Dynamically Quantifying and Improving the Reliability of Distributed Storage Systems

Rekha Bachwani[†], Leszek Gryz[‡], Ricardo Bianchini[†], and Cezary Dubnicki[‡]
[†]Rutgers University [‡]NEC Labs America
{rbachwan,ricardob}@cs.rutgers.edu {gryz,dubnicki}@nec-labs.com

Abstract

In this paper, we argue that the reliability of large-scale storage systems can be significantly improved by using better reliability metrics and more efficient policies for recovering from hardware failures. Specifically, we make three main contributions. First, we introduce NDS (Normalcy Deviation Score), a new metric for dynamically quantifying the reliability status of a storage system. Second, we propose MinI (Minimum Intersection), a novel recovery scheduling policy that improves reliability by efficiently reconstructing data after a hardware failure. MinI uses NDS to tradeoff reliability and performance in making its scheduling decisions. Third, we evaluate NDS and MinI for three common data-allocation schemes and a number of different parameters. Our evaluation focuses on a distributed storage system based on erasure codes. We find that MinI improves reliability significantly, as compared to conventional policies.

1 Introduction

High reliability is critical for storage systems. However, achieving it at a reasonable cost can be a challenge, especially in large-scale systems built from commodity devices. Even when the devices are fairly reliable, failures are frequent in such systems due to the large number of devices they employ. Redundancy is used to improve reliability, either by replicating the data blocks, as in RAID-1 [15] or replica-based distributed systems [16], or by storing additional information, as in RAID-5 or erasure-coded distributed systems [8, 13]. Unless the amount of redundancy in the system is extremely large, when a device fails in a large-scale system, the data stored on it has to be immediately reconstructed on other devices, since device repair or replacement may take a long time and new failures can occur in the interim.

In this paper, we seek to improve the reliability of large-scale storage systems through more sophisticated failure-management policies and better reliability metrics, rather than increased redundancy (and its associated storage cost).

Several previous papers have studied the reliability of large-scale storage systems, e.g. [3, 8, 12, 13, 14, 17, 18, 21]. In these papers, reliability is often quantified using metrics that estimate the Probability of Data Loss (PDL) and/or the Mean Time To Data Loss (MTTDL). PDL is estimated either as the percentage of simulation runs that result in data loss or by using a (typically combinatorial) model of the PDL for the system. Similarly, MTTDL is estimated either as the mean of the time-to-data-loss values over a large number of simulations or by using a (typically Markovian) model of the system reliability.

Regardless of how they are computed, PDL and MTTDL quantify reliability with a single, static measure, irrespective of time or the current state of the system. Although very useful, these metrics provide only a macroscopic, long-term view of system reliability; they provide no help in assessing reliability at each point in time, as device failures, data reconstructions, and device replacements occur.

To address this limitation, we propose the Normalcy Deviation Score (NDS), a new metric for dynamically quantifying the reliability status of a storage system. Specifically, NDS computes a number or score that represents how far the system is from normal (i.e., failure-free) operation by focusing on the amount of data redundancy that has been lost, in a flexible and tunable fashion. The reliability under normal operation can be estimated using standard metrics, such as PDL or MTTDL.

NDS has many benefits. Because NDS can be efficiently computed, the system can use it to quantify reliability dynamically, after each device failure, data reconstruction, or device replacement event. Doing so provides a clear picture of the reliability behavior over time. Perhaps more importantly, with the ability to quantify and predict the reliability impact of different events, the system can attempt to improve reliability in a dynamic and informed manner.

We study this latter benefit in this paper. In particular, we focus on the problem of reconstructing the data from failed nodes and disks as quickly as possible to avoid long periods of reduced redundancy in a distributed storage system. This problem is becoming ever more important, as disk capacities and overall storage system sizes continue to increase at a rapid pace. Few papers have addressed this issue [7, 11, 14, 21]. [7, 11] considered the scheduling of reconstructions but only at the level of entire storage units in data centers, rather than individual nodes and disks in a distributed storage system. [14, 21] did not consider the scheduling of concurrent reconstructions. Other works have simply assumed that the recovery bandwidth (i.e., the bandwidth available for reconstructing data from failed devices on other devices and, when devices are replaced or repaired, copying the data back to their original locations) is shared concurrently by all reconstructions (and copy backs). Furthermore, none of the previous papers leveraged the ability to quantify reliability dynamically.

Given these gaps in the literature, we propose a new policy for scheduling data reconstructions in distributed storage systems. The policy, called Minimum Intersection (MinI), determines when each reconstruction should be performed and which disks should participate in it. Because of redundancy, multiple disks can potentially participate as data sources in each reconstruction. For higher performance, MinI tries to use a different target disk for each reconstruction. To make its decisions, MinI leverages the NDS metric to tradeoff reliability and performance. For example, in one of its configurations, MinI increases the disk bandwidth dedicated to reconstructions up to a pre-defined limit, if this increase would generate a percentage NDS gain that exceeds the expected percentage loss in performance.

To evaluate NDS and MinI, we simulate an erasure-coded, distributed storage system with the three data-allocation schemes. Using the simulator, we compare MinI to two other scheduling policies. Our results show that the three allocation schemes achieve substantially different NDS over time, even when they are subjected to the same device failures. Furthermore, our results show that the gain in reliability due to MinI differs for different data-allocation schemes. We find that MinI increases reliability significantly, as compared to conventional policies.

2 Dynamic Metrics and NDS

In this section, we discuss the space of possible dynamic reliability metrics as part of the motivation for NDS. We also describe the guidelines that drove the definition of NDS, before actually describing it.

2.1 Metric Classes

The reliability status of distributed storage systems changes dynamically, as nodes and disks fail, their data is reconstructed, and the failed devices are replaced or repaired. Trying to quantify the reliability of the system over time involves one fundamental choice: Should we quantify reliability at each point in time with respect to (1) the potential future events that may lead to data loss or (2) the events that have already occurred, reducing the redundancy of the data? We refer to metrics that focus on each of these classes of events as "forward-looking" and "backward-looking" metrics, respectively.

Obviously, forward- and backward-looking metrics quantify different aspects of reliability and are not comparable numerically. Intuitively, forward-looking dynamic metrics would be more similar to traditional (static) metrics, such as PDL or MTTDL, in that they would rely on the probability of future events. Backward-looking metrics would not depend on potential future events; rather, they would represent the actual current state of the system.

As we could not find any previous proposals for dynamic reliability metrics, we set out to develop one metric of each class. We started out trying to create a forward-looking metric that would estimate the probability of data loss at each point in time, and called it "Dynamic Probability of Data Loss" (D-PDL). D-PDL involved defining the number of combinations of potential future failure events every time the configuration of the system changed, i.e., when devices failed, data blocks were reconstructed, or devices were replaced or repaired. The number of these possible combinations would then have to be multiplied by the probability that they would occur, which would be added up together.

We attempted to create a closed-form D-PDL formula and, when that attempt failed, also considered computing D-PDL by enumerating all possible failure scenarios using a computer program. We were not able to create a closed-form D-PDL formula because, in modern distributed storage systems, each device failure typically causes data to be reconstructed (or replicated) at multiple other devices for increased performance. For example, a disk failure is handled by immediately reconstructing each block stored on the failed disk and storing it on different disks. This creates complex relationships between disks that cannot easily be represented in closed form. In considering computing D-PDL using a program, we realized that such a computation would be hopelessly inefficient, as the number of possible future failure combinations is very large for systems of non-trivial size. Furthermore, for a precise estimation of D-PDL, we would also have to keep track of where each reconstructed block was stored, which again is highly undesirable for a general reliability metric.

Interestingly, the difficulties we faced would probably affect other forward-looking metrics as well. For example, we expect that trying to compute a dynamic version of MTTDL would be seriously complicated by reconstruction operations in large distributed storage systems.

For these reasons, we decided to consider backward-looking dynamic reliability metrics, which can be much more easily expressed in closed form and computed. In particular, we propose to compute a reliability score based on the amount of redundancy that has become unavailable at each point in time. In other words, we focus on how far the system currently is from normal operation, as opposed to how likely data loss currently is or how long it would take to lose data. Hence, we call our proposed metric "Normalcy Deviation Score". We are not aware of any previous works defining backward-looking reliability metrics.

2.2 Normalcy Deviation Score

Before presenting the metric, we discuss our guidelines for it: (1) we wanted a metric that could be efficiently computed dynamically; (2) we wanted a metric that was flexible and parameterizable by the storage system administrator; (3) we wanted to heavily weight blocks that are close to being lost in the metric; (4) along the same lines, we wanted to make sure that serious losses in the redundancy of relatively few blocks were not amortized by vast amounts of remaining redundancy in the system; (5) we wanted to take the time to reconstruct the data on a disk into consideration in the metric. Time to reconstruct lost redundancy is becoming ever more important to reliability, as disk sizes have been increasing consistently. Taking time into account also allows for different data allocations under the same failure events to be directly compared.

Based on these guidelines, we define NDS at time t as:

$$NDS(t) = (\sum_{i=0}^{k-1} b_i(t) \times f^{k-i}) \times T_{alloc}$$

where k is the level of redundancy of *all* blocks under normal operation, b_i is the number of blocks that have i levels of redundancy left at time t, T_{alloc} is the minimum time to reconstruct a disk, and f is a scaling factor chosen by the system administrator.

The administrator can pick f to reflect how much more critical the loss of an additional level of redundancy is. For example, for f = 10, each level of redundancy lost harms the system by an additional order of magnitude. The k value depends on the level of redundancy built into the system. We represent the system redundancy by its (n, m) notation, where each data block is either replicated, striped, or encoded into n fragments, but only m ($m \leq n$) of which are required to read the block. k is equal to $n - m$. For example, a RAID-1 system can be described as $(2, 1)$ with $k = 1$, since each block has a replica but only one of the copies (fragments) is necessary to read the block. T_{alloc} depends on the data allocation as we discuss in the next subsection.

This formulation of NDS achieves all of our goals. In particular, (1) it can be efficiently computed dynamically, since information about b_i is readily available (any distributed storage system needs data-allocation and device-failure information), the exponential components can be easily pre-computed, and T_{alloc} is a constant that can also be easily pre-computed; (2) it allows the administrator to weight the loss in redundancy in a flexible manner by setting f appropriately; (3) it weights losses in redundancy exponentially (and the number of blocks at each redundancy level linearly); (4) it is not affected by the blocks that have not lost redundancy, since i ranges from 0 to $k - 1$; and (5) it considers the time to reconstruct a disk linearly.

Note that NDS is unit-less. Under normal operation, the value of the metric equals 0. If all blocks lose all of their redundancy (i.e., one more failure anywhere in the system will cause data loss), the value becomes $B \times f^k \times T_{alloc}$, where B is the total number of blocks in the system. When data is lost, we define NDS to be infinity. Thus, lower values for the metric are better. Although we do not consider this variation here, we could normalize the NDS values with respect to the worst possible score (before data loss), giving us a range from 0 to 1 for the normalized scores.

NDS allows us to compare states of the same system or states of different systems that have the same redundancy scheme (i.e., same n and m) but different data allocations. NDS does not allow us to directly compare systems with different redundancy schemes. We could have tried to eliminate this restriction by including a component quantifying the overall redundancy in the system. However, doing so would violate our guideline (4). NDS can be combined with standard (static) reliability metrics, such as PDL and MTTDL, which can be used to estimate reliability under normal operation.

2.2.1 NDS and Data Allocations

As we mentioned above, T_{alloc} depends on the data allocation. In this paper, we consider three common data allocations: clustering [4], chained declustering [9], and declustering [5]. Below, we define T_{alloc} for each data allocation and refer to it as T_{clus}, T_{cdc}, and T_{declus}, respectively.

Clustering. Clustering places the fragments of data blocks so that the number of disks that store fragments of the same blocks is minimized. To illustrate, Figure 1(a) presents a $(2, 1)$ storage system with 4 disks and a total of 12 blocks, each of them with two fragments. Only one of the fragments is necessary to read the block. In the figure, $0 : 1$ indicates block 0 and fragment 1. As shown, disks 0 and 1 store fragments of blocks 0 to 5 whereas disks 2 and 3 store fragments of blocks 6 to 11. As shown in Figure 1(b), if disk 0 fails, the only data that is left vulnerable to the next failure is that stored on disk 1; the only way there could be data loss is if disk 1 fails. However, to reconstruct the fragments that are stored on disk 0, disk 1 is the sole source, i.e.

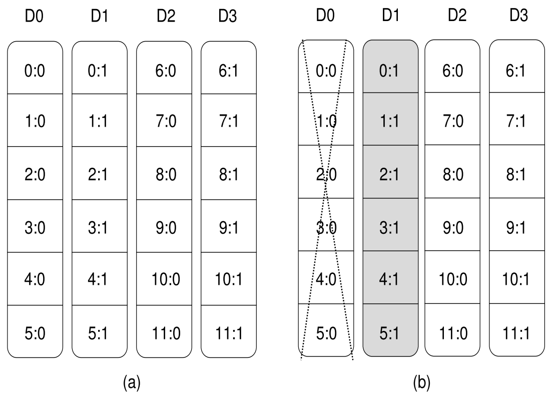

Figure 1. *Clustering for $(2, 1)$ redundancy and 4 disks: (a) no failure, (b) disk 0 failed.*

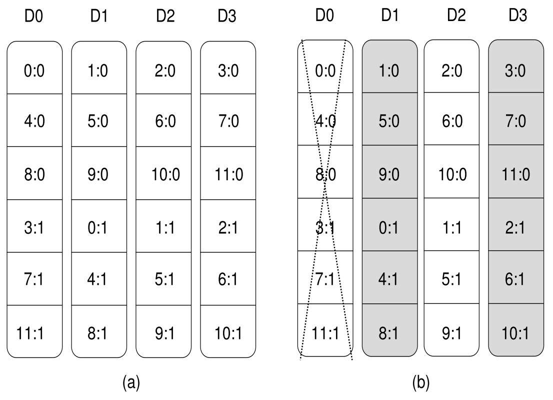

Figure 2. *Chained declustering for $(2, 1)$ redundancy and 4 disks: (a) no failure, (b) disk 0 failed.*

it is the only disk on the "recovery set" of all fragment reconstructions. This causes all reconstructions to contend for recovery bandwidth on disk 1. The optimal schedule (for a constant recovery bandwidth) in this case would be to execute the reconstructions sequentially, rather than in parallel.

In general, performing reconstructions with overlapping recovery sets splits the recovery bandwidth of the overlapping disks, thereby slowing all the reconstructions down. Under clustering, the maximum number of reconstructions that can be executed in parallel after a disk failure is $\lfloor (n-1)/m \rfloor$. When the recovery bandwidth is constant, executing this number of reconstructions in parallel produces the minimum reconstruction time. Thus, the minimum time to reconstruct all the data fragments of a failed disk under clustering is: $T_{clus} = d_{size}/(b_r \lfloor (n-1)/m \rfloor)$, where d_{size} is the amount of data stored on the failed disk and b_r is the recovery bandwidth.

Note that T_{clus} only considers data reconstructions, disregarding the transfer of those data back to their original disk, after it is replaced/repaired and reintegrated into the system. The reason for this choice is that NDS is concerned with redundancy; reconstructions increase redundancy after a hardware failure, whereas transfers back to original disks do not. Furthermore, note that T_{clus} is the minimum time to reconstruct the data, even when the disk is quickly replaced/repaired, since we always assume that the disk is empty when it comes on-line, i.e. the entire contents of the disk have to be reconstructed before they can be copied back. T_{cdc} and T_{declus} below are defined in the same way.

Chained declustering. Chained declustering distributes the fragments of each block so that they are stored on logically neighboring disks in a balanced way. For example, Figure 2(a) shows the placement of fragments under this redundancy scheme. If a disk fails, say disk 0, both disks 1 and 3 can serve as the source for the reconstructions, as shown in Figure 2(b). Two fragments may be reconstructed in parallel, reducing the overall reconstruction time and the time during which data is vulnerable, in comparison to the clustering allocation scheme. However, the failure of either disk 1 or 3 would lose data, if it failed before the data on disk 0 is reconstructed.

Given this allocation scheme, the maximum number of reconstructions that can be executed in parallel after a disk failure is $\lfloor 2(n-1)/m \rfloor$. Assuming that the recovery bandwidth is constant, the minimum time it will take to reconstruct the data stored on a failed disk is: $T_{cdc} = d_{size}/(b_r \lfloor 2(n-1)/m \rfloor)$.

Declustering. Declustering (short for Group Rotated Declustering [5]) distributes fragments of data blocks to minimize the degree of co-location among disks. This leads to a balanced reconstruction load across the active disks in the group. Figure 3(a) shows the placement of data fragments for declustering. As shown in Figure 3(b), after disk 0 fails, each of the remaining disks could serve as the source for reconstruction of exactly two fragments, allowing up to 3 reconstructions to take place in parallel. However, data will be lost if another disk fails before the data stored on disk 0 can be reconstructed.

The time to complete disk reconstruction will be the minimum for declustering among the three schemes. Since declustering spreads fragments evenly, the number of reconstructions that can potentially be executed in parallel is $\lfloor (d_g - 1)/m \rfloor$, where d_g is the number of disks in each group, i.e., the number of disks over which the data of each disk is spread. Again assuming that the recovery bandwidth is constant, the minimum time it will take to reconstruct the data of a failed disk is: $T_{declus} = d_{size}/(b_r \lfloor (d_g - 1)/m \rfloor)$.

Example. To illustrate the NDS values observed over time, Figure 4 plots NDS using $f = 2$ and $f = 10$ for chained declustering for a simulated $(12, 9)$ erasure-coded distributed storage system with 24 nodes, each of them with 8 disks. (More details about our simulation methodology and parameters are presented later.) The X-axis of the graph

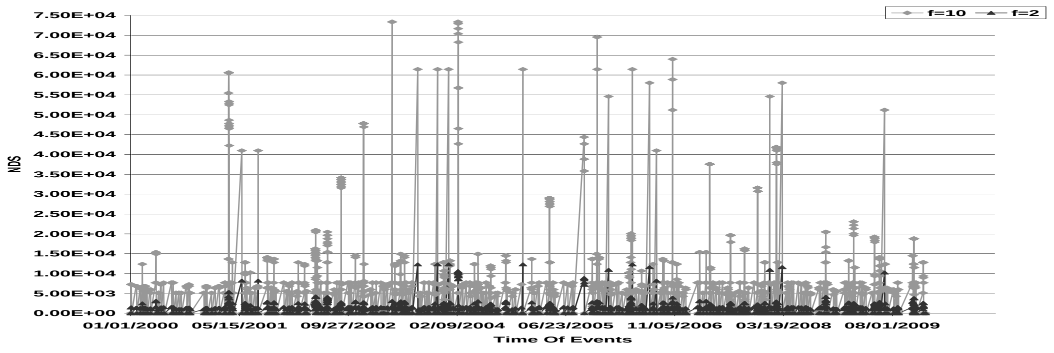

Figure 4. *NDS values for chained declustering for $f = 2$ and $f = 10$ during a period of 10 years.*

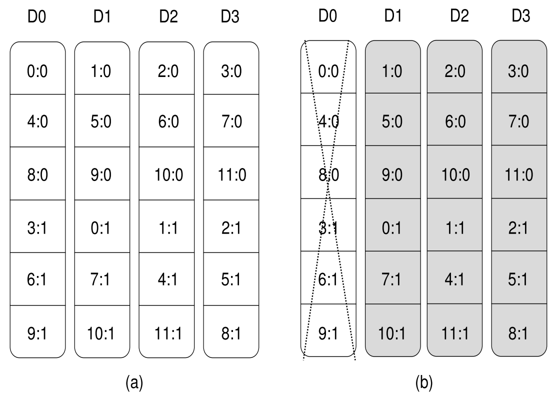

Figure 3. *Declustering for $(2, 1)$ redundancy and 4 disks: (a) no failure, (b) disk 0 failed.*

represents 10 years of simulated time, while the Y-axis represents the NDS values. Each point in the graph represents the NDS of the system at that time. NDS was recomputed after every reliability event (device failure, data reconstruction, and device replacement or repair).

The spikes in the value of NDS represent instances when multiple devices failed concurrently. In particular, from left to right, the ten highest spikes when $f = 10$ occurred when fragments from two disks were being reconstructed concurrently (first and eighth spikes); one node failed (third, fourth, fifth, seventh, and nineth spikes); one disk and one node were being reconstructed concurrently (sixth spike); and one disk failed before the data from another disk was fully reconstructed (second and tenth spikes).

Note that the spikes are not as pronounced when $f = 2$. In fact, some spikes with $f = 10$ do not even occur with $f = 2$, since the impact of a relatively small number of blocks with low redundancy is less pronounced in the latter case. These observations suggest that $f = 10$ is a better setting when one wants to exacerbate the NDS values during reduced redundancy periods.

3 Recovery Scheduler

We propose a new recovery scheduling policy, called MinI, that selects a recovery set for each fragment reconstruction and orders the set of reconstructions to minimize the overall reconstruction time. MinI uses NDS to tradeoff performance and reliability.

3.1 Guiding Principles

The scheduling problem that MinI addresses is very hard to solve optimally for large storage systems (most scheduling problems are NP-hard), so we use a greedy heuristic. MinI is based on the following guiding principles:

1. MinI prioritizes reconstructions based on their current redundancy levels. The lower the current amount of redundancy, the higher the priority. Reconstructing the fragments of the most vulnerable data blocks first ensures that the possibility of data loss if additional failures occur is minimized. In addition, reconstructions have higher priority than copy backs; the latter do not increase redundancy, so they are only allowed to proceed when no reconstructions are taking place on the corresponding disks.
2. MinI selects a recovery set for each reconstruction to maximize parallelism, while avoiding interference. To reconstruct a fragment, any m of the remaining fragments of the same block can be used. MinI leverages this flexible choice of source disks to minimize the intersection among the recovery sets of concurrent reconstructions. It uses a two-dimensional diskscore and a greedy algorithm to choose m disks with the smallest diskscores. Reconstructions that cannot not be made completely independent (i.e., they must have overlapping recovery sets) are only run concurrently if the gain in reliability, as computed by NDS, would justify the potential loss in regular-access performance.
3. Increasing the disk bandwidth allotted to recoveries improves overall reliability [21]. However, higher recovery

bandwidth results in lower bandwidth for actual accesses to the storage system. Furthermore, increasing the recovery bandwidth of all the disks in the system may be inefficient if only a small set of disks are the bottleneck in the recovery process. MinI dynamically increases the recovery bandwidth for the subset of disks that participate in multiple reconstructions up to a certain pre-established limit. This approach results in higher system reliability for a small loss in regular-access bandwidth.

3.2 MinI Policy

MinI takes the set of reconstructions to be performed as input and produces a schedule as output, containing the reconstructions that should be executed next and the recovery sets that they should use. To compute the schedule, MinI divides the set of reconstructions into separate queues based on their remaining amounts of redundancy, i.e. reconstructions for blocks that have the same number of remaining fragments are grouped together. The policy starts by scheduling the reconstructions associated with the non-empty queue that has the least amount of redundancy left. An *intersection matrix* is computed for these reconstructions, as described in detail in Section 3.2.1.

From the intersection matrix, MinI chooses the pair of reconstructions that have sets of potential disk sources with smallest intersection. If there are multiple pairs with the smallest intersection, a random pair in this set is selected. (We also considered a more sophisticated tie-breaking approach that minimized future intersections within the same redundancy level, but it produced only negligible improvements in comparison to the random approach.) After that, MinI selects recovery sets for the chosen reconstructions using a two-dimensional *diskscore*, as described in Section 3.2.2. If the chosen reconstructions have overlapping recovery sets, MinI adds them to the schedule depending on a *tradeoff between reliability and performance*. The actual tradeoff function can be provided by the user, as described in Section 3.2.3.

The policy then iterates through the remaining reconstructions in the current redundancy-level queue, chooses the reconstruction that has the smallest intersection with the reconstructions already in the schedule (again looking at the intersection matrix for this redundancy level), assigns recovery sets, and trades off reliability and performance, as mentioned above.

It repeats the above process for the reconstructions in the other redundancy-level queues, in increasing order of redundancy left. For each other redundancy level, intersections are computed with respect to those reconstructions from previous queues that appear in the schedule and the reconstructions in the current queue. Information about the latter intersections appears in the current intersection matrix. For each redundancy level, no additional reconstructions have to be considered after the first is rejected for inclusion in the schedule.

The policy halts when reconstructions across all the redundancy-level queues have been considered once for inclusion in the schedule. Any reconstructions that were not included in the schedule will be considered again after the current schedule is performed.

Implementation. When implemented as part of an actual system, the MinI policy is activated by a recovery manager whenever there are reconstructions to be performed. The manager keeps track of the required reconstructions by querying each storage node about any failed disks, their contents, and the potential sources for each fragment reconstruction. The manager immediately schedules reconstructions concurrently according to the output of MinI. When these reconstructions complete, the manager calls MinI again, until all reconstructions have been performed.

Again by interacting with the storage nodes, the manager finds out about disk replacements. After any of these reliability events (failures, reconstructions, and replacements), the manager computes the NDS of the system using the models of Section 2.

3.2.1 Intersection Matrix

An intersection matrix is computed for each redundancy-level queue. The matrix contains the size of the pairwise intersection of the potential source sets of the reconstructions in that queue. The i^{th} row contains the size of the intersection of the source set of the i^{th} reconstruction with all the remaining reconstructions in that queue. Thus, each intersection matrix is symmetric, i.e. the intersection (i, j) is the same as (j, i).

3.2.2 Diskscore

The diskscore is a two-dimensional score computed for all the disks in the system. The diskscore comprises a *static score* and a *dynamic score*. The static score of a disk indicates the number of reconstructions in which it could participate as a source or destination. The dynamic score of a disk indicates the number of *scheduled* reconstructions whose recovery set it belongs to either as a source or destination.

Initially, all disks are assigned a diskscore of $0 : 0$. The first number indicates the static score and the second the dynamic score. MinI iterates through the reconstructions and, for each disk that is a potential source for some reconstruction, it increments the static score of the disk. The dynamic score is updated when MinI adds reconstructions to the current schedule. Comparing the diskscores of two disks involves first comparing their dynamic scores and, only if there is a tie, comparing their static scores later.

MinI uses the diskscore of the disks in the potential source set to choose m disks with the smallest diskscores. If the destination disk is not chosen already (it may have been chosen if the same reconstruction had been started before but interrupted by another event in the system), the disk with the smallest diskscore among the other available disks is chosen and its dynamic score is also incremented.

3.2.3 Reliability vs. Performance

MinI leverages NDS to tradeoff reliability and performance: it only schedules two non-independent reconstructions in parallel if doing so would improve NDS enough compared to the potential loss in performance.

The reason for a potential performance loss is that MinI assigns recovery bandwidth to each reconstruction running concurrently on a disk (up to a user-specified limit discussed below) as if it were running alone on the disk. This means that reconstructions with overlapping recovery sets take away bandwidth that could be used for regular storage accesses. Thus, when trading off performance and reliability, performance is represented by the percentage loss in regular-access bandwidth.

The gain in NDS is computed as the percentage difference between the NDS value before the reconstruction and the predicted NDS value after the reconstruction.

When the recovery set of a reconstruction overlaps with the recovery sets of other reconstructions already on the schedule, MinI compares the sum of the NDS gain of each of the reconstructions on the schedule and the additional performance loss that the system would incur if the recovery bandwidth of the overlapping disks were increased. The actual comparison function can be provided by the user. The default version of MinI uses a linear comparison between reliability gain and the potential loss in performance. In other words, if the percentage gain in reliability is higher than the percentage loss in performance, the reconstruction is added to the schedule.

Finally, there is a user-defined limit on the maximum acceptable performance loss resulting from additional disk bandwidth assigned to reconstructions. MinI jumps to the next redundancy-level queue, if either the gain in reliability is relatively small compared to the loss in performance, or it reaches the performance-loss limit.

A longer version of this paper [2] includes a detailed pseudo-code describing MinI.

4 Evaluating MinI

4.1 Methodology

We simulate a 60 TB distributed $(12, 9)$ erasure-coded storage system. Each fragment is 10 GB to reduce disk access and meta-data management overheads. (These parameters are similar to those of the commercial NEC Hydrastor system.) The simulated system comprises 24 nodes, each having 8 disks. Each disk has capacity of 300 GB and is roughly 50% full with fragment data. We study three data-allocation schemes: clustering, chained declustering, and declustering. Regardless of the scheme, no node (and consequently no disk) can store more than 1 fragment of the same block to avoid compromising redundancy. To simulate the system's behavior over time, we inject node and disk failures using exponential distributions with their corresponding MTTFs as means. Replacements and repairs are also exponentially distributed with their corresponding MTTRs as means. A disk failure requires the failed disk to be replaced. A node failure requires the failed node to be replaced/repaired, but its disks can be reused by the new/repaired node (the node failure does not affect the content of the disks). Regardless of the type of failure, data reconstructions are started as soon as possible. We assume that reads are going on in the system at all times and that no new data is added to the system during the simulation.

In addition to MinI, we simulate two base recovery-scheduling algorithms: *Base* and *BaseT*. The Base algorithm schedules all the recoveries (i.e., reconstructions and copy backs) in the system in parallel (splitting the available recovery bandwidth evenly across the recoveries) and chooses the recovery sets of concurrent recoveries in round-robin fashion. The recovery bandwidth is never increased in Base. BaseT is similar to Base, except that it increases the recovery bandwidth used by a disk when multiple recoveries execute concurrently on it. BaseT increases the bandwidth by adding a pre-defined acceptable performance loss (i.e., disk bandwidth that is taken away from regular accesses and given to recoveries) threshold to it; the increased bandwidth is split evenly across the concurrent reconstructions and copy backs, if any. The performance loss threshold is the same as that used in MinI. (Recall that, in MinI, copy backs are assigned the lowest priority, so they are scheduled when there are no reconstructions executing on the participating disks. Additionally, there is no extra bandwidth assigned to copy backs in MinI.) Table 1 summarizes the values of the parameters used in our simulations.

4.2 Results

In this subsection, we first present our simulation results for MinI under chained declustering, and then compare them with those for clustering and declustering.

Time to compute schedule. MinI takes very little time to compute its schedules, especially when compared to the overall time taken to actually effect the schedules. Specifically, under our default simulation parameters and chained declustering, it takes at most 0.08 seconds to compute the

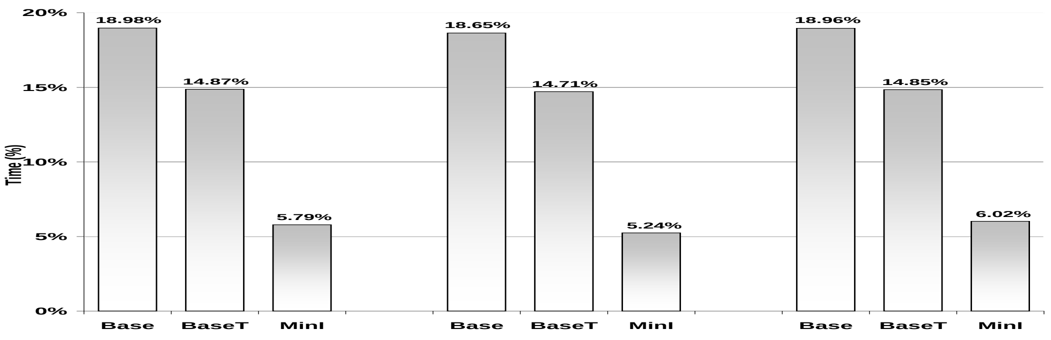

Figure 5. *Partial redundancy in chained declustering under Base, BaseT, and MinI.*

Simulation parameter	**Default**
Number of fragments (n)	12
Number of required fragments for reads (m)	9
Disk MTTF	$50,000$ hours
Disk MTTR	24 hours
Node MTTF	$100,000$ hours
Node MTTR	24 hours
Number of nodes	24
Disks per node	8
Number of data blocks	240
Data fragment size	10 GB
Disk capacity	300 GB
Disk bandwidth	20 MB/s
Performance threshold	5%
Percentage of disk bandwidth for recovery	5%
f factor (NDS)	10
Simulated time	10 years

Table 1. Parameters and their default values.

MinI schedule to reconstruct the 15-17 fragments (10 GB each) stored on a failed disk. Upon a node failure, it takes roughly 3.4 seconds to compute the MinI schedule to reconstruct the 8 disks associated with the node. Compare these times with the times to actually perform the reconstructions, which are at least 170 minutes (disk failure) and 680 minutes (node failure). During our 10-year simulations, the system takes roughly 2 minutes computing the MinI schedules. These results indicate that the overhead of MinI is minimal.

Improvements in reliability. The key goal of MinI is to improve reliability by speeding up data reconstructions. One aspect of this goal is to bring the system back to a state of full redundancy as quickly as possible after one or more failures occur. Figure 5 depicts the fraction of time each scheduling policy leaves the system with partial redundancy; during the remaining fraction of time (i.e., 100% minus the fraction reported in the figure) the policy leaves the system with full redundancy. Each set of three bars corresponds to a different set of failure events over the course of 10 years. As we can see in the figure, MinI significantly increases the amount of time in full redundancy, with respect to both Base and BaseT, by significantly speeding up data reconstructions. Specifically, MinI reduces the percentage of time in partial redundancy by 70% (compared to Base) and 61% (compared to BaseT) on average, while increasing the percentage of time in full redundancy by 16% and 11% on average, respectively. The figure also shows that these trends are consistent across different 10-year simulations.

Another important goal of MinI is to improve reliability during periods of partial redundancy by prioritizing the most critical data reconstructions and sequencing them. Figure 6 presents the Cumulative Density Function (CDF) of the NDS values during one representative 10-year-long set of failure events. In other words, for each point (x, y) in the graph, the system was at an NDS value $< x$ for y% of the time. For clarity, the figure focuses solely on the periods with partial redundancy, i.e. when $NDS \neq 0$; the leftmost point in each curve represents the percentage of time in full redundancy. As we can see, MinI reduces the amount of time the system spends in the different partial-redundancy states, as compared to Base and BaseT.

We also considered the effect of MinI on the time to data loss. To do so, we ran 100 simulations with Base and MinI, assuming near-future 1.5-TByte disks (and 50-GByte fragments). Each run simulates the system for 200 years or until the first instance of data loss, whichever occurs first. The results show that the system suffers data loss in 18% of the runs under MinI–i.e., in 82% of the MinI runs the system does not suffer data loss in its first 200 years. In stark contrast, when Base is the scheduling policy, the system suffers data loss in 60% of the runs. In addition, MinI lengthens the average time to data loss by 40% when it does happen: 85 years for MinI and 54 years for Base. Most strikingly, the minimum time to data loss with MinI (14.5 years) is 6 times higher than that with Base (2.4 years). The BaseT results lie between these extremes. Under BaseT, 21% of the runs suffer data loss in the first 200 years. When data loss occurs, BaseT achieves an average time to data loss of 73 years and a minimum time to data loss of just 2.5 years.

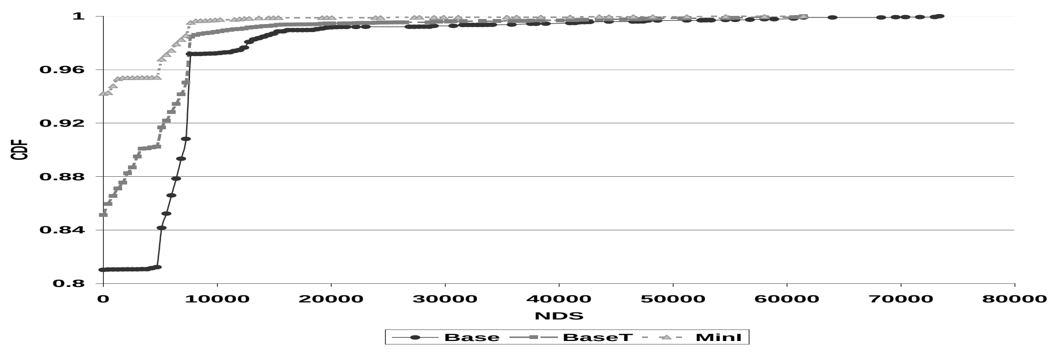

Figure 6. *CDF of NDS values for chained declustering under Base, BaseT, and MinI.*

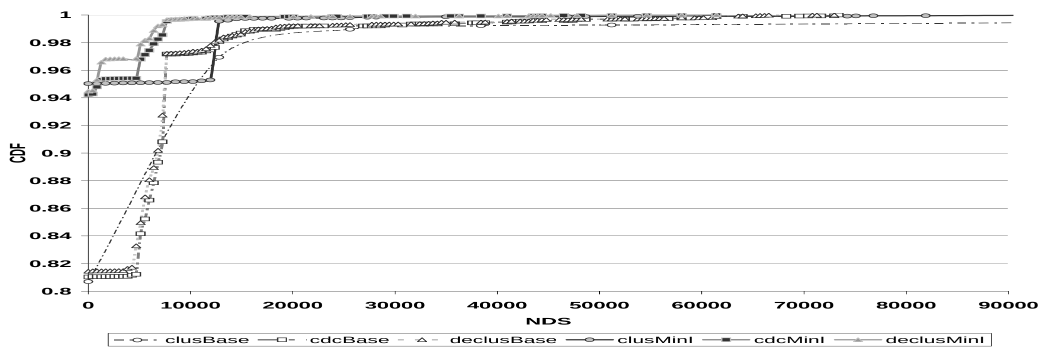

Figure 7. *Comparing allocations under Base and MinI. The right side of the graph has been cut for clarity.*

Effect of data allocation. Figure 7 depicts the CDF of the NDS values of Base and MinI (we do not show results for BaseT for clarity, but our observations apply to it also) for the three data allocations we consider in a representative 10-year period. So far, we had only presented results for chained declustering.

The figure shows that MinI improves reliability for clustering and declustering as well. In fact, MinI improves reliability for these data allocations in all respects: increased time in full redundancy, reduced time at lower redundancy levels, and reduced maximum NDS value.

Chained declustering and declustering accrue the largest gains from MinI. Under clustering, the reconstructions have overlapping recovery sets and, therefore, there is not as much parallelism that MinI can exploit. As a result, clustering leads to very large NDS values even under MinI (seen on the right of the figure). The overlapping recovery sets (and the fact that all reconstructions associated with each failure end at the same time) are also the reason why clustering under the Base policy exhibits many fewer NDS values. On the other hand, under chained declustering and declustering, the potential recovery sets overlap much less, leading to relatively low (yet numerous) NDS values under MinI.

Additional results. Our longer technical report [2] studies the effect of the recovery bandwidth and the performance loss threshold. It also isolates the benefits of the key aspects of MinI: serializing and prioritizing reconstructions, and using additional bandwidth beyond the default recovery bandwidth. In short, the results show that recovery bandwidths and performance losses have to be unreasonably high for Base and BaseT to approach the benefits of MinI. Finally, the results demonstrate that all aspects of MinI are important, especially serialization and additional bandwidth.

5 Related Work

Quantifying reliability. Several papers have quantified PDL and/or MTTDL for storage systems either by running large numbers of simulations or modeling reliability explicitly, e.g. [3, 8, 12, 14, 15, 17, 18, 20, 21]. Combinatorial modeling of reliability has also been used to estimate PDL, e.g. [20]. These previous papers quantified reliability with a single value for the entire lifetime of the system. In contrast to these forward-looking metrics, we focused on NDS, a dynamic, backward-looking metric. We are not aware of any other dynamic, backward-looking reliability metrics.

Improving reliability. Several works have sought to improve reliability through sophisticated data-allocation schemes, e.g. [4, 5, 6, 9, 14]. However, some recent studies [1, 19] have shown that no data-allocation scheme works well for all workloads and, hence, it is important to develop techniques to improve reliability that are independent of the underlying data allocation.

MinI is independent of data allocation (although the NDS estimation used by MinI is tied to the data allocation). Previous studies of recovery times did not consider scheduling concurrent recoveries efficiently [14, 21]. For example, [21] introduced the notion of assigning a different destination disk to each fragment reconstruction. The distributed systems we study in this paper take this approach. However, [21] (and [14]) did not consider how to schedule the accesses to the source disks for the reconstructions.

More along the lines of our work, [7, 11] do discuss the importance of recovery scheduling. However, they consider scheduling recoveries at the application level in the context of data centers. Their aim is to schedule the backups and recoveries of different applications to reduce the recovery time as seen by the applications. Their scheduling policies consider high-level storage units, such as an entire disk array, rather than individual disks as in MinI. Further, MinI is implemented at the storage-system level and is independent of the applications running above it. MinI can be used in tandem with [7, 11] and is orthogonal to other reliability improvement techniques.

Finally, Jiménez-Peris *et al.* proposed a protocol for improving the reliability of replicated databases [10]. Their protocol updates the nodes that have recovered from failures in parallel, while guaranteeing the consistency of the data. In contrast, MinI operates at a lower level and focuses on selecting the sources for efficiently reconstructing data when nodes/disks first fail.

6 Conclusion

In this paper, we presented a new reliability metric, called NDS, that efficiently and dynamically computes a score representing how far the system is from its fully redundant state. Using NDS, we also proposed a recovery scheduler, called MinI, that efficiently schedules recoveries based on a tradeoff between reliability and performance. Our results demonstrate that NDS and MinI contribute to significant increases in reliability. We now plan to evaluate NDS and MinI in the context of the *Hydrastor* distributed storage system currently being sold by NEC.

References

[1] M. Abd-El-Malek *et al.* Ursa Minor: Versatile Cluster-based Storage. In *Proceedings of FAST*, December 2005.

[2] R. Bachwani, L. Gryz, R. Bianchini, and C. Dubnicki. Dynamically Quantifying and Improving the Reliability of Distributed Storage Systems. Technical Report DCS-TR-639, Dept. of Computer Science, Rutgers University, July 2008.

[3] M. Baker *et al.* A Fresh Look at the Reliability of Long-term Digital Storage. In *Proceedings of EuroSys*, April 2006.

[4] D. Bitton and J. Gray. Disk Shadowing. In *Proceedings of VLDB*, August 1988.

[5] S. Chen and D. Towsley. A Performance Evaluation of RAID Architectures. *IEEE Transactions on Computers*, 45(10), October 1996.

[6] P. M. Chen *et al.* RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys*, 26(2), 1994.

[7] S. Gaonkar *et al.* Designing Dependable Storage Solutions for Shared Application Environments. In *Proceedings of DSN*, June 2006.

[8] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly Durable, Decentralized Storage Despite Massive Correlated Failures. In *Proceedings of NSDI*, May 2005.

[9] H.-I Hsiao and D. DeWitt. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. In *Proceedings of ICDE*, February 1990.

[10] R. Jimenez-Peris, M. Patino-Martinez, and G. Alonso. An Algorithm for Non-Intrusive, Parallel Recovery of Replicated Data and Its Correctness. In *Proceedings of SRDS*, October 2002.

[11] K. Keeton *et al.* On the Road to Recovery: Restoring Data After Disasters. In *Proceedings of EuroSys*, April 2006.

[12] R. Kotla, M. Dahlin, and L. Alvisi. SafeStore: A Durable and Practical Storage System. In *Proceedings of USENIX*, June 2007.

[13] J. Kubiatowicz *et al.* OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of ASPLOS*, October 2000.

[14] Q. Lian, W. Chen, and Z. Zhang. On the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems. In *Proceedings of ICDCS*, June 2005.

[15] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of SIGMOD*, June 1988.

[16] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Proceedings of SOSP*, October 2001.

[17] Y. Saito *et al.* FAB: Building Distributed Enterprise Disk Arrays from Commodity Components. In *Proceedings of ASPLOS*, October 2004.

[18] T. J. E. Schwarz *et al.* Disk Scrubbing in Large Archival Storage Systems. In *Proceedings of MASCOTS*, October 2004.

[19] E. Thereska *et al.* Informed Data Distribution Selection in a Self-Predicting Storage System. In *Proceedings of ICAC*, June 2006.

[20] A. Thomasian and M. Blaum. Mirrored Disk Organization Reliability Analysis. *IEEE Transactions on Computers*, 55(12), 2006.

[21] Q. Xin, E. L. Miller, and T. J. E. Schwarz. Evaluation of Distributed Recovery in Large-Scale Storage Systems. In *Proceedings of HPDC*, June 2004.