Reliability Analysis of Deduplicated and Erasure-Coded Storage

Xiaozhou Li Mark Lillibridge Mustafa Uysal
Hewlett-Packard Laboratories
firstname.lastname@hp.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 saving 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 we show how to analyze the reliability of a big data object given its constituent parts' reliabilities. We also outline several research challenges in designing large scale reliable deduplication systems.

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, 5, 11]). Erasure codes (e.g., systematic Reed-Solomon codes [7]) 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 [8] and data replication are two special forms of erasure coding.

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 individual objects.

In this paper, we present a reliability analysis of deduplicated data. We show that deduplication need not result in loss of reliability if the redundancy of the shared chunks is properly adjusted. We also show how to calculate the minimum amount of redundancy to keep space overhead low. We address these problems in the context of the HP Key-Value Store (HP-KVS), a deduplicated and erasure-coded key-value store.

We first outline how we carry out deduplication in HP-KVS (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 parameters of the erasure code change, and how the reliability of an object depends on those of its constituent chunks (Section 3). We then outline a number of open research problems (Section 4) and discuss related work (Section 5) before concluding (Section 6).

2. THE HP KEY-VALUE STORE

We first give a brief summary of the HP-KVS system [1] (originally called Pahoehoe), highlighting the features that are most relevant to our discussion. We then present our deduplication architecture inside the HP-KVS.

HP-KVS is a key-value store tailored for large 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 maps it to the locations of its value's fragments.

HP-KVS exports two interfaces for clients: `put(key, value, r-spec)` and `get(key)`. Put allows a client to associate a value (or object) with a key and get allows a client to retrieve an object version associated with the specified key. The redundancy specifi-

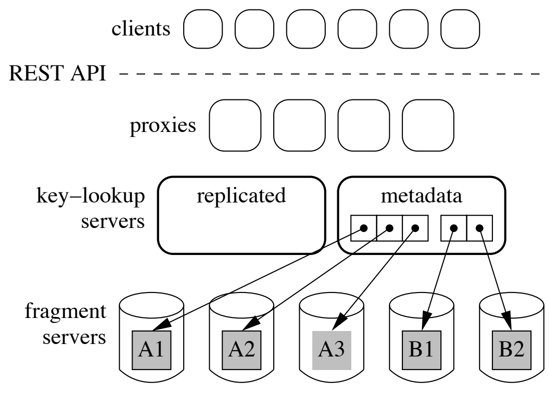

Figure 1: Architecture of HP-KVS.

```
foreach (value to be deduplicated)
    read the value's fragments;
    reconstruct the value;
    run a chunking algorithm;
    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;
```

Figure 2: The high-level deduplication procedure.

cation (or *r-spec* for short) contains the desired level of reliability for the object being stored; more precisely, it specifies how to erasure encode the object and 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. 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 both data fragments.

We deduplicate the objects 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 an object'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 object. 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.

3. RELIABILITY ANALYSIS

The key question we address in this paper is how to determine the r-spec of the chunk containers as a function of the original objects' r-specs so that after deduplication each object is at least as reliable as it was before. In the absence of deduplication, each object is erasure coded using some user-specified (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 an

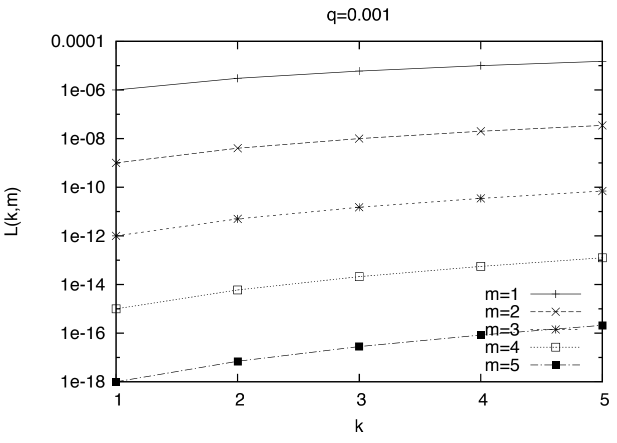

Figure 3: Growth of $L(k, m)$ versus k.

object is deduplicated, only the unique chunks in that object are stored. Non-unique chunks are eliminated and replaced with pointers to the unique copies. This has a profound effect on the reliability of individual objects: the data of a given object can potentially be spread across many drives because its chunks may be spread across many chunk containers. In other words, the reliability of individual objects are potentially tied to the reliability of a large part of the entire data store due to the sharing of chunks.

The primary contribution of this paper is a method for determining a proper r-spec for a container. To achieve this, we first present our reliability model (Section 3.1), then derive the basic properties of this model (Section 3.2), and finally use these properties to obtain our main result (Section 3.3). We also mention some additional analytical results that may be of independent interest (Section 3.4).

3.1 Reliability model

We adopt the following simple interpretation of a (k, m) r-spec. Let $p(t)$ be the probability that a disk remains functional after time t and let $q(t) = 1 - p(t)$ be the probability that a disk fails before time t. Typically, $p(t)$ is close to 1 and thus $q(t)$ is close to 0 for reasonably small values of t. We call a value a (k, m)-value if the value is erasure coded with k data fragments and m parity fragments and all fragments are stored on different disks. Let $n = k + m$ and let $L(k, m, t)$ be the probability that a (k, m)-value cannot be recovered after time t. We assume that all disk failures are independent. Since t remains the same throughout the entire analysis, we will omit explicitly writing down t. Under these assumptions, we have:

$$L(k, m) \quad = \quad \sum_{i=m+1}^{n} \binom{n}{i} p^{n-i} q^{i}. \tag{1}$$

Some understanding of the behavior of $L(k, m)$ can be gained 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, but only modestly. Figure 4 shows that as m increases, $L(k, m)$ decreases, but 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). Different values of q result in similar observations, but with different slopes.

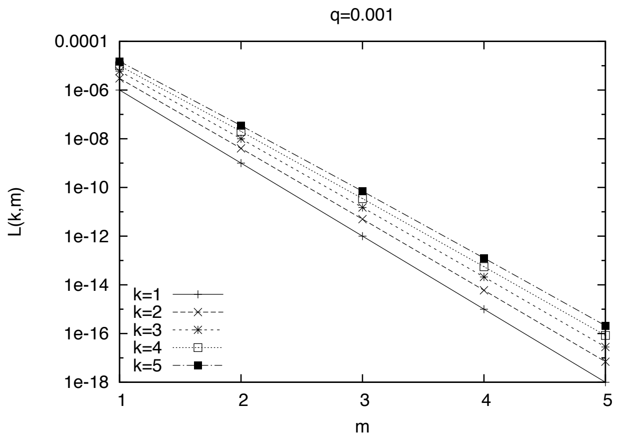

Figure 4: Growth of $L(k, m)$ versus m.

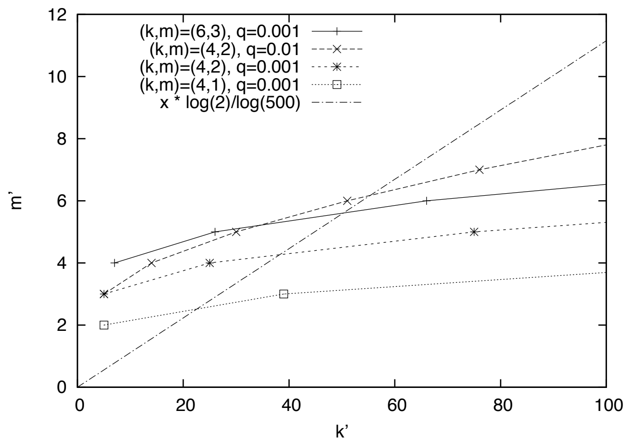

Figure 5: Growth of m' versus k'.

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 $L(k, m)$. 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, the ratio between two subsequent terms in $L(k, m)$ is $(q/p) \cdot ((n - i)/(i + 1))$. If n, and thus i, is not big, then the above expression is largely determined by q. Therefore, a reasonable 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 our approximation (Equation 2) and obtain

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

That is, increasing m by one decreases $L(k, m)$ greatly because q is close to 0. Thus, with a small increase in space redundancy, we can obtain a large increase in reliability. This is good news 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 fixed 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 significantly 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 efficiency 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 $L(k + \delta, m + 1) \leq L(k, m)$? We can approximate this 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 $(p/r)^{\delta} \cdot (q/\bar{r}) \leq 1$. 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 $n = 4, k = 2, m = 2$, and $q = 0.001$. Then we have $\delta \leq \log_2 500 \approx 9$. In other words, a $(2 + 9, 2 + 1) = (11, 3)$ code has the same L value as a $(2, 2)$ code, with the 9 as large as possible. We remark that this is actually a very *conservative* estimate.

Figure 5 shows how $L(k', m')$ can match $L(k, m)$ for various k, m, and q using numerical calculations. From this figure we observe several trends. First, the larger q, the faster the growth of m': compare the plots for $(k, m) = (4, 2)$, and $q = 0.001$ and 0.01. Second, if we fix k, then m' grows faster for a bigger m: compare the plots for $(k, m) = (4, 1)$ and $(4, 2)$ with $q = 0.001$. Three, for fixed k/m (i.e., the same space efficiency), the growth rates of m' are about the same: compare the plots for $(k, m) = (4, 2)$ and $(6, 3)$, and $q = 0.001$. Lastly, the function $x \cdot \log_{500} 2$ clearly grows faster than the plot for $(k, m) = (4, 2)$ and $q = 0.001$, indicating that our estimate for the growth of m' is indeed conservative. (Note that since we are plotting m' as a function of k', we have inversed δ as $\log_{500} 2 = 1/\log_2 500$.)

3.3 R-spec of a container

We now address the main question in this paper: What is a proper r-spec for a container? We consider the following special case as a first step. Suppose all objects in the KVS have the same (k, m) r-spec, and after deduplication, each object consists of at most s chunks. 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:

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

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 set $\Pr[\text{value } v \text{ lost}] \leq s \cdot L(k, m + d) \leq L(k, m)$. Using our approximation of $L(k, m)$ (Equation 2), we have

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

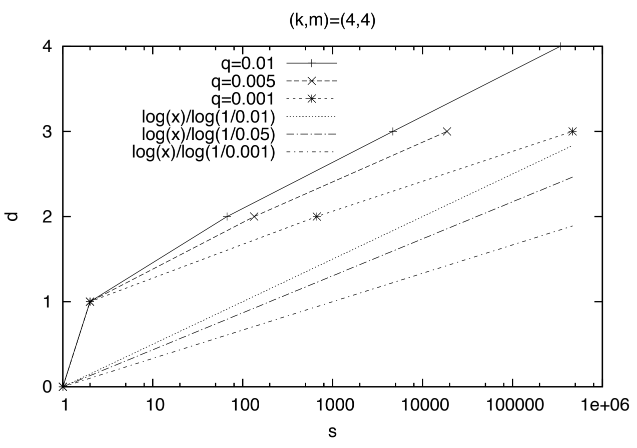

Figure 6: Growth of d versus s.

Therefore, $s \cdot q^d \leq 1$, meaning $d \geq \log_{1/q} s$. A more careful analysis, which we omit due to space limitations, shows that

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

where ϵ is a small constant such as 1. Figure 6 plots the smallest d such that $s \cdot L(k, m+d) \leq L(k, m)$ for $(k, m) = (4, 4)$, based on both numerical calculations and our analysis. As we can see, the growth rate of $\log_{1/q} s$ is a remarkably accurate prediction. (In this figure, e.g., $\log(x)/\log(1/0.01) = \log_{100} x$.)

We now turn to the more general case: What if the container can have a k' that is different from an object's k? To address this case, we first make a few remarks on the impact of k and m on the reliability, space efficiency, and access speed of the code. 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, $k/(k + m) = k/n$. 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 a high value of n or a needlessly high value of m.

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' reliabilities are matched or exceeded, and (3) containers are space efficient (i.e., maximize k'/n'). To do so, we let $k' = k + \Delta k$, $m' = m + \Delta m + d$, where d is as given in Equation 6, and Δk and Δm satisfy $\Delta k \leq \delta \cdot \Delta m$, where δ is given as in Equation 5. Then we have

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

In other words, we need to choose Δk and Δm such that (1) $\Delta k + \Delta m + k + m + d = n'$, (2) $\Delta k \leq \delta \cdot \Delta m$, and (3) Δk is maximized. This task is straightforward. For example, if $\epsilon = 1$, $q = 0.001$, $s = 1000$, $k = 2$, $m = 2$, $n = 4$, and $n' = 12$, then $\delta = 9$ and $d = 2$. Therefore, we have $\Delta k = 5$ and $\Delta m = 1$. In other words, the containers should be encoded with a $(2+5, 2+1+2) = (7, 5)$ code.

(Remark: 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 object's reliability is below RAID-5 9-disk (8+1) reliability? By the analysis in Section 3.3, we want to ensure $s \cdot L(7, 2) \leq L(8, 1)$. Assuming $q = 0.001$ and using Equation 4, we have $s \leq L(8, 1)/L(7, 2) \approx (2p)/(7q) \approx 300$.)

3.4 Additional properties of $L(k, m)$

In this section, we present some additional properties of $L(k, m)$ that may be of independent interest. The following property of $L(k, m)$ is straightforward: For all k and m, $L(k, 0) = 1 - p^k$, $L(0, m) = 0$. 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 coefficients: $\binom{i+1}{j} = \binom{i}{j} + \binom{i}{j-1}$. 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. Our current proof uses 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 is to take the first two or three terms (rather than one) in the definition 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 $((n-i)/(i+1)) \cdot (q/p) \leq ((k-1)/(m+2)) \cdot (q/p)$, because $i \geq m+1$, which is less than 1 for most values of k and m. Denote this by α and $L(k, m)$ is upper bounded by its first term times $1 + \alpha + \alpha^2 + \cdots = 1/(1 - \alpha)$.

4. ONGOING WORK

More refined reliability analysis. In this paper, we have used $L(k, m)$ as a simple combinatorial interpretation of the (k, m) r-spec. We plan to consider other interpretations such as MTTDL. However, additional considerations such as disk repairs and latent sector errors [2] may make the analysis for such models very complicated, if not intractable.

Space-reliability tradeoff. Under what circumstances is this entire "dedupe and re-encode" procedure worthwhile? For example, if the deduplication factor is expected to be small, then it might not make sense. Can we determine the best tradeoff dynamically?

Multiple r-specs. We expect that for most use cases of HP-KVS, the users would require several, but not many, different r-specs. Multiple r-specs brings up several new technical challenges. For example, what r-spec should a container have if it contains chunks that are pointed to by values with different r-specs? Is it a good idea to do cross-r-spec deduplication?

Severity of data loss. Another way to assess the impact of deduplication on reliability is to consider not just the *likelihood* of data loss, but also the *severity* of it. However, we argue that existing reliability metrics are inadequate to assess this impact, thereby motivating the need for a new metric. Suppose 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 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., PDL, 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

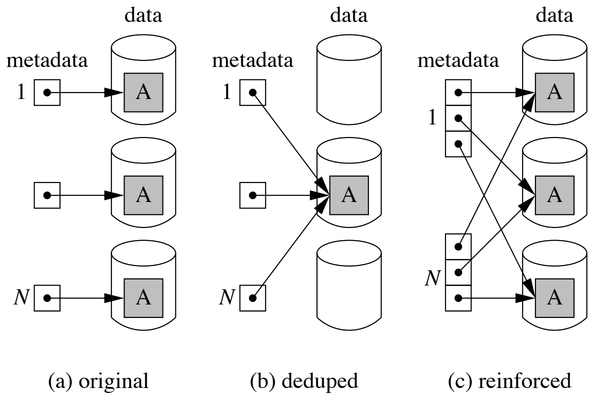

Figure 7: Comparing N copies with one copy.

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

Figure 8: Comparing the original and the deduped.

can increase the reliability at the cost of storing more metadata by pointing to multiple copies of the same data.

5. 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 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 analysis targets erasure-coded storage and our metric is per-value r-spec.

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 this paper or proposed, nor is it clear how the reliability of a chunk affects that of any given document.

Liu et al. [6] 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 "dedupe then RAID." As we have shown, this means that the minimum reliability of a document under this scheme grows worse as the document lengthens (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 [10]. 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 [9]. This study will be useful for determining proper values of q in the empirical validation of our analysis.

6. CONCLUDING REMARKS

In this paper, we have outlined a few reliability analysis problems that arise from the deduplication of a erasure-coded key-value store. Although the analysis problems arise in this particular context, we believe that they are of independent interest and may shed light on other related problems.

7. REFERENCES

[1] E. Anderson et al. Efficient 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)*, June 2010. To appear.

[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] 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.

[6] 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.

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

[8] 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.

[9] 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.

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

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