Reliability of modular mesh-connected intelligent storage brick systems

C. Fleiner
R. B. Garner
J. L. Hafner
KK Rao
D. R. Kenchammana-Hosekote
W. W. Wilcke
J. S. Glider

A key objective of the IBM Intelligent Bricks project is to create a highly reliable system from commodity components. We envision such systems to be architected for a service model called ***fail-in-place*** *or* ***deferred maintenance****. By delaying service actions, possibly for the entire lifetime of the system, management of the system is simplified. This paper examines the hardware reliability and deferred maintenance of intelligent storage brick (ISB) systems assuming a mesh-connected collection of bricks in which each brick includes processing power, memory, networking, and storage. On the basis of Monte Carlo simulations, we quantify the fraction of bricks that become unusable by a distributed data redundancy scheme due to degrading internal bandwidth and loss of external host connectivity. We derive a system hardware reliability expression and predict the length of time ISB systems can operate without replacement of failed bricks. We also show via a Markov analysis the level of fault tolerance that is required by the data redundancy scheme to achieve a goal of less than two data loss events per exabyte-year due to multiple failures.*

Introduction

The Intelligent Bricks project investigates storage systems based on a modular brick architecture with the objectives of simplifying system management, providing a large scaling range, and creating a reliable system from commodity components. Storage servers built with a single type of module, or *brick*, are attractive in terms of simplicity, scalability, and cost. Bricks include processing, memory, networking, and storage sufficient to run a distributed software system that delivers higher data reliability than that offered by the underlying hardware.

A key property of an intelligent storage brick (ISB) system is its *fail-in-place* or *deferred-maintenance* architecture: By over-provisioning or adding additional bricks while operating, hardware maintenance can be delayed for several years—possibly for the entire lifetime of the system. The distributed system software is responsible for automatically invoking spare disks or bricks as components fail. The only maintenance task users are expected to perform is to physically add bricks to meet growing capacity requirements.

This paper presents quantitative insights into the operating characteristics of mesh-connected ISB systems, in which bricks communicate only with physically adjacent bricks. We characterize such systems by the fraction of unusable bricks due to degrading internal bandwidth and external host connectivity, and a reliability expression that approximates the length of time that ISB systems can operate without replacement of failed bricks. Our goal is that ISB systems provide nearly 100% data availability, no ongoing hardware maintenance actions for several years, and a very low probability of data loss due to multiple failures. This paper is a companion to [1], which presents the overall ISB system and an operational $3 \times 3 \times 3$-brick prototype.

0018-8646/06/$5.00 © 2006 IBM

Related work

The approach of distributing data across independent machines to build scalable storage systems has been explored in DataMesh [2], FAB [3], Self-* [4], Petal [5], and OceanStore [6]. Several companies are shipping products based on distributed data redundancy, including Panasas [7], Pivot3, LeftHand Networks, and Isilon. However, none of these approaches focus on fail-in-place or deferred maintenance. The Panasas system, while implementing a distributed RAID5 scheme, is oriented more toward delivering high performance. DataMesh [2] was a two-dimensional mesh-connected storage server that most closely resembled our ISB system and introduced concepts of distributed redundancy, fault isolation, and recovery. The more recent FAB project [3] proposed to build a brick storage system from commodity parts. The Self-* project [4] has a focus on simplifying administration by using a brick storage system, including mechanisms to schedule resources, classify files, and manage replicas.

An initial analysis of overprovisioning for capacity and bandwidth in an ISB system was first described by Kirkpatrick et al. [8]. They conservatively defined usable bricks as those that were connected to at least two or three other bricks. Our approach places data on bricks that may have only a single remaining connection to other bricks while avoiding possible set partitioning due to brick failures.

Brick usability in degrading cubes

A pristine cube (i.e., an initial cube with no failed bricks) contains N bricks arranged as a two-dimensional (2D) ($h \times h$) or three-dimensional (3D) ($h \times h \times h$) nearest-neighbor network mesh. Each brick contributes storage, network bandwidth, memory, and processing resources. The bricks run system software that manages the storage data and implements a distributed RAID (dRAID) scheme, in which storage data is copied or encoded in multiple chunks and each placed on a distinct brick. As bricks progressively fail, a pristine cube slowly declines in performance and capacity. In this section we establish operating ranges of usable bricks in 2D and 3D mesh-connected ISB systems.

For purposes of our analysis, we make the following assumptions:

1. All bricks in the system are identical and contain sufficient processing, memory, networking, and data storage. Bricks communicate with adjacent (neighbor) bricks in a 2D or 3D network mesh topology.
2. A given brick is either completely functional (live) or completely inoperative (failed). When a brick fails, it reduces system network bandwidth as links between it and neighbors are lost, creating "holes" in the mesh.
3. Storage data is redundantly distributed across multiple bricks to ensure a high probability of restoring data after a failure of disks or bricks. When there is a failure, redundant data is rebuilt by the operating software over all surviving storage bricks. We assume that data is randomly distributed across all storage bricks, parameterized by the number of bricks k over which the redundant data chunks are distributed (its set-k, ranging from set-2 for simple mirroring to set-14 for space-efficient or higher-fault-tolerant codes). For example, a traditional 6-data and 1-parity RAID5 scheme would be set-7. Although details of distributed redundancy schemes are not discussed here, see [9] for implementation approaches.
4. The system is overprovisioned with bricks when it is assembled. In realistic deployments, a user could add bricks over time to compensate for brick losses, undoubtedly with improved cost and capacity attributes. Nevertheless, to simplify the analysis, we assume that bricks are not added over time.

Although we analyze symmetrical 3D systems in this paper, our approach can also be applied to systems with nonsquare, rectangular cross sections. This is relevant for actual implementations, because the system height may be limited by floor loading or other structural considerations.

Simulation methodology

For our analysis, we performed Monte Carlo simulations of mesh-connected cubes with randomly selected failing bricks. To find the number of usable bricks in a degrading cube, we assumed that redundant data is placed via a distributed-redundancy, set-k scheme in equal-sized chunks across k bricks. A brick may be live but unusable because of the set-k placement constraints described next.

For each run, the simulator started with an initial, pristine cube and progressively failed one brick per iteration step. For each step, the placement algorithm first found the largest network of connected bricks with at least two surface bricks and then constructed as many set-ks as possible. An additional placement constraint was that the potential failure of any brick would leave at least $k - 1$ usable bricks for all of the constructed set-ks (i.e., a set-k cannot be partitioned if any single brick fails). This estimation process yielded a lower bound on the number of usable bricks for a given set-k placement [10]. We also assumed that chunks in 3D cubes are not placed together in vertical columns of bricks that can fail due to shared

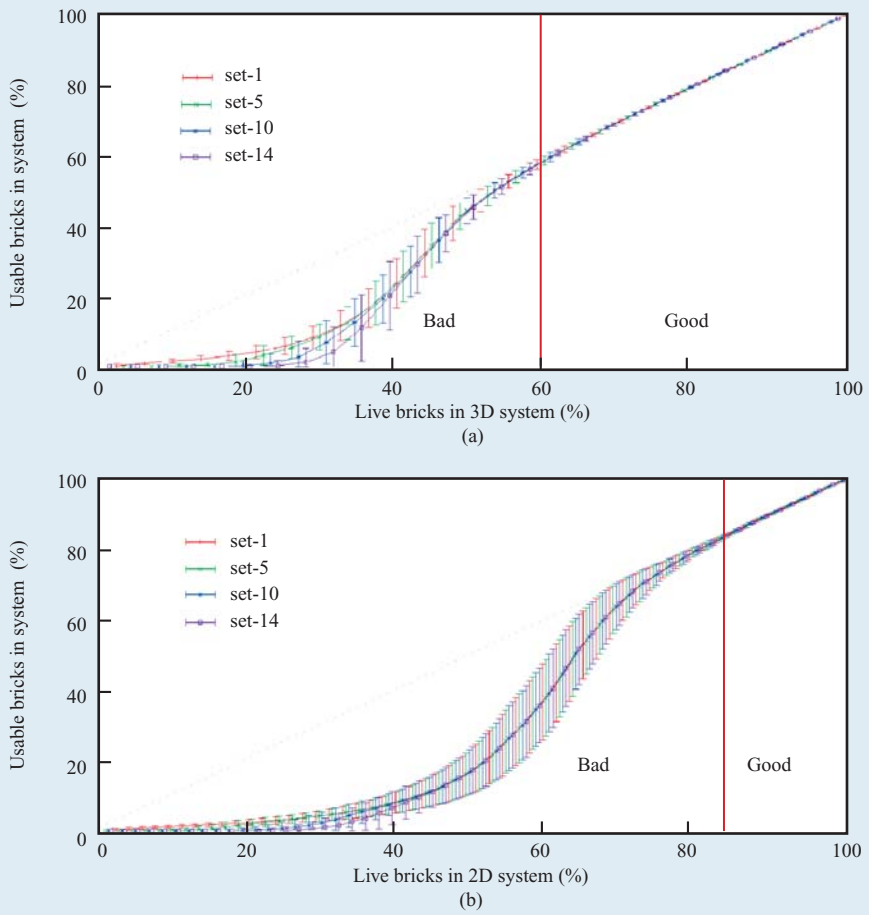

Figure 1

(a) Mean and standard deviation of usable bricks in a degrading $6 \times 6 \times 6$ mesh-connected cube with distributed data redundancy (assuming that no new bricks are added). (b) Percentage of live bricks usable in a degrading 15×15 mesh-connected 2D system with distributed data redundancy (assuming that no new bricks are added).

power and cooling in the column. (We found that this had little effect on the results.) We ran at least 300 different simulation runs for each pristine cube size.

Figures 1(a) and **1(b)** respectively show the average and standard deviation results for degrading $6 \times 6 \times 6 = 216$ and $15 \times 15 = 225$ brick systems. We also ran simulations for larger and smaller cubes and found that the results presented here generally apply to 3D cubes down to 64 bricks and 2D systems down to 16 bricks (with set-5 placement).

Usable bricks in 3D mesh systems

Figure 1(a) plots the number of usable bricks in a degrading 3D system of $6 \times 6 \times 6 = 216$ bricks with distributed redundant data. It plots the percentage of live bricks along the x-axis and the percentage of usable bricks along the y-axis. A pristine cube corresponds to the point in the upper right-hand corner. As a cube degrades, the number of live bricks decreases, as illustrated from right to left in the figure. Note that one should track the lower end of the standard deviation error bars, not the statistical mean.

The data shows that as live bricks decline to 60%, even the large set-14 distributed data placement is able to use nearly all live bricks. However, on losing more than about 50% of the bricks, there exists a noticeable fraction (>10%) of live but unusable bricks. As can be seen in the "bad" operating range, because of the increasing variability in the standard deviation of usable bricks, we suggest that a 3D cube not operate below approximately 60% live bricks.

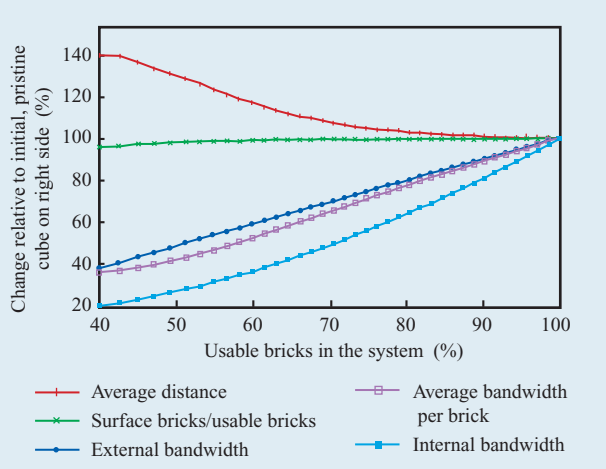

Figure 2

Change in several network bandwidth metrics for a degrading $6 \times 6 \times 6$ cube as a function of usable bricks.

Usable bricks in degrading 2D mesh systems

Figure 1(b) plots the number of usable bricks in a degrading 2D system of $15 \times 15 = 225$ bricks with distributed redundant data. These 2D plots show that once more than 15% of the bricks fail, the number of live but unusable bricks rises rapidly. Note that these 2D results are not applicable to smaller 2D systems (for <16 bricks with smaller set-k placements). We do not examine 2D systems further in this paper.

Network quality in degrading cubes

In the previous section we presented simulation results for the fraction of usable bricks in a degrading cube. In this section, we look at the performance of the interconnection network as bricks fail in terms of total and average random bandwidth (for 3D systems only). These results were derived from the Monte Carlo simulations described above and are combined in **Figure 2**, where the x-axis shows the percentage of usable bricks—not the percentage of live bricks, as in Figures 1(a) and 1(b)—and the y-axis shows the percentage of change in the metric as bricks fail progressively from right to left in the figure.

Effect of brick failures on average network distance

Network distance, the number of hops required to route a packet between two bricks via the shortest path, is a key metric, reflecting brick-to-brick latency and internal bandwidth. For a pristine, mesh-connected cube of edge length h, the average network distance (i.e., the average of the distance between two bricks taken over all source and destination bricks in a cube) is given by $d_{\text{pristine}} = h - (1/h)$, derived under the assumption that all source bricks communicate with all destination bricks via the shortest paths. As bricks fail and the cube degrades, the simulated brick configurations are evaluated at each step to determine the actual average network distance of the cube.

The *average distance* plot (Figure 2) increases slowly at the beginning as bricks fail (right side) and then levels off at about 40% usable bricks (on the left side), with an average network distance about 40% greater than the pristine cube.

From this plot, we conclude that the average network distance does not appreciably increase in 3D cubes as long as about 70% of the bricks remain usable. At 60% usable bricks, the average network distance has increased by about 10%.

Effect of brick failures on internal bandwidth

The total peak internal bandwidth of a cube is given by the number of connections between all usable bricks multiplied by the bandwidth provided by an internal brick face. The internal bandwidth plot in Figure 2 decreases faster than the loss of usable bricks: At 80% of usable bricks, only about 65% of the original peak bandwidth remains, while at 60% of usable bricks, only about 40% remains.

Although total internal bandwidth degrades faster than failing bricks, the number of usable bricks also declines, so the average bandwidth degradation per brick does not decline as quickly. We plot the average internal bandwidth per brick by dividing the total peak internal bandwidth by the average network distance and by the number of usable bricks. The average bandwidth per brick plot in Figure 2 is nearly proportional to the number of usable bricks: At 80% usable bricks, about 80% of the original average bandwidth per brick remains, and at 60% of usable bricks, about 50% remains.

Effect of brick failures on external bandwidth

The external bandwidth of a cube is given by the number of usable surface bricks multiplied by the bandwidth provided by a surface brick face. The external bandwidth plot in Figure 2 indicates that the external bandwidth of the hosts decreases proportionally as the number of usable bricks drops.

Figure 2 also shows that the number of usable surface bricks in a cube is proportional to the total number of usable bricks (labeled surface bricks/usable bricks). This ratio remains nearly constant as bricks fail.

External host connectivity

We now examine the reliability of host connectivity to a degrading cube. For simulations, we assumed that external hosts can access the set of remaining usable

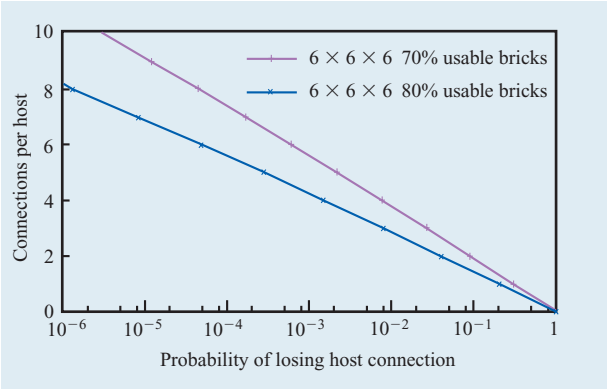

Figure 3

Probability of an external host not having a connection to the usable bricks as a function of number of per-host connections for the cases of 70% and 80% usable bricks in a $6 \times 6 \times 6$ cube.

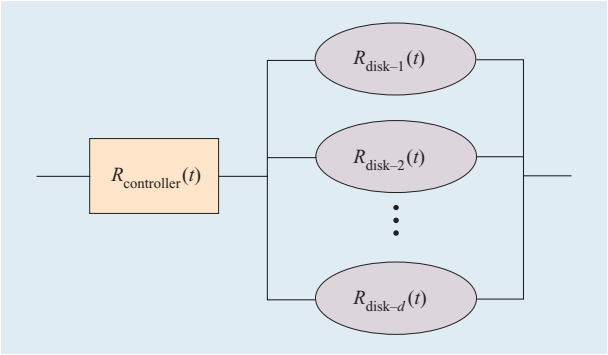

Figure 4

Simplified reliability graph for a storage brick illustrating the reliability of the controller electronics $R_{\text{controller}}(t)$ in series with the reliability of d independent disks in parallel, each with reliability $R_{\text{disk–}d}(t)$.

bricks in the cube via at least one surface brick. However, this implies that every host is connected to every brick on the surface of the cube, which is not practical. Instead, we need to assume that each host has multiple cube connections, each to different surface bricks. We then calculate the probability that the usable surface connections to a host will not be members of the usable bricks in the cube.

Given a cube with the number of usable surface bricks U out of the total number of surface bricks $S = 6\sqrt[3]{N^2} - 12\sqrt[3]{N} + 8$, C connections per host to the cube, and assuming usable surface bricks proportional to total usable bricks (as in the previous section), the probability that a host has no connections to the usable bricks is given by

$$P_{\text{host\_has\_no\_connection}} = \frac{(S-C)!(S-U)!}{(S-U-C)!S!}.$$

Using the above equation, **Figure 3** plots the probability of a host being unconnected to the usable bricks against the number of per-host connections with either 70% or 80% usable bricks. The graph illustrates that, for every additional two per-host connections, the risk of being unconnected to the usable bricks is reduced by a factor of approximately 10. To achieve $P_{\text{host\_has\_connection}} > 0.99999$ in a cube with 70% usable bricks, the number of per-host connections should be 9 or greater. Plotting the same graph for different cube sizes shows that the probability depends primarily on the number of host connections, not size.

From these results, we conclude that either an external network switch is needed between the surface bricks and the hosts or, alternatively, the hosts are in the cube itself (which then moves the surface connectivity problem to the clients of the hosts). Although it is possible for a user to remove a host connection from a failed brick and reconnect it to a remaining usable surface brick, we assume that such maintenance activity is undesirable.

System hardware reliability

A key objective for ISB systems is deferred maintenance, i.e., that they can operate without replacement of failed hardware for long periods of time. Failed bricks remain in place at least until service or maintenance is performed, and possibly until end of life of the system. Our goal is a system that provides nearly 100% data availability with no ongoing maintenance actions (except possibly to add capacity). In this section, we calculate the deferred-maintenance time period for a system as a function of the fraction of brick failures and the reliability of disk and brick controller electronics. We assume that a system is fully functional and optionally overprovisioned with bricks at its start of operation.

The reliability of a system is defined as the probability that it operates properly over a time t. As illustrated in **Figure 4**, a storage brick contains controller electronics in series with d parallel disks that can fail independently while leaving the brick operational. The reliability of the independent, parallel-connected disks of a brick is the complement of their unreliability; i.e., it is one minus the probability that all of its disks fail over time t.

Furthermore, the reliability of the series-connected controller electronics of a brick and its collection of parallel disks is the product of their reliabilities [11]. Thus, the hardware reliability of a storage brick over a time period t is given by

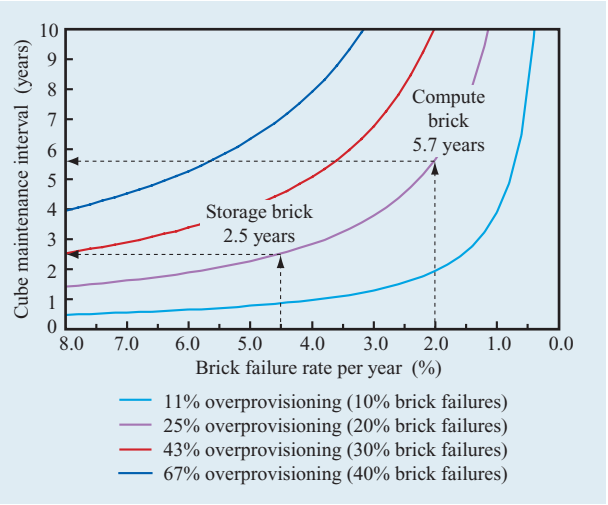

Figure 5

Maximum maintenance-deferred durations for four cases of brick failure/overprovisioning in a 6 × 6 × 6 system with a system reliability target of 0.99999 as a function of the brick hardware failure rate. The four constant brick reliability curves correspond to 10%, 20%, 30%, and 40% brick failures at the end of the deferred-maintenance time period.

$$\begin{aligned} R_{\text{brick}}(t) &= R_{\text{controller}}(t) \times R_{\text{disks}}(t) \\ &= R_{\text{controller}}(t) \times \left[1 - F_{\text{disk}}^{d}(t)\right] \\ &= R_{\text{controller}}(t) \times \left\{1 - [1 - R_{\text{disk}}(t)]^{d}\right\}. \end{aligned}$$

From this equation, we observe that the parallel disks can achieve a very high hardware reliability, even with disks that have high rates of failure (e.g., $R_{\text{disks}} = 0.99999$ for six parallel disks at 3% annual constant failure rate each over five years). Thus, to simplify the analysis below, we assume that the reliability of the storage brick is approximately equal to the reliability of the controller electronics, or $R_{\text{brick}}(t) \approx R_{\text{controller}}(t)$. Note that $R_{\text{controller}}(t)$ includes everything in the brick exclusive of disks.

The system hardware reliability $R_{\text{system}}(t)$ is defined as the probability that a system will meet its mission to provide a target number of live bricks and storage capacity through the deferred-maintenance time period t. Our approach to approximate $R_{\text{system}}(t)$ is to use the frequency interpretation of probability [12] and assume ideal, identically deployed systems with no systematic environmental influences, each with identical bricks and disks that fail independently. The hardware reliability of our uniform brick system can then be approximated by the cumulative binomial distribution [11]—the probability that M out of N independent and identical bricks survive over time period t:

$$R_{\text{system}}(t) = \sum_{i=0}^{N-M} \binom{N}{i} R_{\text{brick}}^{N-i}(t)[1 - R_{\text{brick}}(t)]^{i}.$$

To determine the maximum deferred-maintenance time duration t of our system, we first decide on a target value for $R_{\text{system}}(t)$ and then solve the binomial probability distribution for t as a function of that target reliability, M/N fraction of live bricks, and the predicted brick reliability.

For setting a system reliability target, if our acceptable goal is that one out of D deployed systems fails to achieve a deferred-maintenance mission, we can set target $R_{\text{system}}(t) = 1 - 1/D$. For example, if $R_{\text{system}}(t) = 0.99999$, approximately one system out of 100,000 will fail its mission to remain above the target number of live bricks or storage capacity over deferred-maintenance time t.

Substituting an expected, constant, and memoryless failure rate λ_{brick} with brick reliability $R_{\text{brick}}(t) = e^{-\lambda_{\text{brick}} t}$ into the above binomial distribution yields

$$R_{\text{system}}(t) = \sum_{i=0}^{N-M} \binom{N}{i} e^{-\lambda_{\text{brick}} t(N-i)} \left(1 - e^{-\lambda_{\text{brick}} t}\right)^{i}.$$

Using target constant values for $R_{\text{system}}(t)$, M, N, and a range of brick failure rates λ_{brick}, we iteratively solve this equation for maximum values of t.

As an example, we select a system hardware reliability target of $R_{\text{system}}(t) = 0.99999$ for $N = 6 \times 6 \times 6 = 216$ brick systems and vary the fraction M/N of live bricks from 90% down to 60% in 10% steps, corresponding to 10% to 40% of failed bricks and optional initial overprovisioning levels of $1/(1 - M/N)$, or 11%, 24%, 43%, and 67%, respectively.

In **Figure 5**, we plot the brick failure rate λ_{brick} on the x-axis against the maximum deferred-maintenance period t on the y-axis for these four brick failure cases. The labeled compute brick point shows that a brick failure rate of 2% in a system that expects 20% failed bricks (optionally 25% overprovisioned), would allow for deferred-maintenance system operation for nearly six years. The labeled storage brick point in Figure 5 illustrates a 2.5-year deferred-maintenance period, achievable with a combined controller, and a disk failure rate of 4.5% per year, as described next.

Next we look at the hardware reliability of storage capacity; that is, the probability that the system will have sufficient storage disk capacity at the end of the deferred-maintenance time period. Although all of the disks operate in parallel and can fail independently, from the perspective of the data stored on a disk, the brick controller electronics are in series with each disk. Because the reliability of series-dependent components is the product of the component reliabilities [11], the reliability of the storage capacity itself is $R_{\text{storage}}(t) = R_{\text{controller}}(t) \times$

$R_{disk}(t)$. With our assumption of constant failure rate and exponential reliability for brick electronics and disks, the overall resulting storage capacity failure rate is $\lambda_{storage} = \lambda_{controller} + \lambda_{disk}$, per these equivalent equations:

$$R_{storage}(t) = R_{controller}(t) \times R_{disk}(t),$$

$$e^{-\lambda_{storage} t} = e^{-\lambda_{controller} t} \times e^{-\lambda_{disk} t},$$

and

$$\lambda_{storage} = \lambda_{controller} + \lambda_{disk}.$$

Substituting $\lambda_{storage}$ in place of λ_{brick} as the failure rate in the cumulative binomial equation, Figure 5 then shows the expected deferred-maintenance intervals for storage capacity. For example, if one assumes a brick controller annual failure rate of 1.5% [mean time between failures (MTBF) ≈ 580,000 hours] and a disk failure rate of 3% (MTBF ≈ 300,000 hours), the effective hardware storage failure rate is 1.5% + 3% = 4.5% per year, yielding a system deferred-maintenance duration of 2.5 years for 100,000 systems that expect up to 20% failed bricks (optionally 25% overprovisioned).

System storage data reliability

The previous section examined the relationship between brick hardware failure rates and the deferred-maintenance duration of ISB systems before accumulated hardware failures warrant maintenance. To implement a high level of data reliability and to guard against irrecoverable data loss, a distributed data redundancy scheme is implemented by the system software across multiple bricks. In this section we present results of Markov models for deriving the probability that a system will not lose data after multiple brick or disk failures, assuming different distributed data redundancy schemes and system performance characteristics.

Standard storage redundancy schemes store either one or more copies of the original data or precomputed redundancy information, such as parity. After brick or disk failure and data erasure are detected, the system software uses a copy to rebuild the original and redundant data in spare, unused storage. Whether a particular hardware or software fault results in the loss of storage data is a function of three elements: the fault tolerance of the redundancy coding scheme; the system network and brick-internal bus bandwidths available to rebuild redundant data (which determines rebuild time); and the probability of additional hardware or software faults during the rebuild process. Note that for performance considerations, redundant parity information is seldom verified on storage read operations.

We assume that if a brick or disk fails, the distributed software will detect that erasure event, for example, when disks return error codes or are unresponsive to multiple retry commands, or when bricks do not respond after multiple out-of-network reboot attempts. Our goal is a very small probability of data loss after multiple brick or disk failures: no more than one data-loss event in five years for 100 one-petabyte systems, or, equivalently, only two data-loss events per exabyte-year.

The redundancy scheme that is the easiest to implement is to mirror or duplicate data. Although the storage write performance is highest, the data storage efficiency is low. There exist several redundancy schemes that guard against multiple failures with high storage efficiency—with the tradeoff of lower write performance. These schemes include RAID5, RAID6, and others that tolerate three or more failures [13, 14]. Redundancy schemes for higher fault tolerance can be realized through several erasure codes such as EVENODD [15], Reed–Solomon [16], low-density parity check (LDPC) [17], and WEAVER, a new constant-efficiency, high-fault-tolerant erasure code [18].

Redundancy schemes can be characterized by several parameters: number of faults tolerated, storage efficiency, and storage write performance. These factors are interrelated; optimizing for two usually results in tradeoffs against the third. From the user's perspective, these parameters translate into the reliability of the storage data, its cost, and overall application performance. Thus, the choice of a particular data redundancy scheme depends on the business cost of a data-loss event, how much the user is willing to pay for higher levels of fault tolerance, and the level of application performance required. Ideally, the operating software will allow the system administrator to make reliability tradeoffs for the user's data.

The fault tolerance and storage efficiency of a particular redundancy scheme defines the minimum number of bricks that must be usable and corresponds to the set-*k* metric defined earlier. For example, an 80% efficient, single-fault-tolerant RAID5 scheme requires a minimum of five usable bricks for placing data and parity chunks (set-5). A 50% efficient scheme with three data and three parity chunks requires a set-6 placement.

When a brick or disk failure is detected, the distributed operating software rebuilds the erased data across all of the surviving usable bricks. Because the large aggregate bandwidth of the entire 3D network is available for this task, the rebuild times scale well with the size of the cube, minimizing the exposure time to subsequent failures. Note that data redundancy may be implemented not only across bricks, but within bricks as well. Schemes with in-brick redundancy assume that failing disks are not replaced, so erased data in the brick is rebuilt across the remaining operational disks in a brick (assuming sufficient remaining capacity).

By constructing and analyzing Markov chain failure models of representative systems, we compared the efficacy of several dRAID schemes assuming typical brick

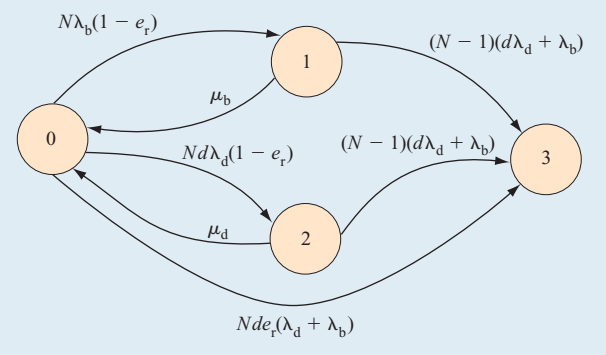

Figure 6

Markov failure model for a 1-fault-tolerant dRAID scheme with no redundancy in bricks.

and disk hardware failure rates. As an example, a simple loss-of-data scenario for a redundancy scheme supporting 1-fault-tolerance[1] among bricks with no in-brick redundancy is illustrated in **Figure 6**.[2] Loss of data due to multiple failures occurs when either a brick controller or a disk fails (at rates λ_b and λ_d), followed by a rebuild and repair process (at rates μ_b and μ_d), during which time a second brick controller or disk fails—or the more probable case of a hard and unrecoverable error occurring during a disk read (with probability e_r). The 3-fault-tolerant Markov graph is more complex, requiring 16 states, as described in [19].

The Markov model reliability calculations for 1-, 2-, and 3-fault-tolerant dRAID schemes for a pristine $4 \times 4 \times 4$ cube with 12-disk bricks are presented in **Figure 7**, which shows that mirrored, single-fault-tolerant schemes such as RAID5 fall far short of our target, coming in at about 30,000 data-loss events per exabyte-year. Mirroring between bricks, together with single-fault-tolerant RAID5 in the bricks, is 1,000 times better, at about 30 data losses per exabyte-year.

At least 2-fault tolerance is required between bricks to achieve our target of two data-loss events per exabyte-year, using either of two schemes:

- A 2-fault-tolerant dRAID scheme across bricks, combined with single-fault-tolerant RAID5 in the bricks for 0.009 data-loss events per exabyte-year (or 9 data losses per zettabyte-year); or, alternatively,
- A 3-fault-tolerant dRAID scheme across bricks with no redundancy in the bricks for 0.001 data-loss events per exabyte-year (or one data loss per zettabyte-year).

[1]The number of disks or bricks that can fail, f, is specified by the fault tolerance of the redundancy scheme. The Hamming distance of the redundancy scheme is $f + 1$.

[2]Note that in the figure a transition from state 2 to 1 (a disk failure followed by its brick controller failure before rebuild completes) is not shown, but has little effect on the results.

The parameters used in these Markov model calculations include disk MTBF = 300,000 hours, brick electronics MTBF = 400,000 hours, disk read hard error rate of 10^{-14}, $4 \times 4 \times 4 = 64$ bricks, 12 disks per brick, 40-MB/s disk bandwidth, six 800-MB/s 3D mesh network links per brick, rebuild block size = 128 KB, and 10% bandwidth utilization for data redundancy rebuilding. These calculations, including assumptions and sensitivity analyses, are discussed in more detail elsewhere [19]. In summary, we found that the probability of a data-loss event was generally insensitive to the cube size, but sensitive to three variables: the disk failure rate in bricks without internal RAID, the brick controller electronics failure rate, and the rebuild block size.

Conclusion

We quantified the effects of brick failures in modular, mesh-connected ISB systems as gauged by capacity, network performance, system reliability, and the probability that a system does not lose data after multiple disk or brick failures. Using Monte Carlo simulations, we examined the effects of loss of usable bricks for distributed data placement on network bandwidth in order to recommend a fraction of usable bricks above which systems should operate.

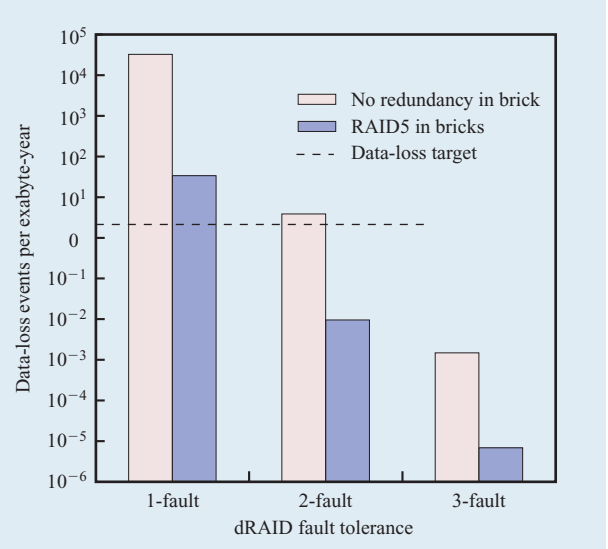

Figure 7

Comparison of predicted dRAID data-loss events per exabyte-year after multiple failures assuming six different fault-tolerant schemes for a pristine $4 \times 4 \times 4$ cube of 12-disk bricks. "No redundancy in bricks" results assume only dRAID across bricks with no RAID in the bricks. "RAID5 in bricks" assumes a "fail-in-place" RAID5 scheme in each brick together with N-fault redundancy across bricks, as shown on the x-axis. The dashed line is our target of two data-loss events per exabyte-year.

Our studies show that for 3D, mesh-connected cubes with failing bricks, essentially all bricks are usable for data placement as long as 60% or more of the bricks are operational. Below this level, we found an increase in the number of isolated and unusable bricks and a progressive decrease in internal bandwidth.

Although our results show that 3D systems scale well down to about 60% live bricks, to minimize optional overprovisioning and allow margin for dependent or nonconstant failure rates, we expect that ISB systems can operate down to an 80% live-brick level (corresponding to optional 25% overprovisioning). While operating in this region for the deferred-maintenance duration, we found the following:

- Essentially all live bricks are usable by distributed data redundancy schemes up to set-14.
- Because the internal average bandwidth per brick is approximately proportional to the number of usable bricks, the average bandwidth per brick is about 80% of the initial, pristine cube value.
- Either multiple connections per host or an external switch is necessary between hosts and the cube surface bricks.

We also showed the following for $6 \times 6 \times 6$ cubes down to the 80% live brick level:

- Deployment of 100,000 systems could achieve a deferred-maintenance duration of approximately 2.5 years assuming typical hardware failure rates for disks (3% per year) and brick controller electronics (1.5% per year).
- A 3-fault-tolerant dRAID scheme across bricks can achieve a storage reliability target of less than two data-loss events per exabyte-year caused by multiple disk or brick failures.

Assuming modest overprovisioning, we demonstrated that maintenance in 3D mesh-connected storage systems can be deferred for several years while maintaining adequate performance and achieving high levels of data reliability and availability.

References

1. W. W. Wilcke, R. B. Garner, C. Fleiner, R. F. Freitas, R. A. Golding, J. S. Glider, D. R. Kenchammana-Hosekote, J. L. Hafner, K. M. Mohiuddin, KK Rao, R. A. Becker-Szendy, T. M. Wong, O. A. Zaki, M. Hernandez, K. R. Fernandez, H. Huels, H. Lenk, K. Smolin, M. Ries, C. Goettert, T. Picunko, B. J. Rubin, H. Kahn, and T. Loo, "IBM Intelligent Bricks Project—Petabytes and Beyond," *IBM J. Res. & Dev.* **50**, No. 2/3, 181–197 (2006, this issue).
2. C. Chao, J. Wilkes, D. Jacobson, B. Sears, R. M. English, and A. A. Stepanov, "DataMesh Architecture 1.0," *Technical Report HPL-92-153*, Hewlett-Packard Computer Systems Laboratory, Palo Alto, CA 94304, December 1992.
3. Y. Saito, S. Frølund, A. C. Veitch, A. Merchant, and S. Spence, "FAB: Building Distributed Enterprise Disk Arrays from Commodity Components," *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2004, pp. 48–58.
4. G. R. Ganger, J. D. Strunk, and A. J. Klosterman, "Self-* Storage: Brick-Based Storage with Automated Administration," *Technical Report CMU-CS-03-178*, Carnegie Mellon University, Pittsburgh, PA 15213, August 2003.
5. E. K. Lee and C. A. Thekkath, "Petal: Distributed Virtual Disks," *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1996, pp. 84–92.
6. J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage," *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, November 2000, pp. 190–201.
7. D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale Storage Cluster—Delivering Scalable High Bandwidth Storage," *Proceedings of the ACM/IEEE Supercomputing Conference*, 2004; see *http://www.sc-conference.org/sc2004/schedule/pdfs/pap207.pdf*.
8. S. Kirkpatrick, W. W. Wilcke, R. B. Garner, and H. Huels, "Percolation in Dense Storage Arrays," *Phys. A* **314**, No. 1/4, 220–229 (2002).
9. R. Golding, D. Kenchammana-Hosekote, C. Fleiner, and O. Zaki, "Design and Analysis of Network RAID Protocols," *Research Report RJ-10316*, IBM Almaden Research Center, San Jose, CA 95120, 2004.
10. C. Fleiner, R. Garner, D. Kenchammana-Hosekote, and W. Wilcke, "Quantitative Study of the Performance and Reliability of a Resilient 3-D Mesh-Based Storage Server," *Research Report RJ-10308*, IBM Almaden Research Center, San Jose, CA 95120, May 2004.
11. M. Shooman, *Reliability of Computer Systems and Networks*, John Wiley & Sons, Inc., New York, 2002.
12. E. T. Jaynes, *Probability Theory. The Logic of Science*, G. L. Bretthorst, Ed., Cambridge University Press, New York, 2003.
13. D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 1988, pp. 109–116.
14. *The RAIDBook: A Source Book for RAID Technology*, Sixth Edition, P. Massiglia, Ed., The RAID Advisory Board, St. Peter, MN, 1999.
15. M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures," *IEEE Trans. Computers* **44**, No. 2, 192–201 (February 1995).
16. J. S. Plank and Y. Ding, "Note: Correction to the 1997 Tutorial on Reed–Solomon Coding," *Software, Pract. & Exper.* **35**, No. 2, 189–194 (February 2005).
17. R. G. Gallagher, "Low Density Parity Check Codes," Sc.D. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, 1960.

18. J. L. Hafner, "WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems," *Proceedings of the 4th USENIX Conference on File and Storage Technologies*, December 15, 2005.
19. K. Rao, J. Hafner, and R. Golding, "Reliability for Networked Storage Nodes," *Research Report RJ-10358*, IBM Almaden Research Center, San Jose, CA 95120, 2005.

Received July 5, 2005; accepted for publication August 8, 2005; Internet publication February 22, 2006

Claudio Fleiner *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (fleiner@us.ibm.com)*. Dr. Fleiner received a Ph.D. degree in computer science from the University of Fribourg, Switzerland. Prior to joining the IBM Almaden Research Center, he worked at the IBM Zurich Research Laboratory and Transmeta. Dr. Fleiner's research interests include large storage systems, distributed computing, and computer networks.

Robert B. Garner *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (robgarn@us.ibm.com)*. Mr. Garner received his M.S.E.E. degree from Stanford University. Before joining the IBM Research Division in 2001 to work in the IceCube project, he worked at Xerox Palo Alto Research Center, Sun Microsystems, and Brocade Communications.

James L. Hafner *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (hafner@almaden.ibm.com)*. Dr. Hafner received a Ph.D. degree in mathematics from the University of Illinois. He later joined the IBM Research Division and has worked in diverse areas, including number theory, complexity theory, image databases, and storage-system protocols. Dr. Hafner currently works in the advanced RAID project.

KK Rao *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (kkrao@us.ibm.com)*. Mr. Rao is a Distinguished Engineer in storage systems. He received B.S.E.E. and M.S.E.E. degrees from the Indian Institute of Technology, Bombay, joining the IBM Research Division in 2002. Mr. Rao has been working on advanced RAID algorithms, reliability of distributed storage systems, and scale-out storage systems.

Deepak R. Kenchammana-Hosekote *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (kencham@us.ibm.com)*. Dr. Kenchammana-Hosekote received a Ph.D. degree in computer science from the University of Minnesota. He has been at the IBM Almaden Research Center since 1997 working on various RAID and distributed storage systems.

Winfried W. Wilcke *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (winfriedwilcke@us.ibm.com)*. Dr. Wilcke is a program director at the IBM Almaden Research Center, where he started the Intelligent Bricks project and IceCube implementation in 2001. He was a Senior Manager at the IBM Thomas J. Watson Research Center, responsible for Victor/Vulcan research, later commercialized in IBM SP* supercomputers. As Director of Architecture (later Chief Technical Officer) of HaL computers, he was deeply involved in the creation of the 64-bit SPARC** architecture. Dr. Wilcke received a Ph.D. degree in nuclear physics and previously worked at the University of Rochester and at the Los Alamos and Lawrence Berkeley National Laboratories.

Joseph S. Glider *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (gliderj@almaden.ibm.com)*. Mr. Glider is a Senior Technical Staff Member and manager in charge of the Intelligent Bricks Storage Software project at the IBM Almaden Research Center. He received a B.S.E.E. degree from Rensselaer Polytechnic Institute. Mr. Glider's research interests include distributed storage systems and fault-tolerant storage systems.