subsequent terms is $\frac{n-i}{i+1}\cdot\frac{q}{p} \le \frac{k-1}{m+2}\cdot\frac{q}{p}$, which is less than 1 for most values of k and m. Denote this ratio by α. Observing $1+\alpha+\alpha^2+\cdots = \frac{1}{1-\alpha}$, we can upper bound $L(k,m)$ by $L(k,m) \le \frac{1}{1-\alpha}\binom{n}{m+1}p^{k-1}q^{m+1}$.