Tools for Predicting the Reliability of Large Scale Storage Systems

Robert J. Hall, AT&T Labs Research

Data intensive applications require extreme scaling of their underlying storage systems. Such scaling, together with the fact that storage systems must be implemented in actual data centers, increases the risk of data loss from failures of underlying components. Accurate engineering requires quantitatively predicting reliability, but this remains challenging, due to the need to account for extreme scale, redundancy scheme type and strength, distribution architecture, and component dependencies. This paper introduces CQSIM-R, a tool suite for predicting the reliability of large scale storage system designs and deployments. CQSIM-R includes (a) direct calculations based on an only-drives-fail failure model and (b) an event-based simulator for detailed prediction that handles failures of and failure dependencies among arbitrary (drive or non-drive) components. These are based on a common combinatorial framework for modeling placement strategies. The paper demonstrates CQSIM-R using models of common storage systems, including replicated and erasure coded designs. New results, such as the poor reliability scaling of spread-placed systems and a quantification of the impact of data center distribution and rack-awareness on reliability, demonstrate the usefulness and generality of the tools. Analysis and empirical studies show the tools' soundness, performance, and scalability.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Reliability

General Terms: Reliability, Modeling, Simulation, Prediction

Additional Key Words and Phrases: Tools, large scale, storage systems

ACM Reference Format:
Robert J. Hall. 2016. Tools for predicting the reliability of large-scale storage systems. ACM Trans. Storage 12, 4, Article 24 (July 2016), 30 pages.
DOI: http://dx.doi.org/10.1145/2911987

1. INTRODUCTION

Modern data intensive systems need larger and larger underlying storage systems that must be highly reliable. For example, CERN's Large Hadron Collider generates 30 PB ($\approx 30 \times 2^{50}$ bytes) of data per year [CERN-LHC], and it would presumably be unacceptable if hard-won evidence of the Higgs Boson were lost due to storage failures. Consumer cloud storage systems could easily grow into the exabyte range ($\approx 2^{60}$ bytes) with tens of millions of people storing tens of gigabytes each of baby videos and other precious data.

While no storage system can be completely invulnerable to loss, well known redundancy designs, including replication and erasure codes, can reduce the probability of loss over a fixed period to a level small enough to be compensated for by other means, such as archival [Storer et al. 2008] or cold [Patiejunas 2014] backup storage, recomputation, or insurance. However, key to engineering redundant storage is to have a method of quantitatively predicting the reliability of a given storage system *both theoretically and as it is implemented in physical data centers.* One use of this is to help

Author's address: AT&T Labs Research, 1 AT&T Way, Bedminster, NJ 07921; email: hall@research.att.com.

decide which erasure code strength to deploy, a decision having significant economic consequences; another is to inform distribution and layout decisions.

In addition to large scale, the other key characteristic of large storage systems examined in this paper is *dependency*, which has previously been observed to be a dominant factor in the reliability of large storage systems [Ford et al. 2010]. Data can be lost if a drive fails, but data can also be effectively lost if a host, rack, or data center necessary to accessing the drive go down. Moreover, a rack going down will not only make many drives inaccessible at once, but it is typical that the sudden loss of power will have damaged some of them permanently [Cidon et al. 2013]. Individual components may obey complex stateful failure models, such as Weibull distributions [Elerath and Pecht 2007], while others may be well modeled by simple stateless failure models. So modeling detailed hardware dependencies is critical to calculating reliability of the storage system. Dependency affects the software level also. For example, in a partitioned placement scheme, such as one consisting of two separate RAID arrays, the partitions fail independently at the software level, and the aggregate reliability is straight forward to compute (Section 6) and inversely proportional to capacity ($\Theta(1/d)$ where d is the total number of drives). By contrast, a spread-placed redundant storage system like HDFS [Borthakur 2007] or QFS [Ovsiannikov et al. 2013] places file shares more or less randomly across all the drives, with different files placed on different sets of drives. While each file is individually a k-out-of-n system (in the Angus [Angus 1988] sense), their aggregate reliability is, as will be shown in Section 7, far worse than inversely proportional to capacity.

This paper describes an integrated set of tools, implemented in Java, that address this prediction challenge and makes several novel contributions.

— A general framework (Section 5) for modeling a large class of storage systems at high scale and deployment complexity in which a combinatoric characterization supports both analytical and simulation models;
— New algorithmic formulae for mean time between loss events (MTBLE) and mean loss rate (MLR) when assuming only drives fail (ODF mode reasoning, see Section 6). Embodied in the ODF-MTBLE and ODF-MLR tools, these can handle predictions to arbitrarily high scale when implemented using unbounded-precision arithmetic and are primarily useful for studying reliability scaling and for comparing scaling of redundancy schemes;
— CQSIM-R (Cloud Quality-of-Service Simulator for Reliability, Sec 8), a high performance simulator that allows measuring (modeled) reliability of storage systems deployed in large (e.g. exabyte-) scale data centers. Its event-based design uses the same combinatorial core used by the ODF models and can simulate 1 trillion device-days in about an hour on a laptop.
— A collection of *novel scaling results and quantitative comparisons* that illustrate these tools, including
  — the *spread catastrophe:* the MTBLE of a spread-placed system decays rapidly (proportionally to d^{-s} initially) with increasing capacity d, much faster than that of the corresponding partitioned system (Figure 5);
  — comparisons of five different storage *placement strategies*, showing generality of the framework and tools;
  — a set of *validation results* providing evidence of the tools' soundness, consistency with each other, and compatibility with Angus's k-out-of-n formula [Angus 1988] in the special cases where the latter applies; and
  — the dependence of MTBLE on distribution, both across data centers and across racks and hosts, and how this *helps* partitioned systems while *hurting* spread systems (Sec 9).

— quantifying the reliability improvement of rack-aware spread placement over normal spread.

2. RELATED WORK

Angus [Angus 1988] derives a formula for mean time between failures (MTBF) of a k-out-of-n system. It assumes the system fails once $n - k + 1$ drives have failed and halts until repaired. While, as we show in Section 10, our MTBLE formula is consistent with the Angus model on systems of total size $n = p + s$, Angus's model is not applicable to most larger systems. For example, we cannot treat erasure coded spread placed files as independent k-out-of-n systems (see below). Moreover, we avoid MTBF in favor of MTBLE, as a large scale system does not halt when a file is lost, and repair times may not satisfy Angus' assumptions at large scale.

Resch and Volvovski [Resch and Volvovski 2013] implement an event-based simulation that matches predictions of the Angus formula. Their simulation does not handle either hardware dependencies nor spread or other non-partitioned placements, as do our tools. They also do not compute mean data loss rate (MLR) as do our tools.

Kao et al [Kao et al. 2013] describe a simulation technique that can estimate loss risks in storage arrays. It is based on a applying a probabilistic loss model once the simulation determines how many drives are failed at any given time. However, the user must provide individual loss probabilities for each possible number of drive failures beyond a threshold, which seems problematic for systems with hundreds to thousands of drives. In addition, the approach cannot correctly handle failure dependencies, such as the difference in outcome between random and rack-aware placement when simulating rack failures. CQSIM-R's simulator handles such dependencies accurately (Section 9).

There are several studies that derive more faithful disk failure models from experience data, such as Schroeder and Gibson [Schroeder and Gibson 2007]. The CQSIM-R simulator currently uses a simple exponentially distributed failure model for consistency with the ODF tools, but it can be easily extended to use any of these more detailed drive failure models.

In [Schroeder et al. 2010], Schroeder et al discuss techniques for protecting against *latent sector errors*. When present but undetected, LSEs can defeat redundancy-based recovery, because when the system goes to do recovery (due to other failures), it then discovers that a needed file block is missing. The framework of this paper does not consider this failure mode, so our predictions of MTLBE may be higher than will be experienced in systems with significant LSEs. By adopting the approach of [Schroeder et al. 2010], our framework can more closely reflect reality.

Venkatesan and Iliadis [Venkatesan and Iliadis 2012], on the other hand, show for replicated systems that the use of richer drive failure models (such as those of Schroeder and Gibson) in ODF reasoning does not affect the results very much, when failure rate is much less than repair rate. This supports the usefulness of the simpler exponential models' results in supporting more abstract reasoning, such as studying scaling laws and comparing the scaling of different schemes, as we use them in this paper.

Elerath and Pecht [Elerath and Pecht 2007] give a detailed model of drive failure and repair behaviors that is applied to reliability of RAID systems. Elerath and Schindler [Elerath and Schindler 2014] provide a closed form equation for reliability of RAID-6 systems. These models can be incorporated into CQSIM-R using the sectioning principle (Section 6) or by building them into component models for use within the simulator. They do not scale to large storage systems, as they are focused on single RAID array modeling.

In the large scale study by Pinheiro et al [Pinheiro et al. 2007] of disk failure trends, they discover useful SMART metrics that can predict with high likelihood when a drive is near failure. When these techniques are effectively deployed, they can have the effect of allowing a graceful transfer of a drive's content to a new drive *before* the drive fails completely. This should greatly increase the effective MTTF of drives. However, in practical datacenter deployments, we show here that the non-drive components' failures tend to dominate the reliability calculation anyway, so this will have limited benefit. For example, comparing Figure 11 (a) and (b), the effect of non-drive failures reduces MTBLE by a factor of at least 10 across the full range of content capacity.

Greenan et al [Greenan et al. 2010] compare different reliability metrics and recommend their normalized magnitude of data loss (NOMDL_t) as the best one. That metric is computable from our MLR by multiplying MLR by t, the mission time. Assuming a finite mission time seems less useful in large scale storage systems, as usually one tries to build systems that last and grow indefinitely. MLR is better suited to measuring the *rate* of loss over an indefinite period, rather than the magnitude over a given finite period. On the other hand, they criticize MTTDL, which is closely related to our MTBLE. They are correct in citing the limitations of the ODF reasoning and in memoryless exponentially distributed hardware failure models. However, I believe these models are useful for studying scaling and for comparing different storage schemes to each other. This paper then takes the further step of providing a way to transfer the insights gained from ODF reasoning into real world practice by using the CQSIM-R simulator to study the effects of implementing storage systems in the real world, including dependencies and realistic hardware failure models. This paper gives examples and relationships between ODF reasoning and more detailed simulations.

Iliadis and Venkatesan [Iliadis and Venkatesan 2015] rebut the idea that MTTDL is not useful. They correctly point out that previous papers deprecating it have incorrectly generalized from the invalidity of particular analyses that produce MTTDL to the invalidity of the metric itself. For example, analyses that assume only drives fail (our ODF reasoning, see below) are strongly limited in applicability to the real world, precisely because they abstract away significant effects of failures of other data center components. On the other hand, ODF analyses of MTTDL (or MTBLE, as in this paper) are useful in studying fundamental scaling laws and in comparing reliability of different schemes. This paper goes further by showing how a common combinatorial model core can support both ODF analyses and richer failure models incorporating failures of arbitrary data center components. In all cases, the MTTDL or MTBLE metric itself is useful as long as one understands the limitations of the modeling underlying its calculation.

Rao et al [K.K.Rao et al. 2006] develop Markov chain based analytical models of the MTTDL of several small storage node designs based on RAID concepts. While these analyses do not individually scale beyond small arrays, they can be combined using our sectioning principle (see Section 6.3) to analyze larger systems. They do not deal with spread placed or other non-RAID placement schemes, nor arbitrary erasure code strengths. They also do not model hardware or other deployment dependencies as can be handled with CQSIM-R's simulator. An interesting aspect of their work is their method for estimating rebuild time, which feeds into the equations for reliability. CQSIM-R assumes the repair time for each component is given as *input*. Rao et al's work, by contrast, attempts to compute the duration of repair, at least for rebuilding the data on drives. This estimation of the actual rebuild time is an active area of future work for the CQSIM-R project; it requires performance and load modeling that are beyond the scope of this paper. For now, we assume that the input repair times given to CQSIM-R are upper bounds on repair. If this assumption holds, then CQSIM-R's results are valid and handle even cases where more failures occur during repairs of other components. This is true both for the simulator and for the ODF analytical tools.

There are a number of placement schemes that fall between spread and partitioned on the reliability spectrum, such as copysets [Cidon et al. 2013], rack-aware as in HDFS [Borthakur 2007], and limited-spread (as attributed to Facebook's HDFS in [Cidon et al. 2013]). CQSIM-R can model all of these, and they will be described and compared in Section 6.

Venkatesan et al [Venkatesan et al. 2012] evaluate the effects of bandwidth limitation on a particular placement strategy (our LimitedSpread(z), see Section 8.2). While bandwidth and other resource constraints (CPU, cache size, etc) are outside the scope of the present paper, the framework does accommodate evaluating the reliability for arbitrary repair times, so it can still apply to bandwidth limited systems whose repair times are long. Note the study of [Venkatesan et al. 2012] is itself limited in that (a) it does not model non-drive component failures (even bandwidth partitions), and (b) it only deals with replicated systems. Our framework is more general in both ways.

In 2011, [Venkatesan et al. 2011] derived the counter intuitive result that the declustered replicated placement scheme (which is the same as the SSS(1, x) spread placed scheme in our parlance), actually *increases* in reliability as the number of drives increases. While this appears to conflict with, for example, Figure 7, the difference lies in a difference in underlying assumptions. Our ODF-MTBLE (see Section 6.1) replicates this result qualitatively[1] if we assume that *repair time is inversely proportional to d*. This would be consistent with the assumptions and derivation in Section III-C of [Venkatesan et al. 2011]. However, in this work we focus on studying scaling under the assumption of approximately constant repair time scaling with number of drives. The reason is that even though distributed repair (e.g. as in Ceph [Weil et al. 2006a]) can indeed do a *single* repair in time (approximately) inversely proportional to number of drives, we believe the number of concurrent repairs increases linearly with number of drives, under the ODF independent failures assumption. Thus, the bandwidth, and hence the time, *per repair* stays constant with increasing d, due to the direct dependence of number of repairs on number of nodes canceling out the inverse dependence of per-node, per-repair bandwidth on number of nodes.

The Ceph distributed storage system [Weil et al. 2006a] can be configured for a wide variety of different placement strategies through specifying different *CRUSH maps*. Ceph's CRUSH map [Weil et al. 2006b] defines a set of rules that constrain replica (or share) placement among drives. By characterizing a given CRUSH map in terms of its GLEP and SLEP (as defined below in Section 6) any such Ceph configuration can be modeled and evaluated using the tools described in this paper. It is an interesting future research item to build a tool to do this analysis automatically.

In a paper related to Ceph's design, [Xin et al. 2003] give ODF calculations of MTTDL for three redundancy schemes, 2-way mirrored, 3-way mirrored, and RAID-5+1 and introduce the idea of limiting the number of redundancy groupings for objects (less than all possible). I discuss this paper's limiting-scheme relative to this paper in Section 11. Their paper does *not* evaluate impact of non-drive failures on the discussed redundancy schemes.

3. CONTEXT: REDUNDANT STORAGE SYSTEMS

3.1. Modeling Hardware Components

Storage systems must be implemented out of physical components. These suffer failures over time due to many things, including normal use and extraordinary events like power outages. Each physical component is assumed to experience failures according to some distribution, and also to undergo repairs at a rate which can be modeled. In this

[1]The approaches do not match exactly due to analytical approximations in [Venkatesan et al. 2011]

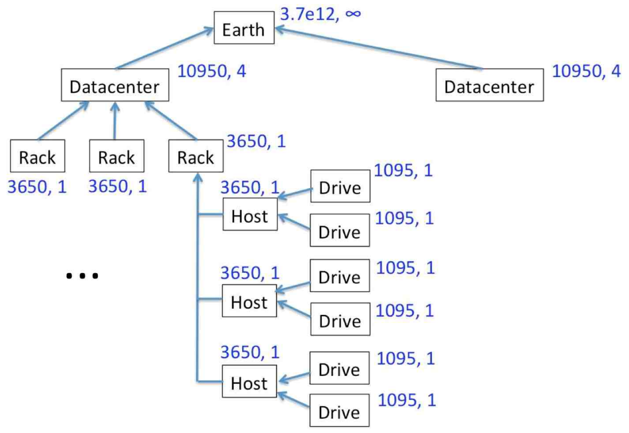


Fig. 1. Example hardware model. (MTTF, MTTR) for each individual component (in days) is shown.

paper, we assume hardware elements obey a simple exponentially distributed failure model characterized by its mean time to failure (MTTF). Moreover, we assume repairs are characterized by a duration termed the mean time to repair (MTTR). The tools can handle exponentially distributed repairs, if desired, but in this paper we use constant deterministic repairs unless otherwise stated. While our ODF analytical results (Section 6) require these simple failure and repair behaviors, the CQSIM-R simulator can easily incorporate arbitrarily complex, possibly stateful, failure and repair behaviors [Schroeder and Gibson 2007; Elerath and Pecht 2007]. By studying here first the common simpler failure and repair, we can cross-check and validate the simulator's correctness and fidelity, as well as compare our results to prior work (e.g. [Angus 1988]), before introducing the complexity of more realistic hardware models.

A *drive* is a device that provides persistent storage of data bytes. Drives are typically the most failure prone components, due to containing (at least for magnetic disk technology) many moving parts that can wear out. For the first part of this paper on combinatorial only-drives-fail reasoning, we simplify by assuming that drives are identical in both storage capacity and MTTF. Also, unless otherwise stated, we assume for experiments that drives store 4 TB (2^{42} bytes) each and have MTTF of 3 years (1095 days). In the second part of the paper, where we describe the CQSIM-R simulator, we allow potentially each component (drive or non-drive) to have its own failure and repair characteristcs (see Figure 1). For example, magnetic hard disks (HDDs) have very different lifetime and failure characteristics than do solid state drives (SSDs), and these components can be modeled separately.

Storage systems are built from more than just drives. We also model *hosts*, which are computers that may mediate access to drives. *Racks* are elements that contain hosts and drives and (often) a network switch that connects hosts to each other and to the outside network. Finally, *datacenters* contain one or more racks. The *Earth* contains all datacenters (at least at time of this writing).

All these elements have their own MTTFs and MTTRs. Beyond that, however, they have dependencies, in that we assume that a device functions at a time t if and only if (a) it has either never failed or it has been repaired since its last failure, and (b) all components upon which it depends are functioning at t. Thus, we distinguish between *failed* and *nonfunctioning*. Failure implies nonfunctioning, but an element may not function even if it has not individually failed.

In the experiments reported here we assume a dependency *tree*; that is, each drive depends upon a host, each host on a rack, each rack on a datacenter, and each datacenter on the Earth. In future work we plan on generalizing to arbitrary acyclic dependency graphs.

Thus, for example, if a host fails, it may cause one or more drives not to function; if a datacenter fails, it causes all elements transitively depending upon it not to function. Components that fail go through their normal repair periods, whether they failed directly or were caused not to function by an ancestor in the dependency hierarchy.

Note that if a host or other non-drive component fails, it may be possible that once it comes back up the data on a dependent drive may be intact. In this study, if the component recovers, the data are intact, and the recovery happens over a short enough time interval, then this is not modeled as a failure at all. If, on the other hand, the data are unavailable long enough for the storage system to conclude the drive is permanently lost and therefore initiate a repair process, we treat it as an unrecoverable failure (as does the storage system) whether or not the data are intact. In general, there are many classes of temporary, partial, or recoverable error states, such as sector errors [Bairavasundaram et al. 2007; Schroeder et al. 2010], that are not modeled here. The enhancement of this paper's framework to handle these classes of error states is left for future work.

Figure 1 shows a representative hardware model that is used within the CQSIM-R simulator. It depicts the dependencies among components as well as each component's MTTF and MTTR in days.

3.2. Files, Chunks, Strength, and Capacity

As systems must store larger amounts of data and use larger numbers of fallible hardware components to do so, redundancy based reliability techniques have become essential. In all such systems, extra space is used to store more than one byte per content byte. Typically, files are stored as collections of fixed-size segments; without loss of generality, in this paper we will define a *file* as one 64 MB segment, understanding that larger files can be constructed out of these. Each file is stored as a collection of n *chunks*, with no two chunks from the same file stored on the same drive. We assume $n = p + s$, where $p > 0$ is the number of *primary* chunks and $s \geq 0$ is the number of *secondary* chunks. In all schemes, data is permanently lost if and only if any subset of $s + 1$ chunks or more are lost at the same time, otherwise data can be reconstructed. We will refer to the choice of (p, s) for a storage system as its *strength*. The *overhead* of a storage system, defined as s/p, is one measure of the cost of the redundancy scheme. The *byte capacity* C_b of a storage system containing d total drives and strength (p, s) is given by $C_b = dBp/(p + s)$, where B is the size of a single drive in bytes (2^{42} unless otherwise stated). This is the number of content bytes that can be stored in the system, given that for each content byte we must also store s/p overhead bytes. Given our assumption of fixed file size, we can also define the *file capacity*, C_f, of the system as $C_f = C_b/s_f$, where s_f is the fixed file size (assumed 2^{26}) here. Storage system reliability varies with capacity, so when comparing different systems, we wish to keep capacity the same, even though two having the same capacity may use very different numbers of drives.

3.3. Redundancy Types

In this paper, we study the reliability of two types of redundancy, *replication* and *error correction*. Replication based systems, such as HDFS [Borthakur 2007] or RAID level 1 [Chen et al. 1994], store one or more copies of content data in secondary chunks. That is, each chunk is a replica of the entire file or object. By default, HDFS's triple

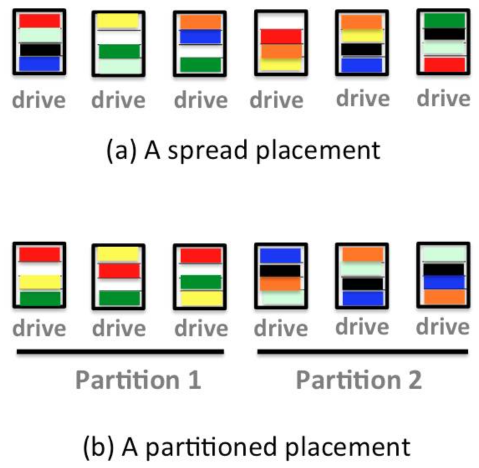


Fig. 2. Illustrating two categories of placement strategy. Individual files are represented by different colors.

replication has strength $(1, 2)$ and therefore overhead 2, while RAID 1's mirroring has strength $(1, 1)$ and overhead 1.

In error correcting systems, such as the Quantcast File System [Ovsiannikov et al. 2013] or RAID level 6, individual chunks do not store copies of the entire file or object; instead each chunk is of size $1/p$ of the original. These schemes store computed codes, such as parity or erasure codes, in the secondary chunks. These enable reconstructing lost data. QFS's default strength is $(6, 3)$ with overhead 0.5, while RAID 6 can have $(p, 2)$ for different values of p, with overhead $2/p$.

In this paper, we do not analyze the design details of a storage system design beyond its strength, overhead, and placement strategy. We wish to evaluate the reliability of storage systems of given strength in order to allow stakeholders to match reliability requirements to costs as indicated by overhead. We do not analyze here other performance metrics, such as read/write performance and bandwidth constraints.

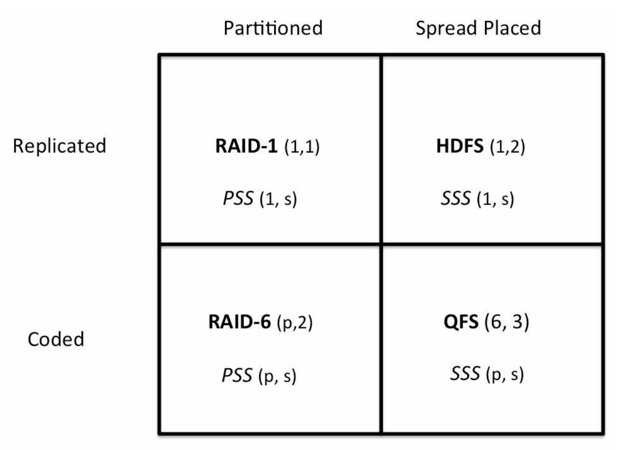
3.4. Chunk Placement Strategy

Assigning chunks to particular drives is termed *placement*. In this paper, we assume that all placement strategies guarantee no two chunks of the same file are assigned to the same drive. There are multiple factors that can dictate placement strategy, including reliability, data locality to computation [Borthakur 2007], i/o performance [Chen et al. 1994], and network load balancing. Here we study two widely used categories of placement strategy, partitioned and spread, as well as selected other placement strategies from the literature, and their impact on storage system reliability.

For a storage system with strength (p, s), a *partitioned placement* strategy partitions the set of all d drives into $d/(p + s)$ disjoint subsets ("partitions") and obeys the constraint that *for any file f, all chunks of f are placed on drives within a single partition.* This definition implies that d is a multiple of $(p + s)$, given our assumption that all chunks of a file are placed on distinct drives.

By contrast, a storage system with strength (p, s) has a *spread placement* strategy if each file's chunks are placed on a random set of drives. Such placements can overlap and do not conform to any partition layout.

Figure 2 illustrates the difference between the two categories. Each colored rectangle represents a file chunk, with chunks of the same color from the same file. Part (a) shows that spread placement mixes chunks from many files on each drive and each subset



	Partitioned	Spread Placed
Replicated	**RAID-1** (1,1) *PSS* (1, s)	**HDFS** (1,2) *SSS* (1, s)
Coded	**RAID-6** (p,2) *PSS* (p, s)	**QFS** (6, 3) *SSS* (p, s)

Fig. 3. Examples of combinations of redundancy type and placement strategy. SSS and PSS are abstract models studied in this paper. Strengths shown in parentheses.

of the d drives can have chunks from one or more files. In this example, each of the 15 pairs of drives has two chunks from some single file; thus, for example, if any two drives fail, at least one file will be lost from a $(2, 1)$ scheme. Part (b) shows a partitioned strategy with two partitions. For every file, either all its chunks are in Partition 1 or all are in Parition 2. Only 6 of the 15 drive pairs have two chunks from any single file. Thus, a partitioned $(2, 1)$ scheme can only fail if two drives from a single partition fail, which is 2.5 times (15/6) less likely than the example in part (a).

There are commercial examples of all four combinations of placement strategy and redundancy type, as shown in Figure 3. (Strengths shown for HDFS and QFS are default settings.) The Ceph storage system [Weil et al. 2006a] can be configured via its CRUSH Map [Weil et al. 2006b] into any of the four cells of the matrix, as well as a wide variety of points on the spectrum between fully partitioned and fully spread.

Other placement strategies. Spread and partitioned placement are two ends of a spectrum. There are many other schemes falling between these two in reliability that are known and used today. Cidon et al [Cidon et al. 2013] define *copyset replication*, which is based on defining z different partition structures over the set of drives such that there is at most one drive in common between partitions of different partition structures. While less reliable than partitioned placement, this allows trading off reliability for increased *scatter width*, leading to more efficient access. Hadoop systems [Borthakur 2007] often use *rack-aware placement*, in which one file replica is placed on a different rack (to mitigate rack level failures) from all other replicas, which are concentrated in a single rack for efficient access. Below, we will also consider a different rack aware scheme, called *Spread-OnePerReack (Spread-1PR)* in which all shares of a given file are placed on different racks. In *scatter-limited spread placement*, attributed by Cidon et al [Cidon et al. 2013] to Facebook's HDFS installation, each file whose primary copy is on drive i has its replicas placed within drives $i + 1 \ldots i + z$ for some parameter z. This improves reliability over spread placement while still maintaining some access efficiency through enforced scatter width.

3.5. Abstract Models PSS and SSS

In what follows, we abstract from design details of extant systems and focus only on strength and placement strategy. The *Partitioned Storage System with strength* (p, s), PSS(p,s), refers to a (modeled) system that uses partitioned placement and has strength (p, s). The *Spread Storage System with strength* (p, s), SSS(p,s), refers to a modeled system using completely random spread placement and having strength (p, s). For SSS, each file's chunks are placed on randomly selected drives. For PSS, the chunks are placed within a single partition so that the partitions are all equally full. For both, distinct chunks are placed on distinct drives.

For both PSS and SSS, we require $p > 0, s \geq 0$. Note that if $p = 1$, then the secondary shares must each individually contain the information to reconstruct the file, so from our abstract viewpoint, these model replicated storage schemes. The rest of this paper will show how to evaluate and compare reliability of storage systems, in most cases using these two modeled storage system classes as examples. The results apply directly (within the limitations of the assumed abstraction level, of course) to all real world instances having these properties, including those shown in Figure 3 among others. The tools and methods apply equally well to other placement strategies, such as those described in Section 3.4.

Neither PSS nor SSS models systems having distinguished metadata nodes as are present in some real world systems, such as HDFS's namenode, or the QFS meta-server node. These are usually handled specially to avoid their becoming a bottleneck for performance and single point of failure for reliability. We focus in this paper only on the reliability with respect to the other nodes and drives of the systems. Note that some storage systems, such as Ceph [Weil et al. 2006a], do not require metadata servers. However, we will show below how metadata servers can easily be handled using a sectioning principle (see Section 6.3).

4. RELIABILITY MEASURES

In the analyses and experiments to follow, the independent control variables governing storage system performance and reliability are size, strength, and choice of placement scheme. Size is indicated either by d, the total number of drives, or else by C_f, the total capacity (related to d through the formula above). Strength is indicated by (p, s) as above. Choice of placement strategy will be indicated as well.

A file is *lost* if at least $s + 1$ of its chunks are on nonfunctioning drives at the same time. We assume this condition persists long enough for the storage system's self-monitoring to detect the loss and mark the drive(s) permanently "out". Thus, even if the drive is intact and is restored to functioning by an ancestor being repaired, the storage system treats the data as permanently lost. Of course, storage systems can tolerate brief drive outages, due to whatever cause, without marking the drives out, as long as the outage is not too long. We ignore this case and treat it as not having happened.

A *loss event* is a component failure that results in at least one file being lost that was not lost immediately before the event. The tools described in this paper report reliability information in terms of two metrics, MTBLE and MLR.

Mean time between loss events (MTBLE). A large system will not stop after a single failure. Instead, our picture is one of a large collection of components constantly failing and being repaired. When a loss event occurs, an extraordinary action is taken to restore or replace the lost file, such as retrieving it from archival storage, recomputing it from other information, or simply replacing it with new content and compensating the stakeholder. Since these restoration events are highly costly almost independently of how large the loss, a key metric to storage providers is the frequency of such events, or its reciprocal, the *mean time between loss events (MTBLE)*. Note that this is a bit different from the mean time between failures (MTBF) [Angus 1988] or to first failure (MTTFF). These are usually analytically evaluated by assuming that once a failure occurs, the system stops until it is repaired, at which time the "clock" starts again. By contrast, MTBLE is simply the interval between loss events in a constantly running system so, while the first loss is being repaired or compensated and the underlying components are being repaired, the "clock" is running until the next loss event. This better reflects experience in a large data center. MTBLE contributes directly to the steady-state cost of running a large storage system, so we focus on it.

If desired, one can convert MTBLE into a probability of non-loss over a given time interval; for example, $P_{365} = 1-(1-1/\text{MTBLE})^{365}$, where MTBLE is measured in days and P_{365} is the probability of non-loss in one year. P_{365} may be quoted as a number of "nines" of reliability, such as five nines indicating $P_{365} \geq 0.99999$.

Mean Loss Rate (MLR). Another metric of interest is the average rate at which a storage system loses content bytes, expressed as a fraction of capacity per unit time. This is the *mean loss rate (MLR)*. This is interesting in situations where the regeneration cost of lost data varies with its amount, or in which data has a low per-byte value, such as when losing a few frames out of a video has no significant impact on user experience. Here, a "loss" is again when the data are irrecoverable due to the system losing too much of its representation, such as losing all replicas. To convert from object losses to capacity, we multiply object loss rate by size of object (bytes) and divide by total byte capacity of the system (capacity not including redundancy overhead).

5. THE CORE MODEL

In this Section, we develop notation that is the common core of both analytical tools and simulation models developed in subsequent Sections.

Allowed sp1-sets. Let $S(p, s, d)$ be a storage system with strength (p, s) and d total drives. When clear from context, we may omit the parenthesized parameters for brevity. S's capacity is $dp/(p + s)$ drives. We define an *sp1-set* (of drives) as any set of $s+1$ drives. Now, S will lay out the $p+s$ shares of each file according to some placement strategy. An sp1-set is in the *allowed set of S, A_S*, iff it is possible for S to place $s + 1$ of some single file's shares on its drives. For example, in SSS *all* sp1-sets are allowed; for PSS, however, shares of a single file will never be placed on an sp1-set that is split between two partitions, so such sp1-sets are not in A_{PSS}.

Modeling assumption. For the analysis below, we will assume that *storage systems place file shares uniformly randomly within their allowed sets*. This is exactly true of spread, limited-spread, and rack-aware schemes, and an excellent approximation for partitioned, copyset, and related schemes.

Probability of Occupation. In operation, once $\text{S}(p, s, d)$ is filled with (placed) files, we define an sp1-set α to be *occupied* iff there exists at least one file having $s + 1$ of its shares placed on the drives of α. In general, a placement need not occupy all allowed sp1-sets. As a consequence of the modeling assumption above, each allowed sp1-set has equal probability of being occupied, so we define this *probability of occupation (PO$_S$)* as the probability that an allowed sp1-set is occupied by a placement of S. Then

$$\text{PO}_S = 1 - (1 - \binom{p+s}{s+1} /|A_S|)^{C_f} \tag{1}$$

where C_f is the file capacity, as defined in Section 3.2. *Proof:* Each time S stores a file, it chooses $p + s$ drives to store the chunks. This means that there are $\binom{p+s}{s+1}$ out of the total $|A_S|$ sp1-sets that are occupied by that file. The probability that a randomly chosen sp1-set, x, is not occupied by that file is $(1 - \binom{p+s}{s+1} /|A_S|)$. Therefore, since the files are placed randomly (by the modeling assumption above), the probability that *none* of the files will be placed so as to occupy x is the complement of that raised to the power C_f. □

Table I. A and PO for selected storage systems.

S	A_S	$S(6,3,1080)$ $z=10$ A	PO
Partitioned (PSS)	$\frac{d}{p+s}\binom{p+s}{s+1}$	1.5e4	1.0
Copyset(z)	$z\frac{d}{p+s}\binom{p+s}{s+1}$	1.5e5	1.0
LimitedSpread(z)	$d\binom{z}{s}$	1.3e5	1.0
RackAware(z)	$d(z-1)\binom{d/z}{s}$	2.0e9	0.95
Spread (SSS)	$\binom{d}{s+1}$	5.6e10	0.1

Note that if the storage system is not 100% full of content, we would replace C_f with the number of files actually stored; for simplicity, we will assume the storage system is near 100% full in what follows.

As described below, in *both* simulation and the analytical tools, PO is used to determine whether the failure of a drive represents a loss event, because an sp1-set of failed drives represents a loss iff it is occupied. Table I gives A_S formulae for selected storage systems as well as A and PO values for a particular set of parameter values.

6. ODF RELIABILITY

In this section, we describe analytical formulae for MTBLE and MLR for storage systems when the only components with finite MTTF are the drives themselves. All others never fail. Moreoever, the drives fail independently and at the same rate. We will term this *only drives fail (ODF) reliability*.

This analysis is useful in understanding the scaling behavior of reliability with capacity, but also gives insight into design and cost tradeoffs. Finally, even though any storage system must be implemented within fallible datacenters, it is useful to understand the contribution to overall reliability of the inherent storage system design itself (as opposed to failures of non-drive elements).

6.1. Computing MTBLE Under the ODF Assumption

Under the ODF assumption, a loss event may only occur when a drive fails. Specifically, it must be that (a) a drive fails while at least s other drives have previously failed and have not yet completed repair, and (b) there exists an allowed, occupied sp1-set consisting of the failing drive and s of the other failed drives. Our approach will be to sum over all such cases weighted by their probabilities.

We first define the *general probability of occupation (GPO)* as the probability that a randomly selected sp1-set is occupied. (Recall that PO is defined with respect to allowed sp1-sets.) GPO can be computed as the product of PO and the probability that the sp1-set is allowed:

$$\mathrm{GPO}_S = \frac{\mathrm{PO}_S \cdot |A_S|}{\binom{d}{s+1}} \qquad (2)$$

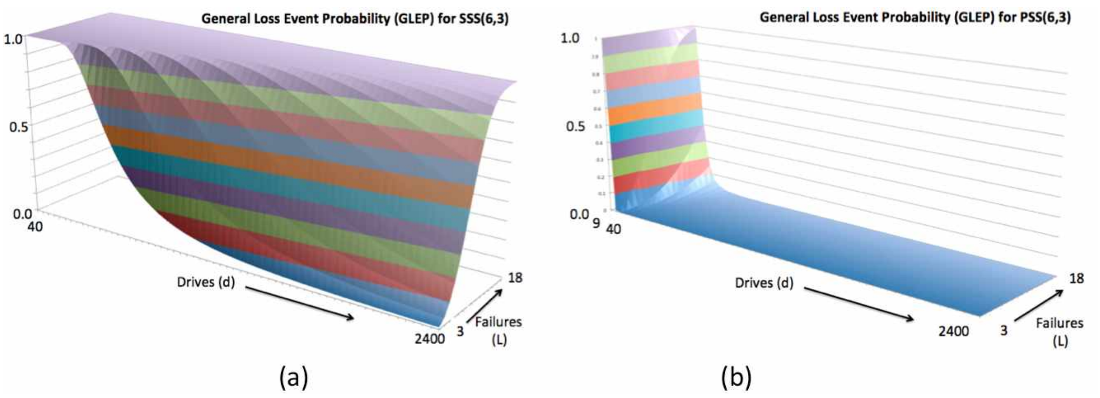


Fig. 4. Graph of $\text{GLEP}_{X(6,3)}$ vs d and L for (a) spread (X =SSS) and (b) partitioned (X =PSS) placement.

Proof sketch: the probability that a randomly chosen sp1-set falls in the allowed set is equal to the ratio of the allowed set to the space of all sp1-sets, which is $|A_S|/\binom{d}{s+1}$ □.

Now, suppose a drive fails (transitions to nonfunctioning) while exactly L other drives are already nonfunctioning (but not yet repaired). This creates $\binom{L}{s}$ sp1-sets of failing drives that could possibly imply a loss event (any s of the nonfunctioning drives plus the current failing drive). The *general loss event probability (GLEP)* function maps L to the probability of a loss event occurring due to the failure of a randomly selected drive while L other drives are already failed:

$$\text{GLEP}_S(L) = 1 - (1 - \text{GPO}_S)^{\binom{L}{s}} \tag{3}$$

Proof: Choose s of the L failed drives. If another drive x fails, the probability that the sp1-set consisting of x plus the s drives is *not* occupied is $(1 - \text{GPO}_S)$. The probability that none of the size-s subsets of the L drives, when x is added, form an occupied sp1-set is obtained by raising that probability to the number of such size-s sets: $\binom{L}{s}$. The probability that at least one *is* occupied, meaning there is a loss event, is the complement of that. □

Figure 4 graphs the value of GLEP vs number of drives d and number of failures L for (a) SSS(6,3) and (b) PSS(6,3). For spread placement (a), note that for any given d, it rises quickly toward 1; for example, at $d = 1080$ it rises to around 0.5 at $L \approx 11$. So in general it does not take many failures to make loss events likely for SSS. By contrast, for partitioned placement (b), the similar surface for PSS(6,3) starts at 0.00000027 at 1080 drives and 3 failures and does not rise to 0.5 until $L \approx 250$, which is too far out to show on this diagram. This explains why PSS's MTBLE is so much higher than SSS's: ultimately, even though PO_{PSS} is 1.0, GPO_{PSS} is tiny, due to A_{PSS} being very small (as compared to A_{SSS}, for example).

We compute MTBLE of S as the reciprocal of the loss event frequency, which can be computed as the following sum capturing the discussion above, where the sum is over all possible numbers of concurrent drive failures, weighting each case by its probability:

Theorem MTBLE. If f is the drive failure rate and r the repair time,

$$(\mathrm{MTBLE}_S)^{-1} = \sum_{L=s}^{d-p} f\, d \binom{d-1}{L} (rf)^L (1-rf)^{d-L-1} \mathrm{GLEP}_S(L) \tag{4}$$

Proof sketch: the overall frequency of loss events is the sum of the frequencies of loss events occurring when L drives are already failed for each L from s up to $d - p$. For each L, the frequency is fd times the probability of exactly L other drives being failed times the probability that such a failure results in a loss event (GLEP(L)).□

The ODF-MTBLE tool embodies the above formula.

6.2. Computing MLR Under the ODF Assumption

The mean loss rate (expressed as a fraction of the total size) of the entire storage system is the same as the loss rate for each file, by symmetry of the underlying ODF hardware model. Any subset of $p+s$ drives experiences a failure of one of its sp1-sets at the same average rate as any other subset of $p+s$ drives. An individual file experiences 100% loss at each loss event involving one of its sp1-sets. The MTBLE of a single file is the same as that of a storage system of the same strength having $d = p + s$, because all files in the storage system are placed on the same set of drives. This proves

Theorem MLR.

$$\mathrm{MLR}_{S(p,s,d)} = (\mathrm{MTBLE}_{S(p,s,p+s)})^{-1} \tag{5}$$

where $d \geq p + s$ and MLR is expressed in units of total capacity per time unit. □

The ODF-MLR tool embodies the above formula.

An interesting corollary of this is obtained by simplifying the above expression for MTBLE using $d = p + s$:

Corollary 1. MLR depends only on p, s, f, r and not on storage system type (A_S) or capacity (d).

Proof: For any S, $A_{S(p,s,p+s)} = \binom{p+s}{s+1}$. □

Even though storage system type and capacity strongly affect MTBLE, the average rate of losing content (MLR) is unaffected by them. Note that this is true only for ODF models where all drives fail independently at the same rate, however.

6.3. ODF Reasoning About Sectioned Storage Systems

Not all storage systems implement one scheme across all of its drives. Large data centers often incorporate multiple heterogeneous storage arrays, appliances, and disks. Each may have its own redundancy scheme.

A *sectioned storage system* is one such that the d drives are divided into disjoint subsets (called *sections*), each having its own independent choice of partitioned, spread, or some other redundancy scheme. No single file has chunks in more than one section. Sections need not all use the same strengths. In fact, a section could itself be a sectioned storage system or some other type, allowing us to build recursive storage structures out of heterogeneous parts.

The reliability of a sectioned system K comprising sections K_i for $i = 0 \ldots u$ obeys the

Theorem (Sectioning Principle):

$$(\text{MTBLE}_K)^{-1} = \sum_{i=0}^{u} (\text{MTBLE}_{K_i})^{-1} \tag{6}$$

assuming drives fail independently, as in ODF reasoning.

Proof: Loss events occur concurrently and independently in all the sections and so the loss event rates, which are reciprocals of the MBTLEs of the sections, add. □

Corollary 2.

$$(\text{MTBLE}_{\text{PSS}(p,s,d)})^{-1} = \frac{d}{p+s} (\text{MTBLE}_{\text{PSS}(p,s,p+s)})^{-1} \tag{7}$$

Proof: This follows from the above formula for sectioned systems as well as our previous formula for MTBLE by setting $d = p + s$ to compute MTBLE of a single partition. Note that GLEP for a single partition is 1.0, because every sp1-set of a single partition must be occupied.□

The formula for deducing MLR of K is similar, but must be weighted in proportion to section capacities:

$$\text{MLR}_K = (1/C) \sum_{i=0}^{u} C_i \, \text{MLR}_{K_i} \tag{8}$$

where $C = \sum_{i=0}^{u} C_i$ is the capacity of K in bytes, and C_i are the capacities of the sections in bytes. *Proof:* In a time τ, section i loses $\tau C_i \text{MLR}_{K_i}$ bytes. Adding these losses up over all sections, we get the total bytes lost in time τ. Dividing that by τ, we prove the above formula, since the total byte loss rate of the entire system is C MLR_K. □

The sectioning principle is useful in several ways. For example, instead of using spread placement across all drives, one might create sections and use spread placement within each. This will increase reliability much faster than linearly in the number of sections, because the reliability growth law is so steep. For example, a sectioned system consisting of 100 sections, each instantiating SSS(6,3) at 108 drives, would have MTBLE $= 723$ days, whereas a single SSS(6,3) at 10800 drives has MTBLE $= 2$ days.

Sectioning is also useful when a datacenter contains heterogeneous storage appliances that are to be combined into a larger system. Each appliance's reliability can be analyzed independently and then combined using the sectioning principle above.

Metadata nodes. Some systems, such as HDFS, QFS, and Ceph's CephFS filesystem, have distinguished nodes that manage metadata. Metadata records information *about* the main data store and its stored entities. These nodes are often treated distinctly from other nodes in the system, as they are critical to functioning and to performance. So, for example, a system using erasure coding for its main data store might use mirroring ((1,1)-replication) for its metadata. Using the Sectioning Principle, the reliability of these systems can be handled easily by our ODF framework; each storage system, main store and metadata store, is analyzed according to our principles, and the results are then combined using the Sectioning Principle formula above. See also Section 8.2.2 for discussion of applying CQSim-R to evaluating reliability of real-world systems, where we include reasoning about the non-drive portions of metadata services.

Tiered Storage Systems. Still other systems may store objects according to *tiers*: frequently accessed objects are “hot” and kept in low latency, high throughput storage devices like SSDs. Objects that have “cooled”, so are less frequently accessed, are mi-

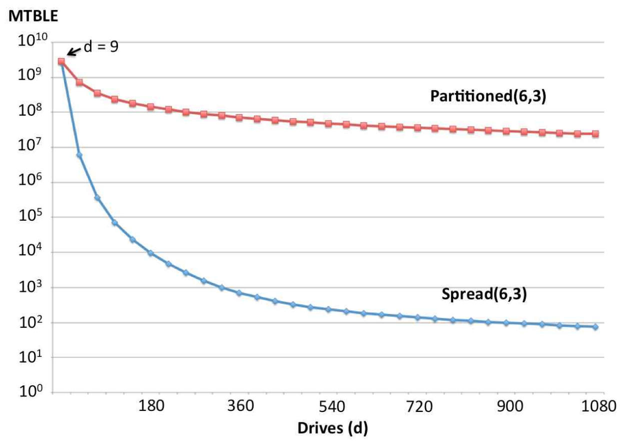


Fig. 5. The Spread Catastrophe at $(6, 3)$: MTBLE of SSS(6,3) vs PSS(6,3). Note log vertical scale.

grated to “warm” storage, where they are hosted on lower performing, but less expensive per TB, HDDs. Again, the Sectioning Principle can be used to combine the reliability results from the hot and warm tiers into a single overall measure of the reliability of the whole system. Any number of tiers can be accommodated in this way, such as adding a “cold” tier for archiving objects as they become even less frequently used.

6.4. ODF Tool Design

Each formula can be directly translated into code using double precision floating point. However, due to the wide dynamic range of intermediate values, the CQSIM-R analytical tools, ODF-MTBLE and ODF-MLR, use a custom floating point representation that accommodates exponents in the range $[-2^{31}, 2^{31}-1]$, which is much larger than the range of IEEE double precision exponents. It also normalizes the mantissas to maximize retained significant digits. The multiplications and divisions within the formulae are reordered and renormalized to keep intermediate results' exponents as small as possible. Using these techniques, ODF-MTBLE has been used to compute MTBLE for large parameter values, such as $d > 100,000,000$, $p > 20$, and $s > 20$.

The tools are designed for extensibility to accommodate new storage systems, where the extension is to redefine the *allowedSets* method. That is, each storage system is characterized by the size of its allowed sets subspace and a method computing the size from p, s, d, and any other parameters specific to the storage system, such as number of racks, scatter width, etc.

7. RESULTS I: ODF RELIABILITY SCALING

We can use the ODF computational tools derived above to compare the reliability of different storage systems at any chosen strength and number of drives. Our first result is verification of the intuition that loss events under spread placement (represented here by SSS) are much more frequent than under partitioned placement (represented here by PSS).

As an illustration, Figure 5 graphs MTBLE (log scale) vs number of drives for strength $(6, 3)$, comparing spread (SSS) to partitioned (PSS). As expected, at 9 drives $(= p + s)$, the two have the same MTBLE of 2.9 billion days. However, while PSS(6,3)'s MTBLE drops inverse linearly, SSS(6,3)'s MTBLE drops much faster. At 1080 drives, SSS(6,3)'s MTBLE is only 76 days, while that of PSS(6,3) is still 24 million days. Note

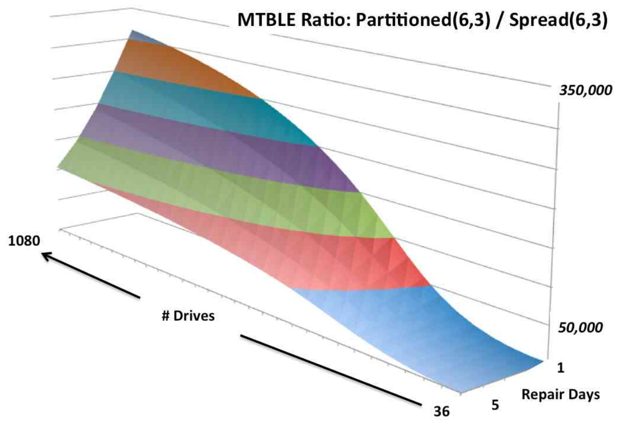


Fig. 6. MTBLE Ratio (6,3) vs. #Drives and Repair

that both schemes have MLR $= 3.48 \times 10^{-10}$ (files per day per file of capacity) at all values of d. For example, assuming $d = 1080$ and 4 TB drives, this would imply a loss of 4,404,098 bytes/day. Thus, whereas one expects to wait over 65,753 years between losses using PSS, one can expect to lose around 335,592,268 bytes once every 76 days when using SSS.

This trend continues into higher d ranges. At 10,800,000 drives ($\approx 40EB$), for example, MTBLE of PSS(6,3) is 2397 days (over 6.5 years), while the MTBLE of SSS(6,3) is 0.0061 days (about 8.8 minutes).

This *spread catastrophe* is present at all strengths where $s > 1$ and is explained ultimately by the sizes of the allowed sp1-sets subspaces, which are large for spread and small for partitioned. These, in turn, dictate the shapes of the corresponding GLEP surfaces. For spread, the GLEP rises to 1.0 quickly as concurrent failures increase, while for partitioned, the GLEP is very low until there are many concurrent failures, which is correspondingly rare. The drop of PSS's MTBLE is inversely proportional to d for all d ranges. SSS's MTBLE, on the other hand, drops early on in proportion to a factor of $1/\binom{d-1}{L}$, for $L \geq s$. Over this early period of relatively low d, this decay is roughly proportional to d^{-s}, so the stronger the scheme (i.e., greater s), the more precipitous the drop-off of SSS's MTBLE. At larger d ranges, the drop slows, approaching proportionality to $1/d$.

As d grows large, the ratio between MTBLE of PSS and that of SSS appears to converge to a constant. That is, after an initial period of rapid decay, the reliability drop with increasing capacity of SSS starts matching that of PSS, albeit at a much lower absolute level. This ratio is different for different strengths. For example, at strength $(1, 2)$, the ratio converges to approximately 64,000 beyond about 10,800 drives. This holds at least out to 108,000,000 drives (432 EB), the maximum checked using ODF-MTBLE. At strength $(6, 3)$, the ratio converges to approximately 393,000 around 5,040 drives. Again, it holds at least out to 108,000,000 drives (ODF-MTBLE took just under 4 minutes on a laptop to compute the latter value). The ratio *seems* to approach Bp/s_f (where drive size (B), file size (s_f), and primary shares count (p) are system parameters defined in Section 3.2), which is what one would expect if all spread files failed independently. (Investigating the generality and limits of this conjecture is future work.)

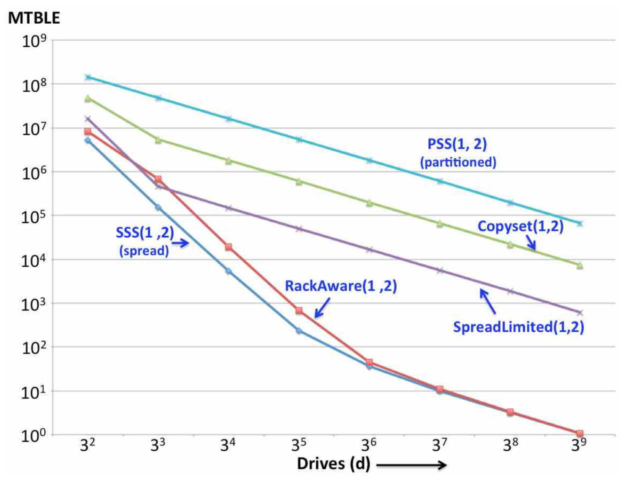


Fig. 7. Five storage systems compared at $(1, 2)$ using ODF tools. Both axes have log scales.

We can also look at the effect of increasing time-to-repair on MTBLE and the spread catastrophe. Figure 6 graphs the ratio of MTBLE of PSS to SSS at $(6, 3)$ vs both number of drives and days to repair for fixed drive MTTF. Both PSS and SSS have their MTBLE drop with increasing time to repair. Apparently, it drops faster for PSS, as the ratio of PSS to SSS drops with increasing time to repair.

Comparing Storage Systems to Each Other. We can use the ODF tools to compare different storage systems' scaling laws to each other, as shown in Figure 7 for five storage systems at strength $(1, 2)$. PSS, which is inverse-linear in d, appears as a straight line in this log-log plot. Copyset and SpreadLimited are evidently also inverse linear for fixed z values. (These graphs used $z = 9$, except at $d = 9$, where $z = 3$ was used.) SSS drops as d^{-2} when $s = 2$.

We can also use the ODF tools to compare storage systems of different strengths to each other as well, though we must be careful to adjust number of drives to match capacities. For example, we may ask which has greater MTBLE between SSS(6,3) or SSS(12, 3) at capacity of 288 TB (72 drives). To have this capacity, the (6,3) strength would need to employ 108 drives, while the (12, 3) capacity would need only 90. In this case, the MTBLE of SSS(12,3), 149,718 days, is *higher* than that of SSS(6,3), 72,266 days. By contrast, the MTBLE of PSS(12,3), 44 million days, is significantly *lower* than that of PSS(6,3), 240 million days, at that capacity.

8. CQSIM-R: DETAILED RELIABILITY MODELING

When we wish to evaluate storage system reliability *as deployed*, we must consider failures of the other elements making up the deployment. While this could, for each such detailed deployment, be handled analytically as we did in the ODF case, the details get messy and complex. Complexity arises from (a) dependencies among failures, (b) heterogeneous component types, such as solid state storage vs magnetic disk storage that have different failure properties, and (c) behavioral impacts on reliability such as drive reliability varying with drive traffic in addition to time. For these reasons, I approach these more detailed problems using modeling and simulation. Specifically, the *Cloud Quality of Service Simulator for Reliability (*CQSIM-R*)* is an event-based Monte Carlo simulation of modeled software components interacting with modeled hardware components.

Hardware and software configurations are input via a *structure file* that describes their types, parameters, and relationships. This description was introduced in Section 3.1. This file can be built manually for simpler descriptions, but the toolset also provides a structure generator `SimGen(...)` that can be used to generate automatically a hardware file from a few input parameters. `SimGen` was used to generate the runs from which the data below were gathered. It is built around a particular multi-datacenter design schema with parameters governing numbers of drives, hosts, racks, datacenters, and structural information like hosts per rack, etc.

At the heart of CQSIM-R is a relatively simple event based simulator. There is an event queue kept sorted by event time. The inner loop simply removes the earliest event from the queue and provides it to all registered event handlers whose event-masks match the event type. They can manage state and schedule one or more events at some time in the non-past.

8.1. Hardware Element Models

Each hardware element starts in the functioning state and schedules at a random time in the future its first failure event. The model used to generate this time is currently an exponentially distributed random delay based on the MTTF declared in the structure file for that component. When a component receives a failure event, it changes state to failed and notifies all components depending upon it to fail. It then schedules its repair event according to the MTTR in the structure file. When a component repairs, it notifies all of its dependents to return to functioning unless they have some other reason not to, such as being failed themselves or some other component on which they depend being failed.

If our structure file gives infinite MTTFs for all non-drive components, we have only-drives-fail semantics. Thus, we expect that the simulation results should be consistent with the ODF tools described above. This gives us a way to validate both ODF tools and simulator by cross-checking their ODF results. (See Section 10). Only once we introduce finite MTTFs to non-drive components will we see the effects of other real world failure scenarios.

In this paper, we use the simple hardware failure and repair models described above. However, it is straight forward to substitute different failure and repair behaviors [Schroeder and Gibson 2007] into CQSIM-R in order to more faithfully model drive and component behaviors experienced in the real world.

Storage system software models, such as SSS and PSS, run on top of this. Each time a drive transitions to the nonfunctioning state, all storage system models of which it is a part are notified. Their models then decide whether to change state based on their semantics.

8.2. A General Modeling Method

To simulate a storage system, we need to determine whether when any given drive transitions to nonfunctioning it represents a loss event. We also need to be able to track the average rate of file loss. However, since we wish to see the effects of hardware and other dependencies, we cannot use GLEP to compute this, since GLEP's definition assumes independent random drive failures. In this subsection, we describe a generic simulator model design for storage systems that is customized to each storage system through defining two combinatoric methods, *failureSets* and *goodFraction*, as defined below.

A *failure set* is an sp1-set all of whose drives are in the nonfunctioning state. Now, for a storage system S the *Specific Loss Event Probability (SLEP$_S$)* is a function mapping a nonnegative integer to a probability value. When the integer counts the number of allowed failure sets that all contain a given drive δ that has most recently failed, the output will be the probability that the transition is a loss event. From the definition of PO,

$$\mathrm{SLEP}_S(L) = 1 - (1 - \mathrm{PO}_S)^L \tag{9}$$

where we define $0^0 = 1$ here. *Proof:* In order for the failure of δ *not* to be a loss event, the L failure sets must all be unoccupied. The probability of this is the Lth power of the probability of a single one being unoccupied. □

A CQSIM-R (storage system) simulation model is an object supporting two methods, *OnElementFailure* and *OnElementRepair*. The former is invoked each time a drive, δ, on which the instance depends goes to nonfunctioning, and the latter when the transition goes back to functioning. To implement a model, the general approach is to use state to keep track of enough information to be able to calculate the method *failureSets*(δ), which returns the number (denoted L in Equation 9) of allowed failure sets involving δ. OnElementFailure then calls SLEP on that number to get the probability P. If positive (simulated) time has passed since the most recent loss event, then the model generates a random number $x \in [0, 1]$ and, if $x < P$, counts a loss event and updates the time of most recent loss event. Note that by keeping track of last time of loss event, we can avoid overcounting loss events due to a hardware element's (e.g. common host or rack) failure causing multiple simultaneous drive transitions.

The details of computing failureSets depend intimately on the specific storage system. Here are a few examples to illustrate the technique.

— *Spread placement model:* SSS only needs to maintain the count `FCnt` of current failed drives. In this case, failureSets can be computed (independent of δ) as $\binom{\mathtt{FCnt}}{s}$. SSS's OnElementFailure method increments `FCnt`, and OnElementRepair decrements it.
— *Partitioned placement model:* PSS must keep track of a count $\mathtt{FCnt}_i$ for each distinct partition i of the number of failures within the ith partition. OnElementFailure looks up the partition j of δ and, if $\mathtt{FCnt}_j$ is exactly s on entry, then failureSets returns 1; otherwise, failureSets returns 0. OnElementFailure increments $\mathtt{FCnt}_j$, and OnElementRepair must decrement the appropriate partition's $\mathtt{FCnt}_j$.
— *Copyset placement model:* This operates similarly to Paritioned, but maintains a separate set of partition counts for each of the z distinct partition schemes. It then computes failureSets as the *sum* of the failureSets values of the different partition schemes, returning this sum.
— *LimitedSpread placement model:* For this storage system type, failureSets is computed as follows. For each $i \in [0 \ldots z-1]$, it counts the number L_i of nonfunctioning drives among drives $[\delta - i, \delta - i + z - 1]$ and sums $\binom{L_i - 1}{s}$ for all i. FailureSets returns that sum. Neither OnElementFailure nor OnElementRepair maintains separate state from that of the underlying hardware simulator, which keeps track of functioning status of the drives.

To track the MLR statistic, the model assumes a *goodFraction* subroutine, whose implementation is specific to the storage system type. GoodFraction returns the fraction of total capacity that is within nonfailed files at the time it is called; this value will generally be less than 1 if several drives are failed at once. Figure 8 shows OnElementFailure and OnElementRepair excerpts that jointly implement a simple algorithm to track MLR using goodFraction. Essentially, drops in goodFraction are summed and MLR is computed as the average of the drops over simulation time.

Here are some examples of goodFraction calculations.

```
Initially
  gf := 1
  SumOfFailures := 0
  LastFailureTime := -1

OnElementFailure
   x := goodFraction()
   SumOfFailures := SumOfFailures + (gf - x)
   LastFailureTime := now()
   gf := x

OnElementRepair
   gf := goodFraction()

Finally
  MLR := SumOfFailures / LastFailureTime
```

Fig. 8. Pseudo code for computing MLR statistic. Excerpted from the full code for OnElementFailure and OnElementRepair.

— *Spread placement model:*

$$\frac{\sum_{i=0}^{s}\binom{L}{i}\binom{d-L}{p+s-i}}{\binom{d}{p+s}}$$

where L is the current total number of nonfunctioning drives. The estimate is computed as the number of nonfailed allowed file placements divided by the total number of allowed file placements.

— *Partitioned placement model:*

$$\frac{\mid \{i \mid L_i \leq s\} \mid}{(\frac{d}{p+s})}$$

where L_i is the number of failed drives in the ith partition.

— *Copyset placement model:*

$$\frac{\mid \{i, j \mid L_{ij} \leq s\} \mid}{z(\frac{d}{p+s})}$$

where L_{ij} is the number of failed drives in the jth partition of the ith partition scheme. z is the number of partition schemes.

8.2.1. Hybrid Storage System Models. Sectioned systems can be modeled elegantly by first extending models of component subsystems, such as PSS or SSS models, to schedule a zero-delay event each time it detects its own loss event; a model of the higher level sectioned system receives this event and counts its own loss event and file statistics accordingly. In this way, assembly of arbitrary sectioned systems is straight forward, given models of all the primitive subcomponents.

8.2.2. Modeling Metadata Subsystems and Tiered Storage. Metadata servers and nodes, as discussed previously, often form a distinguished subsystem whose reliability design is distinct from that of the main store. CQSIM-R can handle this easily as a hybrid system by creating separate simulations of the main store and the metadata subsystem and then combining as discussed above.

Similarly, tiered storage can be handled in this way as well. For example, a hot SSD tier using double redundancy (PSS(1,1)) can be backed by a cheaper and larger warm tier using HDDs protected by erasure coding (SSS(6,3)). The overall reliability of the system can be calculated as discussed above.

8.2.3. Mixing Objects Stored with Different Schemes. Some storage systems allow each object to be stored using a different redundancy scheme. For example, one might have objects stored with (6,3) erasure coding intermixed on the same hardware drives with other objects stored using (1,2) triple replication. This would be very complex to model analytically, because failures are not independent between the different schemes, since they share the underlying storage drives as common failure modes. However, CQSIM-R allows us to combine the modeling concepts above, with each scheme modeled by its own PO, SLEP, FailureSets, and GoodFraction parameters. The simulation model would simply maintain each different schemes' FailureSets and GoodFraction calculations concurrently as the simulation evolves. A failure occurs if and only if at least one of the modeled schemes encounters a failure at a given time. Of course, this depends upon knowing how much of the total capacity is used by the files of each scheme.

8.3. Simulator Performance

The time complexity of CQSIM-R is $O(TV \lg V)$, where T is the number of simulated days and V is the number of components. This result assumes that each component generates on average a constant number of events per day. Constant rate failure models, such as those studied here, satisfy this, as do many common stateful failure models. The result also assumes that the goodFraction and failureSets methods execute in constant time. While the combinatorial definitions given may not appear so, PSS, SSS, and the other storage systems examined to date can all be implemented to be $O(1)$.

From observation, a good rule of thumb is that the current CQSIM-R implementation can process at least 1 trillion component-days in about an hour on a modern laptop running a small number of concurrent storage system models. Thus, a 1000-component model can be simulated for about 1 billion days in an hour, while a 1,000,000-component model can be simulated for about 1 million days in similar time. The largest study I have run handled 2 million components simulated for 1,000,000 days in 4590 seconds running two storage system models concurrently.

Finally, note that for most useful hardware models, this simulation is an *embarrassingly parallel* problem in T: if one wishes to simulate $10T$ days, one can run a T-day simulation on each of 10 machines concurrently and then average the results, weighted according to the sample numbers reported by each simulation. Care must be exercised, however, when parallelizing a simulation involving hardware models with long "memory" behavior.

On the other hand, it is not simple to parallelize efficiently for large V, as the interprocessor communications needed will be significant.

8.4. Statistical Confidence In Results

To use the results of a Monte Carlo simulation, one must address the problem of how close the reported results are to the underlying true values. Confidence increases as the number of samples used in the reported average increases. In CQSIM-R, we adopt Guerriero's result [Guerriero 2012]: one can expect with 95% confidence that the true value lies within $[m\sqrt{n-1}/(\sqrt{n-1}+1.96) \dots m\sqrt{n-1}/(\sqrt{n-1}-1.96)]$, where m is the sample mean and n is the number of samples. See below for an empirical validation of this heuristic.

If one is seeking a particular set of k metric results from the simulation, CQSIM-R can be made to keep extending the simulation duration until the number of samples of each statistic observed is large enough that the width of its Guerriero interval is below a target percentage, α, of its measured value. One would then have $(0.95)^k$ confidence that all k values were within $\alpha\%$ of true values.

9. RESULTS II: IMPACTS OF DISTRIBUTION AND PLACEMENT STRATEGIES

We now illustrate the use of CQSIM-R by examining the effects of non-drive component failures. Such failures have the effect of introducing dependence and correlation among failures that lead to effects not present in the ODF results. This section has two purposes. One is to illustrate information that can be found using CQSIM-R that is not available just using ODF reasoning tools and techniques. However, it is also to further demonstrate the reliability effects of the spread strategy (and related) compared to the partitioned strategy. This is not to deprecate the spread strategy, as it may have benefits such as in reducing network traffic and speeding up i/o requests. However, if one chooses it, one should be fully aware of its reliablity limitations.

Modeling Datacenter Failures. We look first at distribution across datacenters and how that affects reliability. Datacenters provide a coarse level of distribution, with many drives in each one. Below, we will discuss finer grained distribution effects. We use `GenSim` to generate PSS and SSS instances at numbers of drives from 72 to 1080 in steps of 72 crossed with models having from 1 up to 6 datacenters. When we have more than one datacenter, `GenSim` distributes the drives as evenly as possible across them, with no two drives of a single PSS partition lying in the same datacenter when possible.

Figure 9 graphs MTBLE of PSS(1,2) vs both number of drives and number of datacenters. The surface has an interesting structure. Of course, when increasing number of drives for a fixed number of datacenters, we see the inverse-linear decay expected of PSS. However, as we increase datacenters, we see first modest growth, then enormous growth. In one datacenter, since PSS(1,2) abstractly has very high MTBLE, it is the MTTF of the datacenter itself that dominates. At two datacenters, the MTBLE has increased beyond this, because now partitions will have at most two drives per single datacenter, so datacenter failure will not immediately result in PSS data loss. It is, however, more vulnerable during those (4-day) repair periods, so its MTBLE is not as high as the ODF MTBLE. At three datacenters, each partition is fully distributed, so we now get the ODF MTBLE, reduced slightly because the datacenter failure rate increases the effective drive failure rate slightly. Increasing beyond 3 data centers does not further improve MTBLE. All runs reported in that graph had sample sizes at least 1131, so the 95% confidence interval is $[-5.6\%, +6.2\%]$.

By contrast, Figure 10 graphs the same thing for SSS(1,2). It is radically different. First, its highest point is far below the lowest point of the PSS graph in Figure 9. But even more interestingly, *SSS is only disadvantaged by distribution*. That is, the more datacenters chunks/drives are spread to, the worse its MTBLE becomes. The explanation is straight forward: because SSS spreads files across nearly all possible combinations of datacenters, it is vulnerable to any combination of failures, whereas PSS is only vulnerable to combinations that concentrate within partitions. Thus, spreading SSS more widely just gives it more ways to experience loss.

Interestingly, even though in the ODF case the MLR of a spread scheme is identical to that of the partitioned scheme with equal strength, this equality no longer holds once we model other failures. For example, with six datacenters, at 1080 drives, the MLR of PSS(1,2) is 4.2e-9, while that of SSS(1,2) is 2.7e-6. Thus, SSS(1,2) loses data 643 times faster (B/day) in this deployment than does PSS(1,2).

Modeling Hosts and Racks. Where datacenters provide coarse-grain distribution, CQSIM-R allows one to model fine-grain distribution as well. In these experiments, MTTF of racks and hosts were set to 10 years (3650 days) and MTTR to 1 day. `GenSim()`

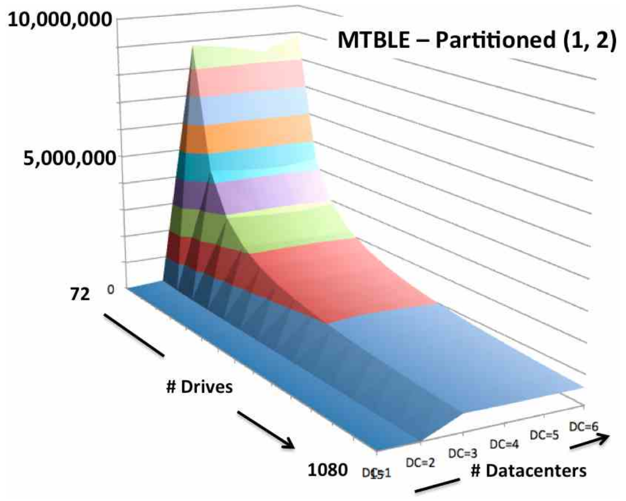


Fig. 9. $\mathrm{MTBLE}_{\mathrm{PSS}}(1,2)$ vs #Datacenters and #Drives

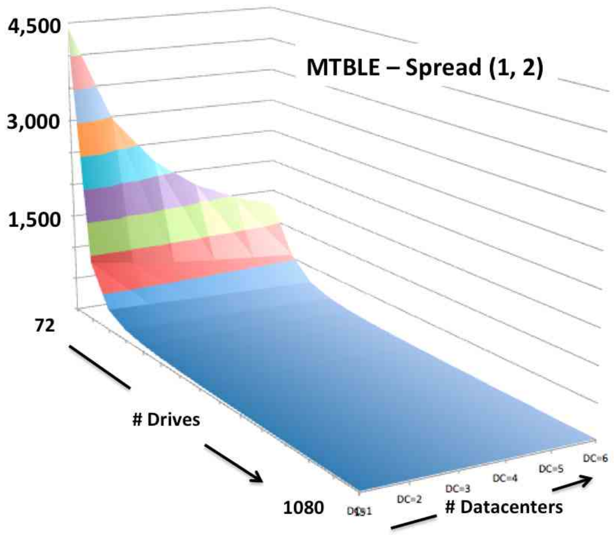


Fig. 10. $\mathrm{MTBLE}_{\mathrm{SSS}}(1,2)$ vs #Datacenters and #Drives

was set to lay out drives 3-to-a-host; that is, each drive must be accessed via a host and each host manages three drives. Each rack contains up to 30 hosts, depending on how large an instance we run. Thus, hosts provide small groupings with the host a common point of failure for the group.

I ran the same set of simulations as in the datacenters-only case. In this way, we can compare the results between modeling only datacenter distribution and modeling datacenters plus hosts plus racks. If we compute the ratio of MTBLE in the datacenters-only run to that of the full model run, we see that SSS(1,2)'s MTBLE drops by a factor of 3, while PSS(1,2)'s MTBLE is only negligibly affected (ratio nearer to 1 than the confidence interval width). This is due to the fact that SSS's random placement can't avoid placing some files' shares on drives depending on the same host, while the PSS layout

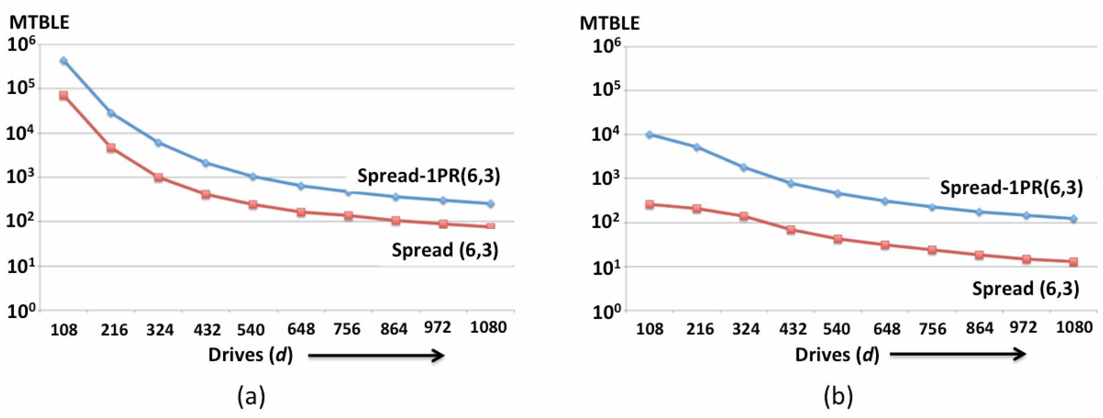


Fig. 11. Spread$(6, 3)$ vs One-per-rack-Spread$(6, 3)$. (a) Graphs generated using ODF assumption. (b) Graphs generated via simulation adding drive, host, and rack failures. (Note log vertical scale.)

will not do so. Thus, single host failures become a significant source of loss events. Similar effects are seen in the MLR metric.

Evaluating Rack Awareness. Another important use of CQSIM-R is to evaluate the impact of design decisions taken to improve reliability, lower cost, or improve performance, once deployed. One example of this would be to use a rack-aware scheme to improve the reliability of spread placement. In another set of experiments, I used the placement scheme *Spread-OnePerRack (Spread-1PR)*, which uses a spread placement strategy, but restricts placements to disallow placing more than one file share on any given rack. Unlike the RackAware strategy studied in Figure 7, Spread-1PR is well defined for any strength (p, s). The purpose of introducing the restrictions is to improve resiliency against rack (and host) failures. Custom failureSets and goodFraction methods enable modeling Spread-1PR within CQSIM-R.

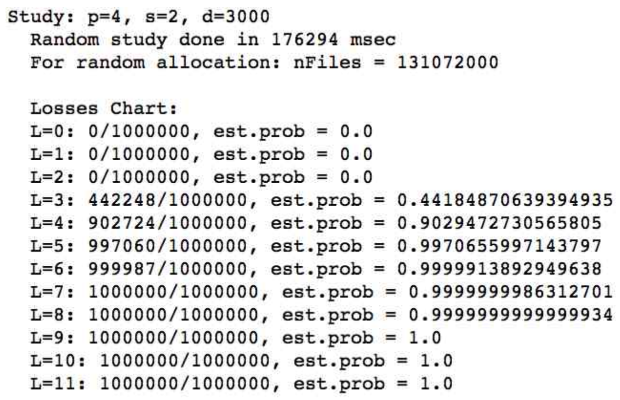
Figure 11(a) graphs MTBLE vs drives for normal Spread and Spread-1PR; the data were obtained from running CQSIM-R on a series of 12-rack deployments, but setting MTTFs for all non-drive components to ∞. Drive MTTFs were 1095 days. As shown in the graphs, the two behave similarly, with Spread-1PR having a small improvement in MTBLE.

Figure 11(b) graphs MTBLE for the two schemes when CQSIM-R adds host and rack failures (3650 day MTTFs) to drive failures. Two things are of note. First, both curves dropped, due to the additional exposure to failures of non-drive components. Second, the two curves are farther apart. Spread-1PR dropped less than Spread, showing that its placment restrictions do indeed help, and the graphs quantify the advantage. (Note log vertical scale.)

10. RESULTS III: TOOL VALIDATIONS

In the absence of large scale and long term empirical testing against data center experience, we must address the soundness of the tools. Can we increase our confidence that the metrics reported are sound, consistent with each other, and representative of reality? This section reports on five separate validations used to build confidence. None of these is a comprehensive proof, but each addresses at least one aspect of the fidelity question and constitutes an initial step in building confidence.

SLEP and GLEP Fidelity. Both the ODF tool ODF-MTBLE and CQSIM-R's models depend critically on the fidelity of the combinatorial models. A basic question is whether GLEP truly models the probability of a loss event. To validate this, at least



```
Study: p=4, s=2, d=3000
  Random study done in 176294 msec
  For random allocation: nFiles = 131072000

  Losses Chart:
  L=0: 0/1000000, est.prob = 0.0
  L=1: 0/1000000, est.prob = 0.0
  L=2: 0/1000000, est.prob = 0.0
  L=3: 442248/1000000, est.prob = 0.44184870639394935
  L=4: 902724/1000000, est.prob = 0.9029472730565805
  L=5: 997060/1000000, est.prob = 0.9970655997143797
  L=6: 999987/1000000, est.prob = 0.9999913892949638
  L=7: 1000000/1000000, est.prob = 0.9999999986312701
  L=8: 1000000/1000000, est.prob = 0.9999999999999934
  L=9: 1000000/1000000, est.prob = 1.0
  L=10: 1000000/1000000, est.prob = 1.0
  L=11: 1000000/1000000, est.prob = 1.0
```

Fig. 12. One of 10 GLEP validation studies.

for spread placement, I built a single-purpose simulation that, for given choice of p, s, d, first generates a random file placement for every one of the $C_f(p, s, d)$ files and then creates a large number of random failure sets of each size, L. It proceeds to count the fraction of file placements that have at least $s + 1$ shares nonfunctioning. It compares this fraction to the corresponding prediction of GLEP for the given $p, s, d,$ and L. Figure 12 shows an example output for $(p = 4, s = 2, d = 3000)$. "est.prob" is the output of the GLEP function.

In all 10 studies of this kind carried out, all GLEP results agreed with the counts to within < 0.001, i.e. 0.1%. Given that each count is a fraction out of 1,000,000 random failure set choices, this is good statistical agreement.

Comparison to Angus' Model. Angus [Angus 1988] gives a formula for mean time between failures for a redundant system having $p + s$ devices that can tolerate at most s failures. The formula does not handle MTBLE directly, nor does it handle systems having a (p, s) redundancy scheme but more than $p + s$ devices. We can, however, use it to cross check the results of CQSim-R and ODF-MTBLE in special cases where $d = p + s$. In these cases, we expect the difference between MTBLE and MTBF to be exactly the expected repair time. Since all failed drives concurrently repair, we can expect that the repair time for the system is MTTR$/L$ under reasonable special case assumptions.

To check for agreement, one selects ($p, s, d,$MTTF, MTTR) and then calls ODF-MTBLE, the Angus model, and runs CQSim-R in ODF mode and compares results. I have studied $(p = 1, s = 2, d = 3)$, $(p = 6, s = 3, d = 9)$, and $(p = 8, s = 1, d = 9)$ for MTTFs in $\{1, 2, 10, 20, 100, 200, 1000\}$ and MTTRs in $\{1, 2\}$. In all cases, all three results agreed, subject to the fact that (a) the ODF-MTBLE result was within the Guerriero confidence interval of the CQSim-R result, and (b) the Angus result differs from the ODF-MTBLE result by exactly the expected repair time. Note that Angus's assumption was that the components stop failing once $s + 1$ components have failed, but this only approximates our assumption of continual failure and repair. In some cases examined, this is not a good approximation. For example, in the $(8, 1, 9, 1, 1)$ case, where MTTF is 1 day, the most common situation is for more than two drives to be failed at any time, because the MTTF is so small and there are so many drives. Here, Angus predicts 0.139 MTBF and ODF-MTBLE predicts 7.111 MTBLE. These are compatible if the repair time is large (6.972). CQSim-R produces MTBLE of 7 which agrees with ODF-MTBLE.

Based on this evidence, it appears that both ODF-MTBLE and CQSim-R agree with Angus's model in the limited cases in which the latter applies.

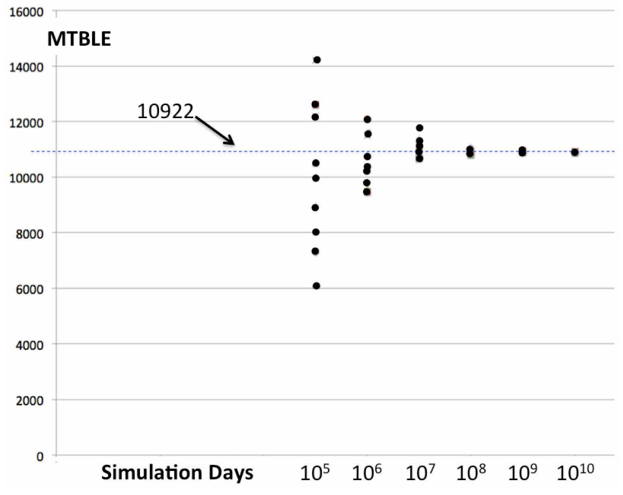


Fig. 13. Test of Guerriero 95% Confidence interval.

Do Simulation and ODF tools agree? If we run CQSIM-R with all MTTFs infinite except those of drives, ("ODF Mode"), and if drive failures and repairs are modeled as constant rate, then its results should be consistent with the predictions of ODF-MTBLE. I have run a set of such tests for selected strengths and verified that more than 95% of the time the value predicted by ODF-MTBLE falls within the Guerriero 95% interval for the CQSIM-R reported value. This verifies consistency of the simulator and the ODF formulae. This validation was performed for both the SSS and PSS models.

MLR Consistency Between Simulation and ODF. CQSIM-R's method of tracking MLR (based upon the goodFraction method) is independent of ODF-MLR's calculation. To check for consistency between the two, I ran CQSIM-R simulations with only drives failing long enough to have a small Guerriero interval and then checked that the reported MLR values matched the ODF-MLR values to within the interval. I did this for several different settings of (p, s, d), including $(6, 3, 36)$, $(6, 3, 1080)$, $(6, 3, 10800)$, $(1, 2, 36)$, $(1, 2, 1080)$, $(1, 2, 10800)$, as well as others. In all cases the values matched.

Confidence Interval. To see whether the simulation results conform to Guerriero's [Guerriero 2012] 95% confidence interval, I ran a particular simulation 60 times; 10 times each with durations 10^5, 10^6, ..., 10^{10} days. The results are plotted in Figure 13. I also checked all 60 points and verified that for 59 of the 60 points, the 10922 line (average of the 10^{10} days runs) lies within the Guerriero 95% confidence interval. Only one point failed to do so. This is perhaps slightly better than expected, as the expectation would be to have seen three such points, but nevertheless, this provides evidence that the Guerriero bounds are a good guide to accuracy of simulation results.

11. DISCUSSION AND CONCLUSION

Limitations and Future Work.

As in any model-based approach to prediction, such as weather forecasting or financial forecasting, there are inevitably ways in which the abstractions and approximations of modeling do not match the real world. These model limitations introduce systematic errors that may cause the actual storage system to deviate from the predictions. Examples of such possible deviations include

— Hardware failures may deviate from the assumed failure distributions, from drive "infant mortality" up to and including a single natural disaster taking out datacenters located too close together;
— Operations personnel may not repair or replace hardware within the time bounds given as inputs to the model;
— The actual implemented storage system may not exactly conform to the assumed ideal behavior (e.g. timely detection of failures and initiation of repairs, speed of repair processes, software or operating system bugs).

The effects of these deviations must be evaluated over time and experience, which allows tuning CQSIM-R's input model parameters.

CQSIM-R's hardware modeling framework is currently limited in that it assumes each element has at most one "parent" on which it depends. In real datacenters, there may be multiple ways things can be connected and dependent. For example, one can have multiple network switches interconnecting things so that if one goes down, the rack or datacenter does not fail. This more complex level of modeling is future work.

Currently, the ODF tools (ODF-MTBLE and ODF-MLR) are tied to the constant failure rate model; it is future work to adapt them to other models, such as the Weibull distribution.

Some storage arrays have redundancy schemes that cannot be classified as a (p, s) scheme. For example, the square array studied by Kao et al [Kao et al. 2013] has a parity drive for each row and column of the square array of content drives and is not characterizable with a single strength (p, s). For example, in a 4x4 array, one can lose all 4 drives of a single row and it will be reconstructed using the column parity drives. However, losing a content drive, its column parity drive, and its row parity drive at the same time will be irrecoverable. Thus, some failure sets have size 3, but others have size 5. Extending the framework of this paper to handle this sort of scheme is future work. However, one could today use the approach of Kao et al for the square array and use our sectioning formula to combine those results with results from other sections of a larger system. Also, one could easily incorporate a simulation model of this array into a larger CQSIM-R simulation, using the techniques discussed above.

Similarly, there are many other detailed array and appliance architectures whose reliability must be modeled. These can then be incorporated via the sectioning principle, in either the ODF tools or in CQSIM-R as described, into larger heterogeneous storage systems.

At the software level, another interesting area of study is in modeling within CQSIM-R *geographically aware placement strategies*, where file shares are purposely placed in different data centers to increase resilience to localized disasters and loss risks. Included within such would be archival [Storer et al. 2008] and cold storage [Patiejunas 2014] systems. While these fall within the scope of our approach in this paper, other issues may complicate the modeling. For example, due to increased bandwidth costs between geographically distant sites, repairs whose traffic must travel between sites may take significantly longer than repairs whose traffic stays within a single data center.

Storage systems are starting to develop ways to specify placement strategies declaratively, such as Ceph's [Weil et al. 2006a] CRUSH map [Weil et al. 2006b]. It is an interesting area of future research to study how to analyze these notations and synthesize CQSimR-compatible models so that the reliability of deployed storage systems can be directly measured using CQSimR.

In a precursor paper to the Ceph paper, [Xin et al. 2003] give ODF calculations of MTTDL for three redundancy schemes, 2-way mirrored, 3-way mirrored, and RAID-5+1. If we use their assumptions about drive MTTF, their calculation is consistent with that produced by ODF-MTBLE in the following sense. They are assuming spread placement, but *limiting the number of placement groups ($p+s$-sets)* by simply only assigning objects to k $(p+s)$-sets, for some k less than the total number possible (d choose $p+s$). It is a straight forward future extension of ODF-MTBLE to accommodate an arbitrary cutoff in this way. The net effect of this cutoff is to increase MTBLE, because there are fewer vulnerable placements. The limit of this operation is, of course, partitioned placement, assuming all d drives are used. For example, for 2-way mirrored placement with strongly limited placement groups, they give an MTTDL of 30 years; ODF-MTBLE predicts 0.1 years for *full* spread placement and 47.6 years for partitioned placement, both at 2 PB of content.

The validation work described in Section 10 has only begun the task of validating the approach and toolset. Future work will consider similar studies involving more storage system types, more extensive comparisons, and comparison to lab experiments. The goal of such work is to further build confidence and to examine the validity of the core model and its assumptions (such as the modeling assumption of Section 5).

Conclusions. To accurately predict quantitative reliability of large scale storage systems and to compare their scaling laws, it is essential to handle large numbers of devices and dependencies among them, both at the hardware and software levels. Our ODF tools, ODF-MTBLE and ODF-MLR enable studying the reliability scaling of storage system designs and comparing them to each other without considering hardware deployment issues. The CQSIM-R simulator, on the other hand, allows one to study the effects of deployment decisions and other dependencies. It accounts for hardware dependencies through nonfunctioning status propagating down the component hierarchy from a failed component. We have quantified the statistical accuracy of the simulation results. Moreover, we have described a general combinatoric framework that allows easily incorporating new placement strategies, both at the analytical tool level and into the simulator. This makes it possible to study and compare many designs of interest to industry today, such as rack-aware, copyset, and limited-spread. We have illustrated and validated the tools, showing the spread catastrophe and various dependency effects, such as the effect of data center distribution on reliability of different storage systems. Finally, we have begun work on validating the tools in five ways, providing evidence of GLEP's correctness, consistency between simulation and ODF tools, consistency between the tools and the Angus model in its domain of applicability, and validity of the Guerriero confidence estimate. The toolset's accuracy and performance encourage further work toward a comprehensive set of tools for designing large scale storage systems to meet stakeholder requirements.

REFERENCES

ANGUS, J. E. 1988. On computing mtbf for a k-out-of-n:g repairable system. *IEEE Trans. on Reliability 37*, 312–313.

BAIRAVASUNDARAM, L. N., GOODSON, G. R., PASUPATHY, S., AND SCHINDLER, J. 2007. An analysis of latent sector errors in disk drives. In *Proc. of ACM 2007 Intl. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS'07)*.

BORTHAKUR, D. 2007. The hadoop distributed file system: architecture and design. Tech. rep., Apache Software Foundation.

CERN-LHC. CERN/LHC: Computing. home.web.cern.ch/ about/computing. Accessed: 2014-09-01.

CHEN, P. M., LEE, E. K., GIBSON, G. A., KATZ, R. H., AND PATTERSON, D. A. 1994. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys 26*, 145–185.

CIDON, A., RUMBLE, S., STUTSMAN, R., KATTI, S., OUSTERHOUT, J., AND ROSENBLUM, M. 2013. Copysets: reducing the frequency of data loss in cloud storage. In *Proceedings of the 2013 Usenix Annual Technical Conference*.

ELERATH, J. G. AND PECHT, M. 2007. Enhanced reliability modeling of raid storage systems. In *Proceedings of IEEE/IFIP Intl. Conf. on Dependable Systems and Netowrks*.

ELERATH, J. G. AND SCHINDLER, J. 2014. Beyond mttdl: A closed-form raid 6 reliability equation. *Trans. Storage 10,* 2, 7:1–7:21.

FORD, D., LABELLE, F., POPOVICI, F. I., STOKELY, M., ANH TRUONG, V., BARROSO, L., GRIMES, C., AND QUINLAN, S. 2010. Availability in globally distributed storage systems. In *Proceedings of the 9th Usenix symposium on operating systems design and implementation*.

GREENAN, K., PLANK, J., AND WYLIE, J. 2010. Mean time to meaningless: Mttdl, markov models, and storage system reliability. In *Proceedings of 2nd USENIX Conference on Hot Topics in Storage systems*.

GUERRIERO, V. 2012. Power law distribution: method of multi-scale inferential statistics. *Journal of Modern Mathematics Frontier 1*.

ILIADIS, I. AND VENKATESAN, V. 2015. Rebuttal to 'beyond mttdl: A closed-form raid-6 reliability equation'. *ACM Transactions on Storage 11,* 2.

KAO, H.-W., PARIS, J.-F., SCHWARZ, T., AND LONG, D. 2013. A flexible simulation tool for estimating data loss risks in storage arrays. In *Proc. IEEE 29th Symp. on Mass Storage Systems and Technologies*.

K.K.RAO, HAFNER, J. L., AND GOLDING, R. A. 2006. Reliability for networked storage nodes. In *Intl. Conf. on Dependable Systems and Networks*.

OVSIANNIKOV, M., RUS, S., REEVES, D., SUTTER, P., RAO, S., AND KELLY, J. 2013. The quantcast file system. In *Proc. of the VLDB Endowment*. Vol. 6.

PATIEJUNAS, K. 2014. Freezing exabytes of data at facebook's cold storage. In *Proc. Workshop on Designing Storage Architectures for Digital Collections 2014*.

PINHEIRO, E., WEBER, W.-D., AND BARROSO, L. A. 2007. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*.

RESCH, J. AND VOLVOVSKI, I. 2013. Reliability models for highly fault-tolerant storage systems. arXiv.org, arXiv:1310.4702.

SCHROEDER, B., DAMOURAS, S., AND GILL, P. 2010. Understanding latent sector errors and how to protect against them. *ACM Transactions on Storage 6,* 3.

SCHROEDER, B. AND GIBSON, G. A. 2007. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *Proceedings of the 5th Usenix conference on file and storage technologies*.

STORER, M., GREENAN, K., MILLER, E., AND VORUGANTI, K. 2008. Pergamum: replacing tape with energy efficient, reliable, disk-based archival storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*.

VENKATESAN, V. AND ILIADIS, I. 2012. A general reliability model for data storage systems. In *Proc. of 9th Intl. Conf. on quantitative evaluation of systems*.

VENKATESAN, V., ILIADIS, I., FRAGOULI, C., AND URBANKE, R. 2011. Reliability of clustered vs declustered replica placement in data storage systems. In *Proc. of 19th Intl. Symp. on Modeling, analysis, & simulation of computer and telecommunication systems*.

VENKATESAN, V., ILIADIS, I., AND HAAS, R. 2012. Reliability of data storage systems under network rebuild bandwidth constraints. In *Proc. of 20th Intl. Symp. on Modeling, analysis, & simulation of computer and telecommunication systems*.

WEIL, S. A., BRANDT, S. A., MILLER, E. L., LONG, D. D., AND MALTZAHN, C. 2006a. Ceph: a scalable, high-performance distributed file system. In *Proc. of 7th USENIX Symp. on Operating Systems Design and Implementations*.

WEIL, S. A., BRANDT, S. A., MILLER, E. L., AND MALTZAHN, C. 2006b. Crush: Controlled, scalable, decentralized placement of replicated data. In *Proc. of the 2006 ACM/IEEE conference on Supercomputing*.

XIN, Q., MILLER, E. L., SCHWARZ, T., LONG, D. D. E., BRANDT, S. A., AND LITWIN, W. 2003. Reliability mechanisms for very large storage systems. In *Proceedings of the 20th IEEE Conference on Mass Storage Systems and Technologies*.

Received July 2015; revised November 2015; accepted April 2016