RJ 10308 (A0311-036) November 10, 2003
Computer Science

IBM Research Report

Quantitative Study of the Performance and Reliability of a Resilient 3-D Mesh-Based Server

Claudio Fleiner, Deepak R. Kenchammana Hosekote,
Robert B. Garner, Winfried Wilcke
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Quantitative Study of the Performance and Reliability of a Resilient 3-D Mesh-Based Server

Abstract

The task of managing large servers is becoming ever more complex as systems are growing. Human errors in reconfiguring and maintaining systems is a dominant cause of outages and data loss. In this paper we present an analysis of a system that is architected around an alternative service model, called "Fail-in-Place" or "Deferred Maintenance".

In such a system resources are either initially over-provisioned or assumes timely addition of new resources. By delaying service actions - possibly for the entire lifetime of the system - management of the system can be simplified.

In this paper we study the effects of progressive resource failures on the capacity, performance, and reliability of a 3-D, mesh-based cube. We assume that the cube is used to store data in a distributed manner across multiple bricks, using an arbitrary data redundancy algorithm. A prototype of such a system is currently being built by our team.

We quantify what percentage of the original bricks may fail before the system becomes unusable. We also quantify the degradation of network performance both within the cube and to external hosts as a function of brick failures. The results show that a 3D mesh-based cube can remain operational until about 40% of the bricks fail, assuming sufficient inter-brick network bandwidth and sufficient connectivity to a cube's surface by external hosts or clients. Finally, we quantify the reliability of the system as a function of brick failure rates. By building bricks based on commodity part, one can design a 3-D mesh-based cube that should remain operational without maintenance for 3 - 7 years with 11% - 25% brick over-provisioning.

1 Introduction

The complexity of managing large and complex servers ranks high among the difficult issues facing the managers of modern information technology systems. Ambitious projects are underway to automate some of the low-level and routine operations which presently are performed by system administrators and field engineers. One of these projects is the “xxx” effort underway at xxx.

A key aspect of this system is the “Deferred Maintenance” or “Fail in Place” model, which assumes that by over-provisioning a system or adding additional resources while operating, maintenance can be delayed for a long time, possibly for its entire lifetime of the system. The software is responsible for automatically invoking spare resources as components fail. The only maintenance tasks humans are asked to perform is to physically add resources if either the original over-provisioning was insufficient or if there are additional - possibly unexpected - capacity demands. An important side-effect of automatic invoking of resources is that capacity scaling for varying demand is easily accomplished by just adding more resources.

This paper presents detailed quantitative insights into the consequences of the “Deferred Maintenance” acrchitecture for important system parameters, such as capacity, internal and external bandwidth, and connectivity requirements. To keep the scope of the investigation manageable, the paper makes certain important assumptions about the system under study. These are:

1. The system is build up from networked processing nodes called “bricks”.
2. All the bricks in the system are identical and contain processing, networking and storage functionality and have little internal redundancy.
3. A given brick is either completely functional (live) or completely inoperative (dead).
4. The system may operate as a storage server, connected to one or more external ‘host’ systems.
5. The bricks are stacked on top of each other, forming a 3D pile or “Cube”

6. Bricks communicate with next-neighboring bricks forming a 3-dimensional communication grid or mesh (see figure 2).
7. Because of brick failures, the mesh can have holes in it.
8. External hosts are connected to the bricks only at the surface of the cube
9. Storage data is replicated across two or more bricks, thus greatly increasing the probability that failure of a brick will not lead to data loss.
10. The details of the data distribution are not central to this paper. Rather, they are parametrized by the number K of distinct bricks over which a given dataset is distributed. This may range from K=2 for simple mirroring to K=14 for certain advanced RAID schemes [5]. BETTER REFERENCE?
11. Certain assumptions about correlated failures are made. Specifically, it is assumed that it is more likely that bricks which are stacked directly on top of each other ("in a column") are slightly more likely to suffer simultaneous failure, because they share power and cooling in our prototype system. (Interestingly the results show that taking those correlated failures into account has little effect on most of the results presented in this paper unless either the cooling or power systems are highly unreliable.)

The main body of this work is concerned with the *spatial* effects of failures. A typical question might be how many good bricks may be inaccessible because all their neighbors have failed. This question can be answered without a notion of time. Only in the last section, *time* is introduced through the metrics of brick and system reliability. This translates the results of the main body of the paper, for a given brick reliability, into how long a given system can be operated before maintenance action is required.

The idea of improving data availability in storage systems by deferred maintenance has been around for some time. As disk drives became more unreliable per byte and relative to the rest of the system, approaches like [5] were proposed to survive their failure. In these approaches replicas or parity-coded data was stored on different disks. When one of the disks failed, the system reconstructed lost data from redundant information thereby allowing uninterrupted access, although at significant performance degradation in some cases. Other schemes were proposed [6] that allowed more than one disk to fail without data loss. However, in order to keep data availability sufficiently high it was necessary to have spare drives (either in advance or by hot swapping the failed drives) to restore redundancy. This approach has proven popular as evidenced by the large market for RAID storage and is employed in many high performance storage products like the IBM ESS [ref] XXX. By relying on being able to "hot-swap" failed parts while the system ran uninterrupted, capacity and bandwidth over-provisioning was kept to the minimum necessary to survive subsequent failures that might occur within the service interval. A similar approach has been taken in building other reliable systems like network switches [Cisco ref].

In the approach proposed in this paper, higher availability is achieved by being able to flexibly move data from one brick to another in response to a failure. This is possible due to a high-bandwidth (3D mesh) network between the bricks. While, in the past, there have been other mesh based interconnect systems [2], none of them to our knowledge have focused on the problem of reliably storing data. Their primary focus has been on parallel compute applications. When a node fails in these systems, failed jobs are restarted or restored from the last check-pointed state stored on a separate storage system. While it might appear that communication networks [Modiano, ...] built out of mesh (or toroid) topologies of switches face similar problems (dealing with failed nodes) as the storage system described in this paper, there are major differences. They all focus on the problem of routing around failures and finding optimal routes. While reconfiguring links incrementally is required for our system too, it is not the problem being addressed in this paper. Secondly, they typically assume an all-to-all communication pattern between nodes. This is different from the patterns seen when hosts (clients) access a storage system (servers). This is a critical problem in this system since the hosts connect to surface (or edges) bricks only, and the loss of surface bricks can render host data unavailability even when there is no real data loss.

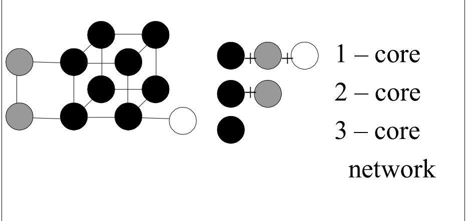

Figure 1: n-core networks

The problem of over-provisioning the capacity and bandwidth in a brick-based storage system was first described in [Scott]. That work discusses over-provisioning requirements by restricting usable bricks to those that are connected to at least two (2-core) or three other bricks (3-core). Those estimates have two drawbacks: Firstly, they are too conservative in their calculation as they do not take into account the mitigating effects of data leverage the data redundancy schemes. With these it is possible to use 1-core bricks. Secondly, in some pathological cases they are not conservative enough. An example of such a case is when 3 (equal sized) groups of bricks are connected via a single 3-core brick. The failure of that brick can partition the system into islands each incapable of continued operation.

Thirdly, the previous paper did not take into account the fact that some bricks may fail simultaneously (due to power or cooling system failure).

The main result of this study is that for all, but small Cubes, these systems are very resilient to a large number of brick failures. As much as 40% of all bricks may fail[1] before good bricks become isolated. This is a consequence of the 3-dimensional network connectivity which provides many alternative paths between bricks. Once more than 40% of bricks fail, the system degrades rapidly with additional failures. Lower dimensional networks don't fare so well, and our simulations show for example that the threshold for 2D mesh networks is around 20% failed bricks.

Other important conclusions are:

1. The 40% failure threshold, above which the system works well, is insensitive to the details of the implementation and the level K of the storage data redundancy scheme selected.
2. The results are insensitive to the absolute size of a cube, provided it is at least 4×4×4 bricks or larger.
3. The total internal bandwidth in a cube degrades quickly with brick failures, implying the need for over 80% over-provisioning of link bandwidthto sustain 40% failed bricks.
4. The total external bandwidth is not very sensitive to brick failures.
5. Multiple connections between a host and cube's surface bricks are essential to guard against brick in-accessibility.
6. With realistic brick reliability (commodity components) and over-provisioning assumptions, it is possible to build a system which needs no service action for about 7 years.

Section 3 shows how RAID systems of various levels can be implemented in a cube. Section 4 discusses the quality of the degraded network using a variety of measures. Section 5 discusses the connectivity of a cube to the external hosts. Reliability analysis is presented in Section 6.

2 Definition of Terms

Term	Definition
Brick	The basic building block to build large Cubes. Each brick has a processor, memory, optional storage devices and network connections to its six neighbors.
N	Total number of bricks in an Cube.
Cube	A heap of bricks arranged in a regular 3D mesh.
n-core	Set of all bricks that have degree >= n, i.e. at least n neighbors.
Set-k, chunks	A redundant data placement scheme where data is encoded into *k* chunks and placed on k distinct bricks each with a distinct network route to a surface brick. This allows data to be reconstructed even when one of the chunks is inaccessible.
failure domain	A set of bricks that may be affected due to a single failure. Failure domains may be formed due to physical arrangements (e.g. tower) or due to network connectivity.
tower	A column of bricks in the z-axis that share common power and cooling; failure of the power supply or cooling system will render the stack of bricks inoperative (this failure domain is specific to our particular hardware design).
Host or Client	Any application server that needs service from a cube.
Surface brick	A brick that has at least one of its faces exposed to the outside. Hosts connect to surface bricks.
Host connection	A network connection between the host and a surface brick. It is the network connection through which all requests and responses from the hosts reach the bricks. Hosts may have more than one host connection to deal with surface brick failure.
Dangling brick	A 1-core or 2-core brick that is in danger of being inaccessible due to a subsequent failure. Bricks connected by a linear chain are all dangling.
Unusable bricks	Bricks that cannot be used for placement. There are two reasons a brick may be unusable - either it is dead or it is inaccessible to other bricks.
RAID	Redundant Array of Independent Disks: a variety of methods to store data on multiple disks in such a way that it can be reconstructed should one or more disks fail.
Diameter of a network	The diameter of a network is defined as the longest distance between two nodes.

[1] This paper assumes that a brick is either completely functional or totally inoperative. For a storage server, where disk failures are the most common failure modes, this model is very conservative. In the first actual cube system being built, where each brick contains 12 disks, the failure of one or more disks will only reduce this brick's capacity by a corresponding amount, but otherwise not affect the functionality of a brick.

Term	Definition
Average distance in a network	The average distance of a network is the sum of the length (measured in hops) of all shortest connections between all nodes of the network divided by the number of those connections.
Reliability $R_{brick}(T)$ of a brick	Probability a brick will perform correctly for a time period T.

3 Placement of data in a degraded cube

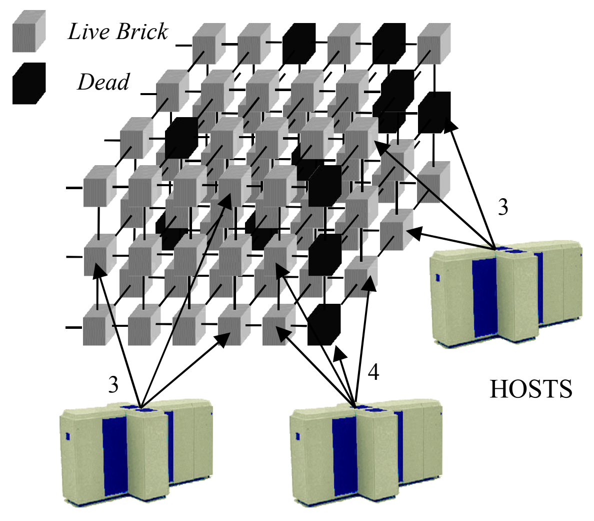

Figure 2: A 3D - mesh network. The column of three dead bricks on the front right face of the cube is an example of a failure domain. In the physical IceCube realization, bricks which are stacked on top of each other are slightly more likely to fail simultaneously, because they share a power bus and a cold-rail.

A new and pristine cube originally contains *N* bricks arranged in a 3D-mesh network (see figure 2). When used as a storage server the bricks run storage software that organizes and manages the disks in each brick. Although bricks are built to be fairly reliable they have insufficient *internal* redundancy to meet the higher data availability requirements of storage applications. Consequently, this requires that the storage software replicate data across several bricks in order to survive brick failures. Simple schemes like mirroring, where replica of data are stored on multiple bricks to more complex schemes like parity-based encoding [RAID reference, Salomon Reed Reference] can be used.

This paper is not concerned with the details of such redundancy codes. Instead, it assumes that the codes may create between 2 chunks of information (named set-2 code for this paper) for simple mirroring up to a maximum of 14 different chunks of information using some redundancy schemes. The *K* different data sets need to be stored as follows in order to guarantee that the data can be recovered after the failure of one brick or of all the bricks stacked on top of each other in a tower

1. No two chunks may be stored on bricks in the same failure domain (same tower).
2. After the failure of a brick tower at least K-1 sets must be reachable by surface bricks[2].

For each cube with some broken bricks we find the largest network of bricks that are connected to each other and to at least two surface bricks and get an estimate on the number of bricks that can be fully used by a set-K encoding. We use a conservative estimation process described in the Appendix that works well for larger networks but fails to fully utilize severely degraded cubes with less than 40% live bricks. As we show such systems are close to unusable as most of the live bricks in such a system are no longer accessible at all.

In [Scott] the authors argue that only bricks that have at least 2 or even 3 connections to the rest of the network (2-core and 3-core networks) should be considered for placement. For comparison we have included graphs for those networks in most figures. All the graphs shown in this paper are the results of simulating 300 degrading cubes of size 6×6×6 (216 bricks) unless otherwise noted. The error bars shown in some graphs represent the standard deviation.

Figure 3 shows how many live bricks can actually be used. We plot the fraction of surviving bricks along the x-axis and the number of usable bricks along the y-axis. A pristine cube is the data point plotted at the upper right hand corner. As bricks fail, the number of live bricks in the system decreases. In the graph, the lines for 3-core and 2-core networks show that quite a few of the running and reachable bricks are prematurely discarded, unlike the lines for set-2 through set-14 which show that placement is still successful and as good as a 1-core network. In fact, as live bricks decline from 100% to 60% the 1-core and the even the set-14 placement are able to use close to all live bricks, and the 2-core and 3-core connected subnetworks are too pessimistic. Only after losing more than 50% of the bricks is there a significant amount of live but unusable bricks (the difference between the dotted line and the graph). Note also that for planning reasons one should look at the lower part of the error bars and not the statistical mean, which makes a real difference after losing more than 50% of the bricks.

[2] This assumption means that in certain cases the layout of the data on the bricks must be known by the host in order to reconstruct the data, which is the case in all client-side network redundancy protocols [reference to network raid paper]

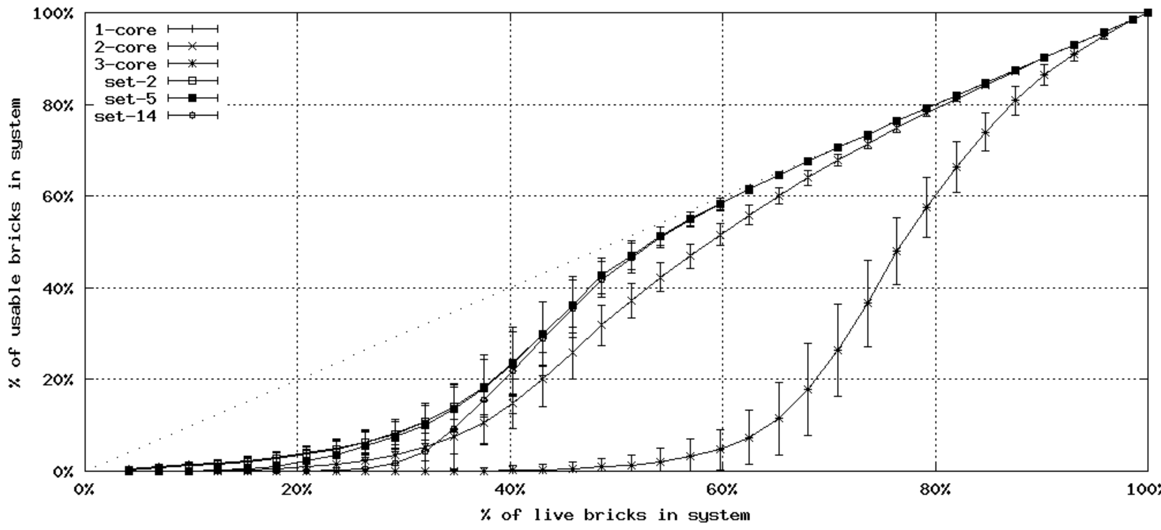

Figure 3: This graph shows how many live bricks are usable in a partially degraded cube, as bricks may be 'live' (fully functional) but may not be 'usable' because they are isolated by dead neighbor bricks or for other reasons.

However, it is not immediately obvious how the results for the 6×6×6 cube can be generalized to larger and/or smaller cubes. The graph in Figure 4 shows the 1-core and set-14 lines for four different cube sizes. It shows that for cubes larger than 5×5×5 we can lose as much as 50% of the bricks while still using most of the running bricks even for set-14. Smaller cubes could only support smaller set sizes though (with set-12 being the limit for a 4×4×4 and set-7 for a 3×3×3 cube). From those graphs we conclude that a cube can safely be used until it loses at least 50% of its bricks.

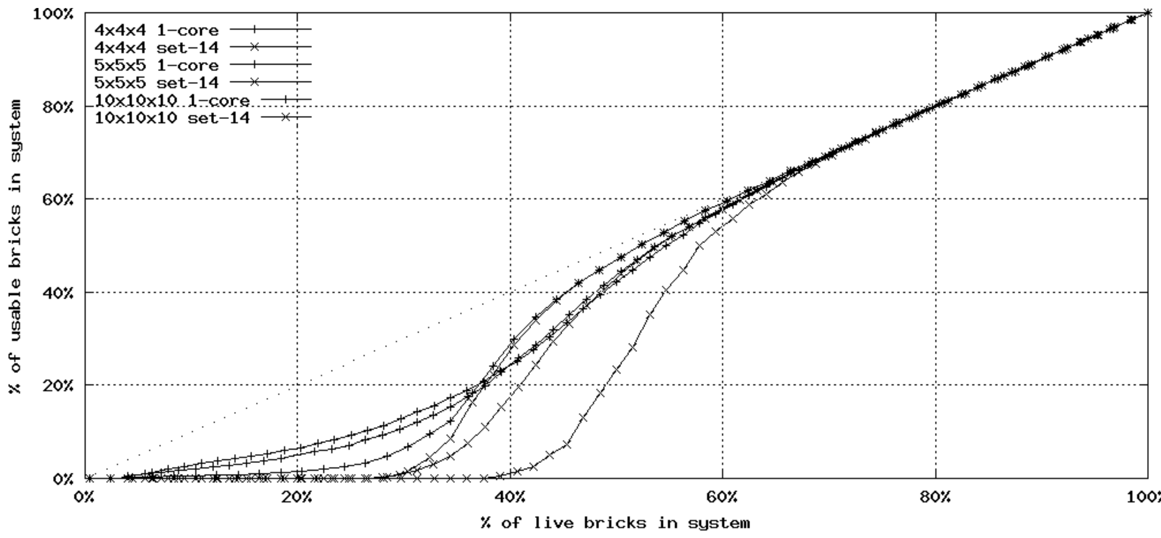

Figure 4: The relationship curves between live and usable bricks for a range of system sizes and redundancy assumptions. Except for a complex, 14-way redundancy scheme running on the smallest cube (bottom curve), all curves are surprisingly similar and indicate that a cube can lose as much as 50% of all bricks without too many live bricks becoming unusable.

The graph in figure 5 compares a 2D mesh of size 15×15 with cube of size 6×6×6 and shows that in a 2D the number of unusable bricks grows much quicker. Interestingly the separation point between set-14 and 1-core sub-cluster happens at about the same time (around 45%), but from the early decline and the much higher deviation at an earlier point we conclude that a 2D network requires about 80% live bricks to work adequately.

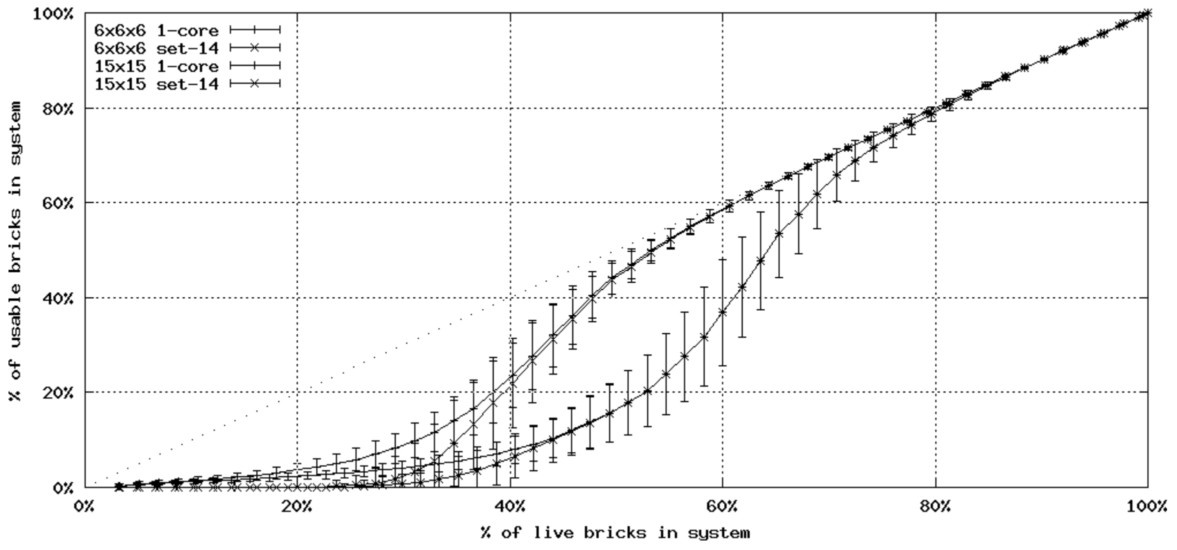

Figure 5 This graph compares degrading 2D and 3D mesh networks of roughly the same size and shows that the 2D network degrades much quicker as expected.

4 Network Quality in a Degraded Cube

We will now analyze the 1-core connected network, which is, as shown in the previous section was fully usable even for set-14 unless more than 50% of the bricks die in a larger cube. All the following graphs compare 1-core, 2-core and 3-core networks and some include hypothetical "optimal cube" network that assumes that the remainingN' usable bricks can be reorganized into a perfect cube size of $h=\sqrt[3]{N'}$.

4.1 Distances within the cube

We begin the network quality analysis by looking at the maximum and average distance between two bricks in a degraded cube as shown in figure 6 (the maximum distance is also known as the diameter of a network). For the perfect cube those numbers can be calculated as shown in table 1 (see the appendix for how the average distance is calculated):

Bricks	N
Height	$h=\sqrt[3]{N}$
Diameter	$d=3(h-1)$
Average Distance	$d_{avg}=h-\frac{1}{h}$

Table 1 Diameter and average distance in a pristine 3D mesh

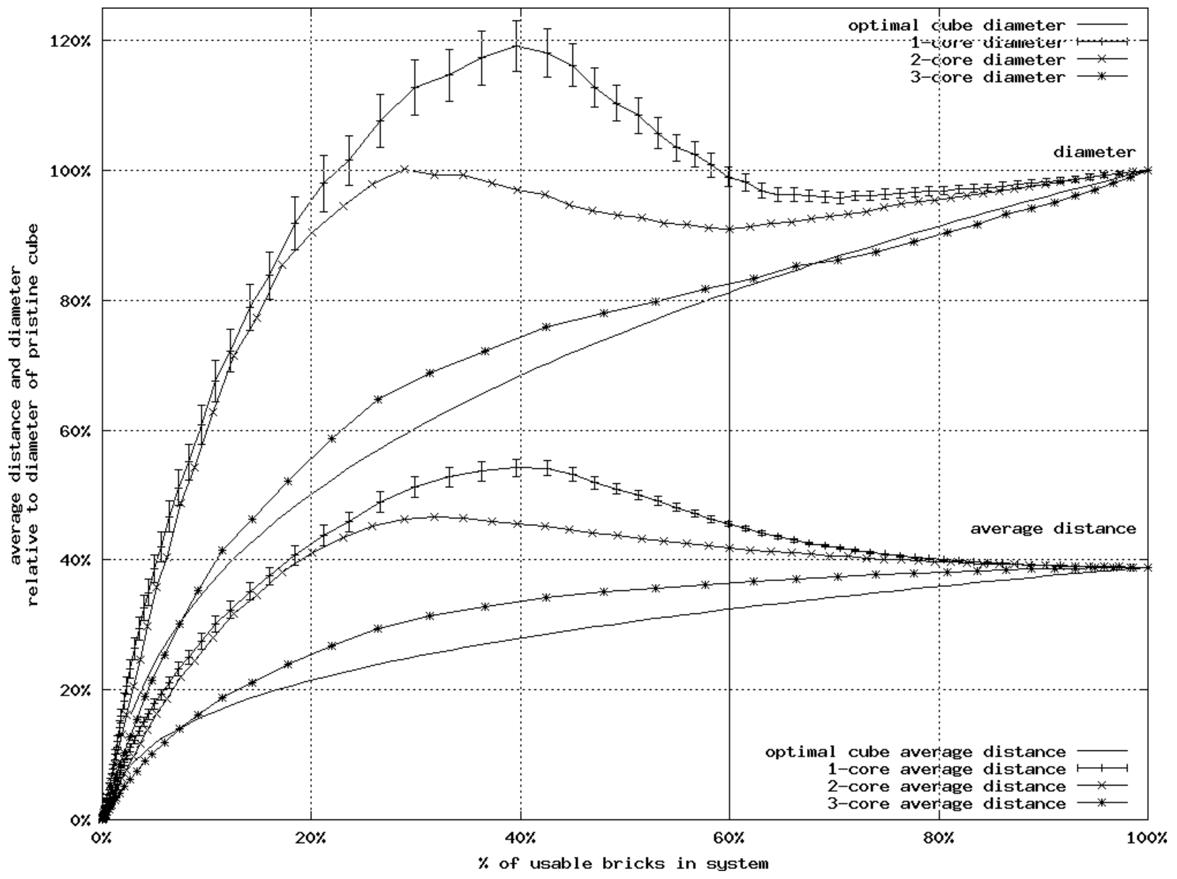

Figure 6: The diameter and average distance between bricks of a cube initially stay the nearly the same as bricks fail, but start to grow below 60% live bricks since messages need to travel more complicated routes. When most bricks have failed, the cube diameter shrinks again.

Looking at figure 6 we see that the diameter for the network stays at or below the diameter of the pristine cube, and the distance grows only a little, as long as we have at least 60% usable bricks. However, once we cross the 60% barrier both numbers go up dramatically (20% higher diameter, 30% higher average distance). Note that the *x-axis* in those graphs (and all following figures) show the number of usable bricks for a given network, not the number of live bricks (some of the live bricks will be ignored as they do not have sufficient connectivity to the live sub-cluster of the network).

From Figure 6 we can conclude that although the number of network hops increases in a degraded cube the increase can be ignored as long as at least 60% of the bricks remain usable. After the 60% line, the distances between nodes (average and maximum) start to grow significantly. This puts the barrier slightly higher than if we only look at placement possibilities as in the previous section, where we got a lower limit of 50% usable bricks.

4.2 External Bandwidth

The external bandwidth of a cube is defined as the number of bricks that are accessible from the outside; i.e. all the bricks on the surface of a cube. Figure 7 shows that the external bandwidth decreases more or less linearly for the 1-core connected subnetwork and the available bandwidth is over 90% that of a perfect cube of the same count.

This result suggests that the number of surface bricks lost is proportional to the total number of bricks lost. Figure8 plots the relative number of surface bricks against the usable bricks and we see that the relative number of surface bricks in fact only declines once we cross the 60% threshold of usable bricks. Not surprisingly the 3-core network shows a steep decline of surface bricks due to the fact that any surface brick that loses two neighbors can no longer be part of that network, which in turn puts its own neighbors at risk as they just lost one working neighbor.

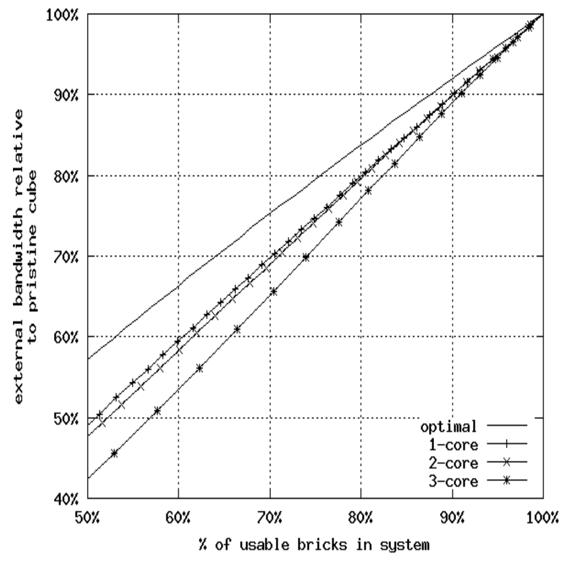

Figure 7: External bandwidth is the total bandwidth through the surface of the cube. It depends nearly linearly on the number of total usable bricks.

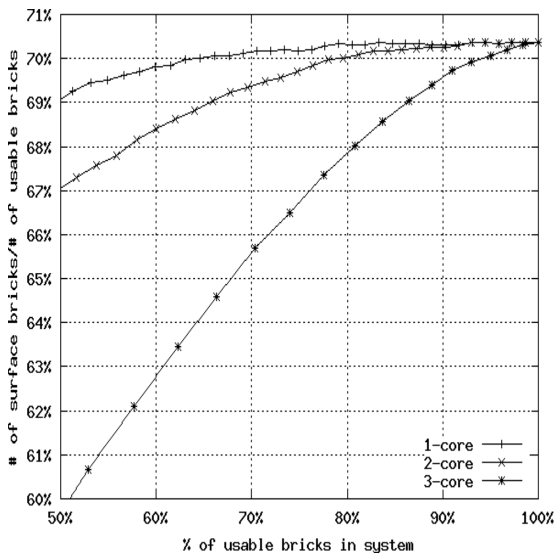

Figure 8: Comparing surface bricks to usable bricks show that the relative number of surface bricks declines only weakly as the cube degrades, unless only 3-core bricks may be used.

4.3 Internal Bandwidth

The total internal bandwidth of a cube is proportional to the total number of connections between usable bricks. Figure 9 shows that the total internal bandwidth decreases quickly as the cube degrades. At 60% usable bricks, only about 40% of the original internal bandwidth remains for the 1-core connected sub-network, which is about 30% lower than the optimal – a full cube of the same count. However, there are also fewer nodes.

Total bandwidth	B
Average distance	d_{avg}
Average bandwidth per brick	$b_{brick} = \frac{B}{N\, d_{avg}}$

Table 2 Average bandwidth per node in a cube.

We calculate the available bandwidth for brick-to-brick communications as shown in Table 2 by dividing the total bandwidth through the average distance between bricks and the number of bricks. Figure 10 shows that there is only half of the bandwidth per brick available when only 60% of the bricks are usable. This means that the pristine cube must be considerably over-provisioned with internal bandwidth. (See the conclusion for some detailed over-design calculations).

4.4 Redundant Connections

The number of redundant connections in a network is defined by the number of links that may be removed until the network degenerates to a tree spanning the bricks. By definition, trees have zero redundant connections. Table 3 shows how to calculate the redundant connections in a general graph and for cubes. Figure 11 shows that the number of those connections drops dramatically with the number of usable bricks, which, together with the fact that the average distance actually grows while bricks fail, explains the decline of the internally available bandwidth seen in Figure 9.

5 External Connectivity

We will now take a look at connectivity. So far we assumed that hosts can always access the live brick sub-cluster if it contains at least one brick on the outside of the cube. This would require every host to have connectivity to every single brick on the surface of the cube, which is neither cost effective nor practical. Instead, assuming a host has a particular number *c* connections to the cube surface, we calculate the probability that a host loses its connection to the sub-cluster. To calculate the probability we apply the multinomial distribution to get the expression shown in Table 4.

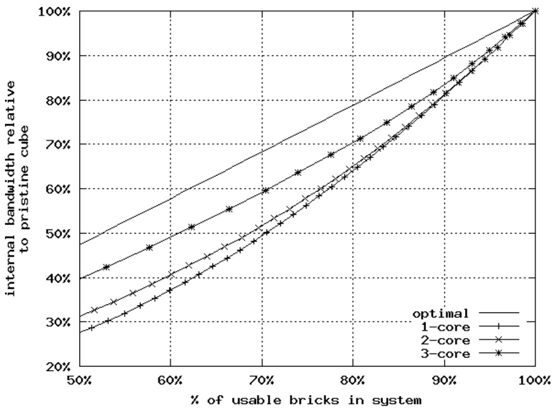

Figure 9: The total internal bandwidth drops as a cube degrades and becomes small when a substantial number of live bricks become unusable. This is exactly what one would expect.

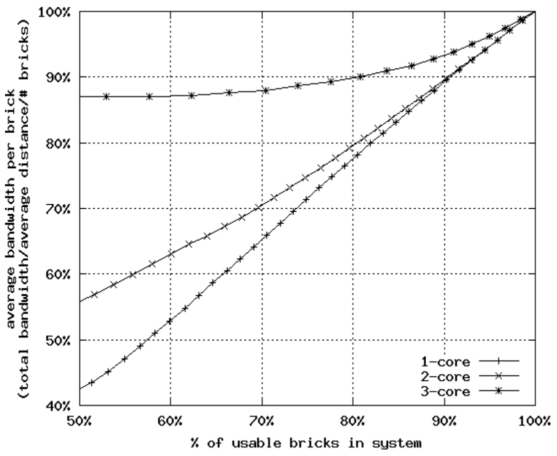

Figure 10: The average internal bandwidth per node declines substantially as the cube degrades.

Internal links	l
Redundant connections in a graph	$r = l - (N-1)$
Connections in a perfect cube	$l_h = 3(h^3 - h^2)$
Redundant connections in a perfect cube	$r_p = 2h^3 - 3h^2 + 1$

Table 3 Calculating redundant connections

Host connections	c
Live surface bricks	g
Usable bricks	N
Surface bricks	$s = 6\sqrt[3]{N^2} - 12\sqrt[3]{N} + 8$
Probability of having a sub-cube that does not include a brick with host connection	$r = \frac{(s-c)!(s-g)!}{(s-g-c)!s!}$

Table 4 Risk

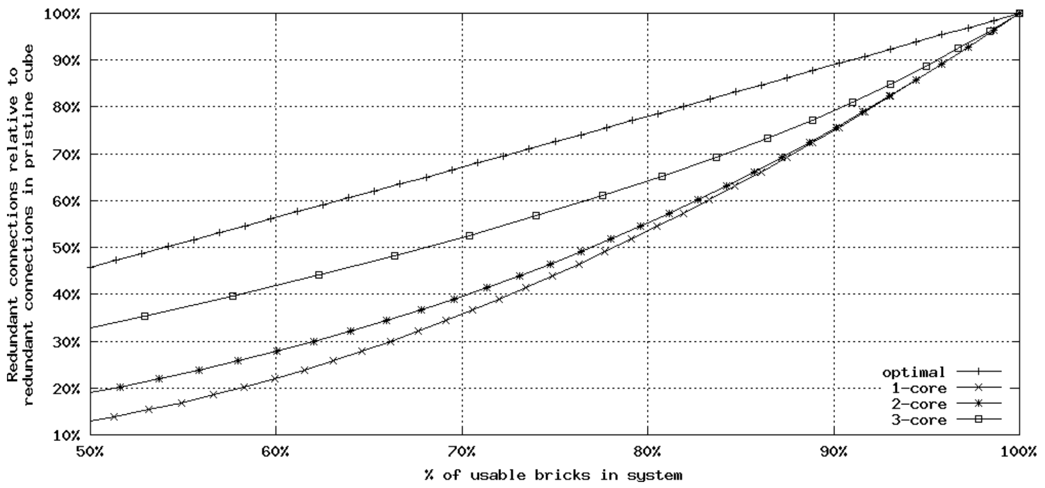

Figure 11: This plot shows the number of redundant connections in a degrading cube. They decline quickly as brcks fail. This explains why the internal available bandwidth declines so fast.

From Figure 8 we know that the number of surface bricks is proportional to the number of usable bricks as long as we do not lose more than 40% of the bricks. We present two graphs, the first one, Figure 12, shows the number of live bricks plotted against the probability of losing connection to the live sub-network for a variety of host connections *h*.

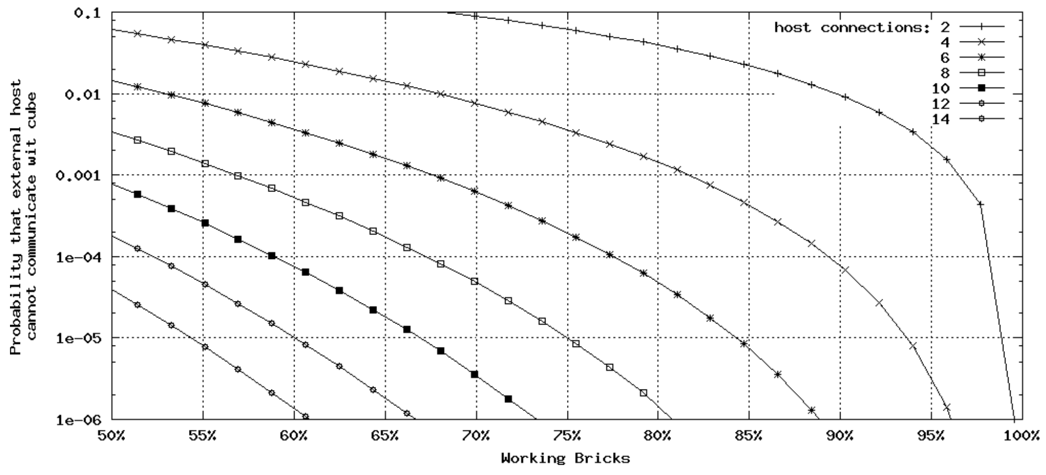

Figure 12: This graph plots the probability of a host losing connection with a degraded cube. The paramtere is the number of host connections *c*. The probability decreases about exponentially with *c*.

The second graph, Figure 13, shows the same number, but plots the risk against the number of host connections for 60% of good bricks. It also includes numbers for cubes up to a size of 8000 bricks. We note that the risk depends mostly on the number of host connections and not the size of the cube. In order to lower the risk significantly, the number of necessary host connections grows quickly into the 10 to 15 range unless the user of the cube is willing to reconnect hosts to different, non-dead bricks. From Figure 13 we can conclude that either a switch is needed between the surface bricks and the hosts, or the hosts need to be part of the cube itself, which moves the surface connectivity problem to the clients of the host.

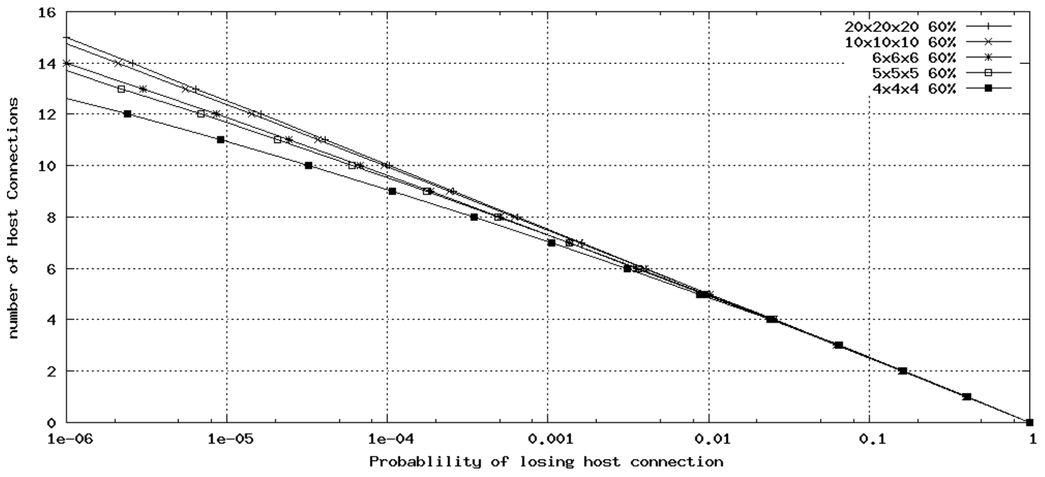

Figure 13 Plot shows how many host connections are required as a function of the size of a (degraded) system and acceptable probability of losing a host connection.

6 Reliability

The operating model for a cube system is that it can run continuously without replacement of failed bricks. Failed bricks remain “in place”– at least until the next service or maintenance interval or when the cube is de-comissioned. New bricks can also be added to a running system, as necessary. To insure a particular mission or maintenance duration for a system at a particular capacity or performance target, a system can be over provisioned with bricks at its start of operation. An alternative possibility is to add more bricks as necessary.

To determine how much over provisioning may be necessary, we plot the relationship between the elapsed time by which a given percentage of a cube's original bricks fail (assuming no new bricks are added) versus the brick reliability. This operating duration can be viewed as the minimum maintenance interval, or "mission time" for a system. This assumes identical bricks with constant, memoryless, uncorrelated, failure rates and implies over provisioning assuming no bricks are added; so for the brick failure rate λ (per time unit) the brick reliability per time unit is given by

$$R_{brick}(T)=e^{(-\lambda\cdot T)}$$

In general, the standard binomial distribution gives the probability that M-out-of-N bricks will survive a mission of duration T, when the reliability of each brick is $R_{brick}(T)$:

$$R_{cube}(T)=\sum_{i=0}^{n-m}\binom{n}{i}R_{brick}^{n-i}(T)(1-R_{brick}(T))^{i}$$

This assumes perfect brick failure detection and isolation, and operating software that properlyrebalances the utilization of bricks. For this calculation, we assume a 6x6x6 cube and an arbitrary system hardware reliability target, $R_{cube}=0.99999$, and then calculate the brick reliability needed to achieve that. This optimistically assumes no other possible failure scenarios (software, environment, etc.)

Assuming four overall brick percentage failure rates, (N-M)/N, of 10%, 20%, 30%, and 40% of the initial good bricks, the needed reliability or probability of brick success over mission time T is 0.962, 0.897 0.817, and 0.734, respectively. In figure 14, we show the mission time T needed to achieve these four brick reliabilities, or overall system percentage failure amounts.

For example, if one allows up to 20% overall failures in a system (25% over provisioning), bricks with a annual brick failure rate of ~1.5% imply a maintenance or successful cube mission duration of nearly 8 years.

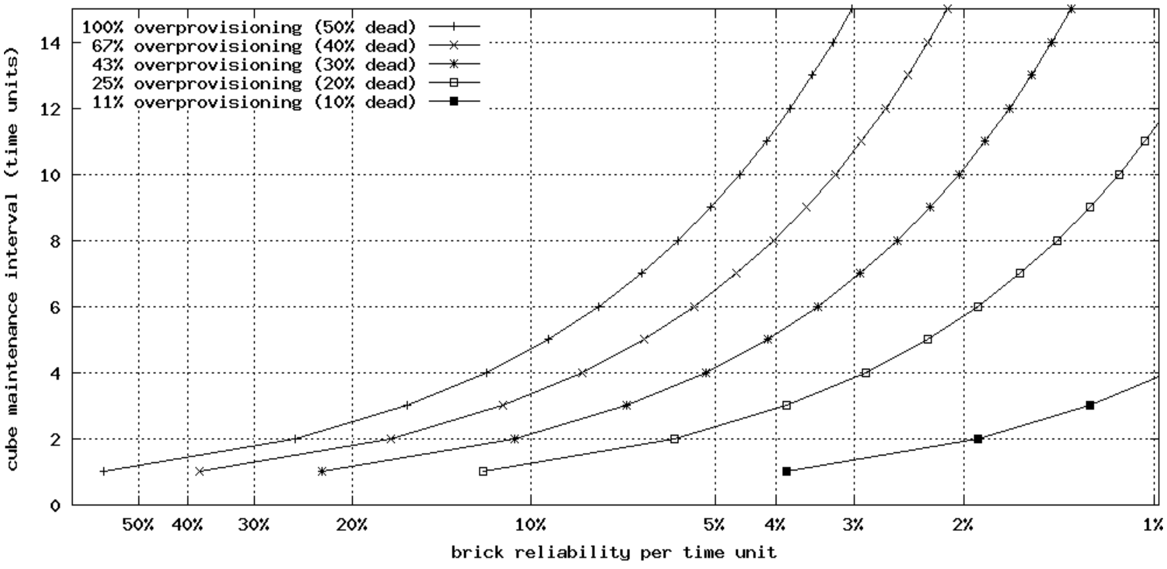

Figure 14: This plot shows the necessary maintenance intervals for a cube reliability of 0.99999 as a function of the individual brick failure rates and the number of tolerable brick failures..

7 Conclusion

We looked at a variety of failure scenarios in 3D mesh based cube and concluded that such a system can continue to operate successfully until about 40% of the bricks die. For this worst-case assumption one has toover-provision a cube with 67% bricks assuming no new bricks are added during the lifetime of the system. If we know the reliability of the bricks, we can use the results of Section 6 to estimate the maintenance interval of the cube. For example, if 10% of the bricks fail within a year, Figure 14 shows that a cube can successfully operate for over 4 years before it requires maintenance (with 0.001% probability of losing more than 40% of the bricks).

However, while there is sufficient externally bandwidth available in such a case, the internal bandwidth will be quite a bit lower than in a pristine cube once 40% of the bricks have failed. This means that the network load in cube will grow when it degrades as the total internal bandwidth and the average internal bandwidth both decline at a steeper rate than the number of bricks. If we need to build a system where the average internal bandwidth will stay above a given minimum, we can calculate the necessary bandwidth over-design. According to Figure 15, the average internal bandwidth must be over-designed by about 90%. This means that the applications running in a new system should run acceptably fast with a network load of less than 50% as the load on the network will double once 40% of the bricks have failed.

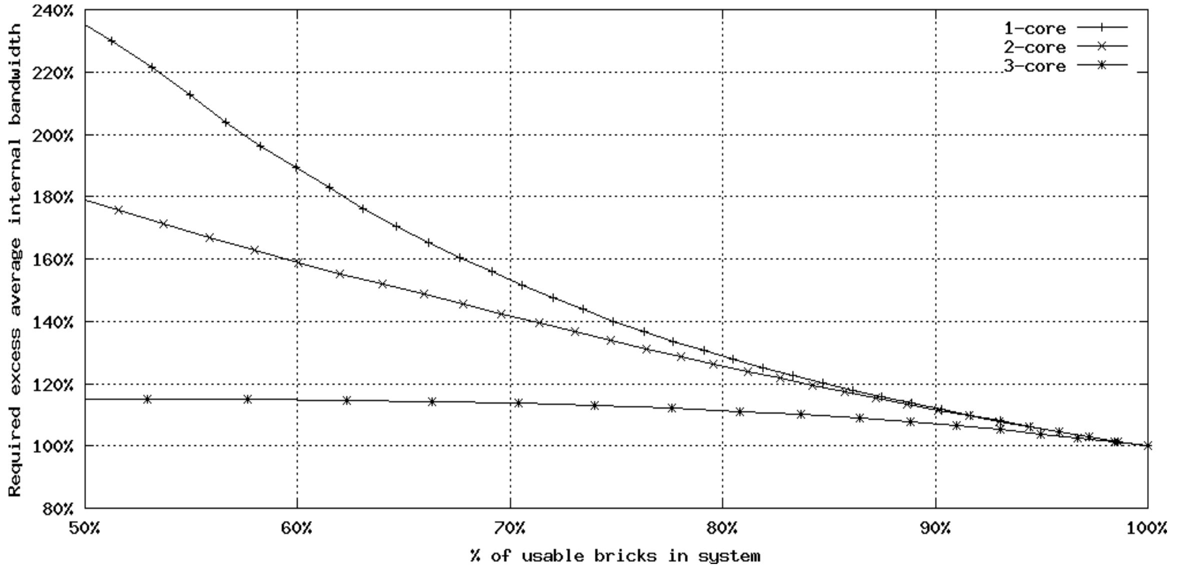

Figure 15 this plot shows how much the internal bandwidth of s system has to be over-designed in order to keep a minimum bandwidth after a number of bricks have failed.

8 References

[1] S Kirkpatrick, W. Wilcke, R. Garner, and H. Huels. Percolation in Dense Storage Arrays, In Physica A, 2002.Also presented at the Messina Symposium, "Horizons in Complex Systems," Dec 5-8, 2001.

[2] R.E. Kessler and J.L. Schwarzmeier. "Cray T3D: A New Dimension for Cray Research" In Digest of Papers, COMPCON Spring '93, pgs 176-82, San Francisco, CA, February 1993.

[3] Jun Sun and Eytan Modiano, "Capacity Provisioning and Failure Recovery in Mesh-Torus Networks with Application to Satellite Constellations," LIDS report 2518, MIT, Sept. 2001.

[4] M. Clouqueur and W. Grover, "Availability Analysis of Span Restorable Mesh Networks", IEEE Journal on Selected Areas of Communication, Vol. 20., No. 4, May 2002.

[5] RAB. The RAIDBook: A Source Book for RAID Technology sixth edition. The RAID Advisory Board, Lino Lakes MN, 1999.

[6] M. Blaum, J. Brady, J. Bruck, J. Menon. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. IEEE Transactions on computers, Vol. 44, No 2, pp. 192-201, February 1995.

[7] T. Pinkston and S. Warnakulasuriya. Characterization of Deadlocks in K-ary n-cube Networks. IEEE Transactions on Parallel and Distributed Systems, V.10, No. 9, September 1999

9 Appendix

9.1 Estimation of usable bricks in a degraded cube

Definitions:

K	number of chunks for the encoding.
n	number of connected and live bricks in a given network.
f	number of failure domains for this network.
max_f	number of bricks in the largest failure domain.
splits	number of failure domains that would generate disjoint networks if all the bricks in that failure domain failed.
hidden	number of bricks that may lose the connection to the surface if all bricks in some failure domain fail.
failsafe	number of failure domains that do not contain bricks that may get disconnected from the surface due to a failure of another brick.

We progressively refine our estimate in several steps:

1. If *(K < f)* we cannot use the cluster at all as we need at least *K* failure domains to safely place *K* data sets. The number of usable bricks in this case is 0.
2. If *(hidden=0)*, which means that no working brick will be disconnected from the surface no matter which failure domain dies, we only have to make sure that all K chunks of a dataset are stored in different failure domains. In order to accomplish this we need for each brick in the largest failure domain at least K-1 additional bricks (see Figure for a possible mapping). So for *hidden=0* we conclude that we can use all bricks if $max_f \cdot K \leq n$.

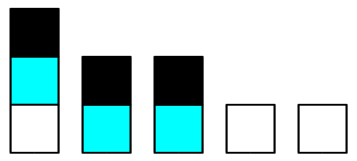
Figure 16: set-3 mapping onto 9 bricks in 5 failure domains

3. If *(hidden > 0)* we have to make sure that we can place all data sets such that if some failure domain fails and disconnects some live bricks from the surface we can still access at least *(K-1)* chunks. As a worst-case scenario we assume that all bricks that may lose contact with a surface brick will do so if the failure domain with the largest number of bricks fails. We basically assume that this failure domain contains *(max_f + hidden)* bricks as shown in Figure 17. Note that if the failure domain with the most bricks contains some hidden bricks we count those bricks twice. This makes the estimate more conservative but if we would not count them twice one can easily show that the following estimate is wrong in some corner cases.
4. As we lump all hidden bricks into one failure domain we effectively reduce the number of failure domains to failsafe. In order to be able to place any data we must therefore assume that *(K ≥ failsafe)*, and abandon placement completely if that condition is not true.

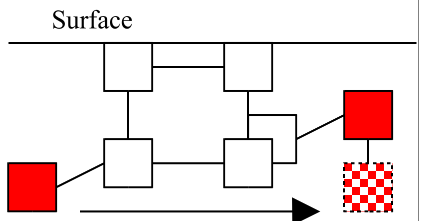

Figure 17: Assume that hidden bricks are connected to largest failure domain

5. As before we need for each brick in the largest failure domain an additional *(K-1)* bricks in order to do a successful placement, which means that $K \cdot (max_f + hidden) \leq n$. If this inequality holds we can safely assume that we can use all live bricks.
6. If above inequality does not hold, we may have to ignore some of the hidden bricks. Assuming that *x* is the number of hidden bricks we use, the following must hold (we simply correct the number of bricks in the failure domain and the total number of bricks and we cannot ignore more bricks than the ones that are hidden, unless we know exactly how many bricks are in each failure domain so that we know how many bricks have to be removed in order to make the largest failure domain smaller):
$$(max_f + x) \cdot K \leq n + x - hidden \wedge x \geq 0$$
$$\Rightarrow x = min\left(\frac{n - hidden - K \cdot max_f}{K-1}, hidden\right)$$
7. If *(x≥0)* we can use *(n-hidden+x)* of the bricks for the placement.

We note that this is a lower estimate on the number of bricks that can be used for a certain encoding. Especially for networks with low connectivity, it will considerably underestimate the number of usable bricks.

9.2 Average Distance in a 3D-Mesh

The shortest path between two nodes N1 and N2 in a regular 3D network is the sum of the differences of the x, y and z coordinates: d=(|N1x-N2x| + |N1y-N2y| + |N1z-N2z|). In a regular 3D cube (where all sizes have the same length *h*) the average minimum distance between two randomly chosen nodes is the same in all three directions, which means that we only need to calculate the average distance in one direction and then multiply it by three. The total distance from a node at position k to all other nodes is given by

$$\sum_{i=1}^{k} i + \sum_{i=k+1}^{h-1} (i-k)$$

By dividing this number though *h* we get the average distance of one particular node to all other nodes in that direction. By summing up all average distances of all nodes in one direction and dividing the result by the number of nodes we get the average distance between two randomly chosen nodes:

$$\frac{1}{h}\sum_{k=0}^{h-1}\left(\frac{1}{h}\left(\sum_{i=1}^{k} i + \sum_{i=k+1}^{h-1} (i-k)\right)\right) = \frac{h}{3} - \frac{1}{3h}$$

By multiplying the average distance in one dimension with 3 we get the final result:

$$d_{avg} = h - \frac{1}{h}$$