



ELSEVIER

Physica A 314 (2002) 220–229

PHYSICA A

www.elsevier.com/locate/physa

Percolation in dense storage arrays

Scott Kirkpatrick^{a,b,*}, Winfried W. Wilcke^b,
Robert B. Garner^b, Harald Huels^c

^a*School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel*

^b*IBM Almaden Research Center, San Jose, CA, USA*

^c*IBM STD, Mainz, Germany*

Abstract

As computers and their accessories become smaller, cheaper, and faster the providers of news, retail sales, and other services we now take for granted on the Internet have met their increasing computing needs by putting more and more computers, hard disks, power supplies, and the data communications linking them to each other and to the rest of the wired world into ever smaller spaces. This has created a new and quite interesting percolation problem. It is no longer desirable to fix computers, storage or switchgear which fail in such a dense array. Attempts to repair things are all too likely to make problems worse. The alternative approach, letting units “fail in place”, be removed from service and routed around, means that a data communications environment will evolve with an underlying regular structure but a very high density of missing pieces. Some of the properties of this kind of network can be described within the existing paradigm of site or bond percolation on lattices, but other important questions have not been explored. I will discuss 3D arrays of hundreds to thousands of storage servers (something which it is quite feasible to build in the next few years), and show that bandwidth, but not percolation fraction or shortest path lengths, is the critical factor affected by the “fail in place” disorder. Redundancy strategies traditionally employed in storage systems may have to be revised. Novel approaches to routing information among the servers have been developed to minimize the impact.

© 2002 Elsevier Science B.V. All rights reserved.

PACS: 5.50.+q; 81.05.Rm; 89.70.+c

Keywords: Percolation; Transport in networks; Information systems; Reliability; Highly parallel computing

* Corresponding author. School of Engineering and Computer Science, Hebrew University, Jerusalem, Israel. Tel.: +972-2-658-5838; fax: +972-2-658-5261.

E-mail address: kirk@cs.huji.ac.il (S. Kirkpatrick).

It is encouraging to see that technological challenges still require scientific understanding. We would like to present an exciting technology direction that fundamentally involves, in fact stretches, our understanding of percolation in regular networks, an area that Gene Stanley has been very much involved with.

A confluence of powerful technologies, getting smaller, faster, and, most important of all, cheaper every year has made it possible to assemble previously unthinkable amounts of computing power and data storage in very small volumes. The difficulties of communicating between them and to the outside world are being attacked by “switching fabrics” made up of very simple but very fast local routing processors. We are working on “Ice Cube [1]”, a dense array of storage and switches built up as a 3D cubic array of simple building blocks—“bricks”. The switching fabric coexists with the storage, connecting each brick with its immediate neighbors. While we are focussing on the storage array application, the extensions of this framework to other mixtures of cpu-intensive or communications-intensive function are evident, and require solving many of the same problems.

For concreteness, consider eight of the 150 GB high volume low cost disks (2.5'' in diameter) that the storage industry will be able to deliver in the next few years, packed into a cube roughly 15 cm on a side, with a cpu and a fast (e.g. InfiniBand™ [2]) 1 GB/s (or 8 Gbps) switch. The switch connects each little “brick” only to its neighbors in a cubic pile of such bricks. The aggregate numbers are awesome. A $10 \times 10 \times 10$ array of these bricks will hold 1.2 PB (petabytes) of data, in a volume of a little over 3 m^3 . Even if each brick is reading or writing data to the outside world at 200 MB/s, for a total of 200 GB/s, that bandwidth can be provided by the switches on just one face of the array. The total internal and external bandwidth of the array is 12 TB/s, although it is far from clear how to use all of that at the same time. There are four significant problems to be solved to make this array a reality: get power in; get the heat out; manage the communications; and make the whole assembly reliable, always available and inexpensive to maintain. The first two require some clever ideas not discussed here—we will talk about the last two challenges.

Reliability in storage arrays starts by requiring that some or all of the data be stored in at least two well-separated spots, to insure that after the failure of one storage unit the data can still be recovered. A simple and robust strategy is to create a mirror copy of each block of data that is written into the array, placing each copy in a different brick. A write transaction would then involve a block of data arriving at the IceCube at a surface, being routed to a brick in the interior, its mirror copy being sent to a second brick in the interior, and an acknowledgement message going back to the sender, probably through the surface site that received the initial write request. All this has to be done in a distributed manner, with the selection of storage locations determined locally according to some global rule that does not require central coordination. It is possible that we might want to write three copies of some blocks of data. This will permit the data in the array to survive two simultaneous brick failures. In the event of a single failure, two copies will remain available so that there will be no delay in accessing data while failure of a single unit is being logically repaired.

What happens if a brick fails and its storage, its switching capability, or both become unavailable? Availability requires that the repair be accomplished quickly. The

requirement of low maintenance cost (and common sense) make it clear that pulling the array apart to change a bad brick in the interior is not an option. In fact, the most frequent outcome of trying to fix a malfunctioning element in a complex computer system is that new problems are created. So we shall have to plan for bricks to “fail in place”, with the faulty components powered off, their data re-mirrored, and all further communications routed around the failed units. It is likely that only 5–20% of the bricks in a large $10 \times 10 \times 10$ system will fail over the 5-year useful lifespan of one of our IceCubes, to the extent that they cannot communicate with their neighbors. But as a worst-case design point, we shall consider the possibility that as many as half the bricks might fail in place. Such a deterioration would be a small factor compared to the technical obsolescence that Moore’s Law inevitably brings, as long as the whole system continues to deliver its originally promised performance.

A possible problem is percolation, a phenomenon studied in physics but quite novel in engineering. The central idea is that various amounts of missing elements fixed in position in a network may have a dramatic effect on the network’s ability to transport electricity, fluid, or in this case, information. If we model failures as complete elimination of both data and communications in a brick, we have precisely the site percolation model [3].¹ There will be a well-defined percolation threshold [3], a fraction of good bricks below which all the bricks remaining are connected to each other only as small clusters, each not large enough to span the Cube or guarantee to reach the surfaces for I/O. Experience studying site percolation on 3D simple cubic (SC) lattices tells us that the array with half of its bricks out of service is still far above the percolation threshold, which is 0.31, so bricks which are perfectly good but cannot be reached because their neighbors are dead will occur only rarely. But Fig. 1 shows that other consequences of percolation must be considered.

We need to insure complete independence of the two locations in which each block of data is written. If all paths to one copy of our data lead through the brick in which the other is stored, or if all paths from the surface to both copies go through the same site, they are no longer independent. Failure of one brick can cause the loss of both copies. The precise definition of a set of independent bricks so that data can be mirrored is that we should use the largest biconnected cluster [4] of bricks, that cluster in which each pair of bricks has two entirely distinct paths connecting them. Well above the percolation threshold, almost all sites which can be disconnected from the largest biconnected cluster at unique “articulation points” occur in short chains, dangling off the main cluster. The “2-core”, found by eliminating any brick with only a single working neighbor until no more such bricks remain, gets rid of all dangling ends. We see from the 2-core data in Fig. 1 that in our 50% failed IceCube, another 10% of the bricks lie in dangling ends, and thus are at risk of being disconnected by the failure of another brick if that brick is in the chain of which they are part. This fraction is not available for us to use in mirroring data unless we take account of its extra dependencies.

¹ There is a large literature on the theory of percolation on graphs and its applications. See Ref. [1], for a coherent presentation.

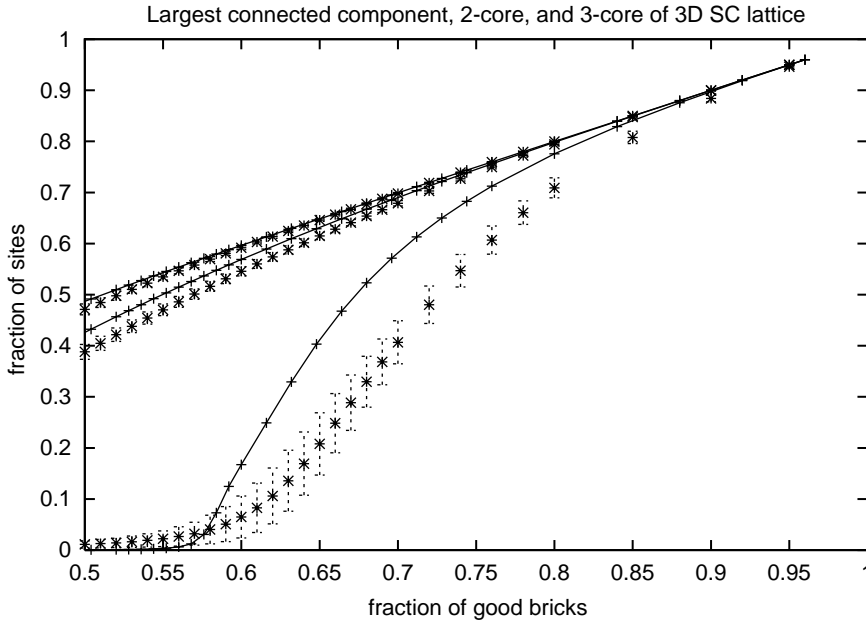


Fig. 1. For two sizes of 3D simple cubic lattices, as a function of the fraction of good bricks remaining, we show the fraction of bricks in the largest connected component (the “percolation fraction”), in the largest 2-core, and in the largest 3-core. The data points connected by lines are for a $50 \times 50 \times 50$ lattice; those with error bars are for a $10 \times 10 \times 10$ lattice. The uppermost points are the percolation fraction, followed by the 2-core, then the 3-core (defined in text). The size of the 3-core is far more sensitive to the fraction of good bricks than is either of the other two characteristics.

What can percolation concepts say about attempts to provide triple redundancy in a 3D array? This could fail if there are parts of the working array which can be disconnected from the rest by the simultaneous loss of two bricks. The 3-core, the largest cluster remaining after removing all sites with two or fewer neighbors until no more such are found, provides an upper bound on the fraction of the array in which we can guarantee immunity against double failures. Fig. 1 shows that this is falling rapidly in size and has almost disappeared before the 50% point. In fact, the 3-core of a random graph (one in which every node has the same probability of being connected to any other node) disappears with a first-order transition [5], that is, its fraction of the system volume drops abruptly to zero over a very small range of failure rates, if the system is sufficiently large. If the 3-core in a 3D lattice behaves similarly, its disappearance must occur at a higher fraction of good units than the percolation threshold. The problem has been studied, mostly by simulation, over the past decade [6],² with the assumption that the transition for the 3-core of the SC lattice is second-order, but occurs at a much higher fraction of good bricks (roughly 0.57) than the percolation threshold.

² In the physics literature, the problem is called “bootstrap percolation”. For a review, see Ref. [4]. Data on the 3-core of the simple cubic lattice can be found in Ref. [4].

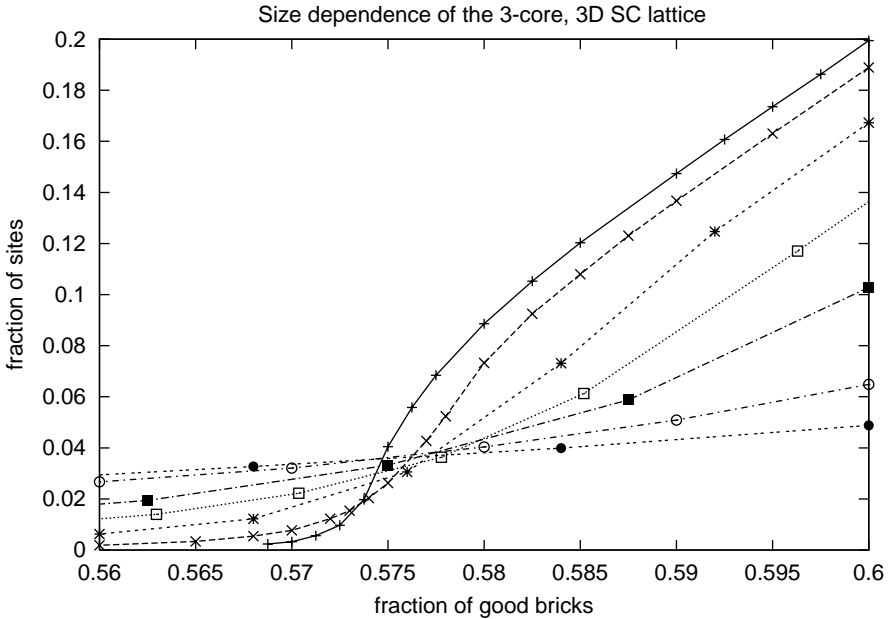


Fig. 2. The 3-core threshold on a finer scale. The data shown are for 3D SC lattices with linear dimensions $L = 5, 10, 20, 30, 50, 100$, and 200 , reading from bottom to top on the right-hand side. 10,000 samples were taken for each data point up to $L = 30$, then 1000 at $L = 50$, 500–1000 samples for $L = 100$ and 20 samples per point for $L = 200$. Note that the finite-size changes seem to involve two different stages, one for $L \leq 50$ and a different evolution for $L \geq 100$, a point which Branco and Silva [5] also note.

Fig. 1 shows that this is the case, so in order to offer triple redundancy we may tolerate relatively few in place failures, perhaps only 20%. If we look at the 3-core transition on a finer scale (Fig. 2), two observations emerge. The finite-size effects are extremely strong (see the effect for the $5 \times 5 \times 5$ array with relatively low fractions of failed units), and have a rather unusual appearance for a second order transition. It is not clear without further analysis of the data whether a first order transition can be ruled out. For the very largest samples (much too large to be possible IceCubes), the limiting form of the fraction of bricks in the 3-core could have a discontinuity.

Effects of the finite size of our $10 \times 10 \times 10$ lattice are significant. (To a percolation expert, 1000 units in 3D is a pretty small sample, exhibiting behavior far from the limiting forms seen for large lattices of a million or more units.) Bricks on the surface of the IceCube, where all I/O to the outside world takes place, have fewer neighbors and thus are more easily disconnected from the main cluster. Fig. 3 shows the fraction of bricks on a surface which are part of the largest cluster, for $20 \times 20 \times 20$, $10 \times 10 \times 10$ and $5 \times 5 \times 5$ brick Cubes with various fractions of their units working. The reduction is greater as the samples get smaller—a greater fraction of bricks lie in corners or on edges of the cube, where they are more convenient for I/O but have fewer neighbors and are thus more easily detached from the main cluster. Also the fluctuations become

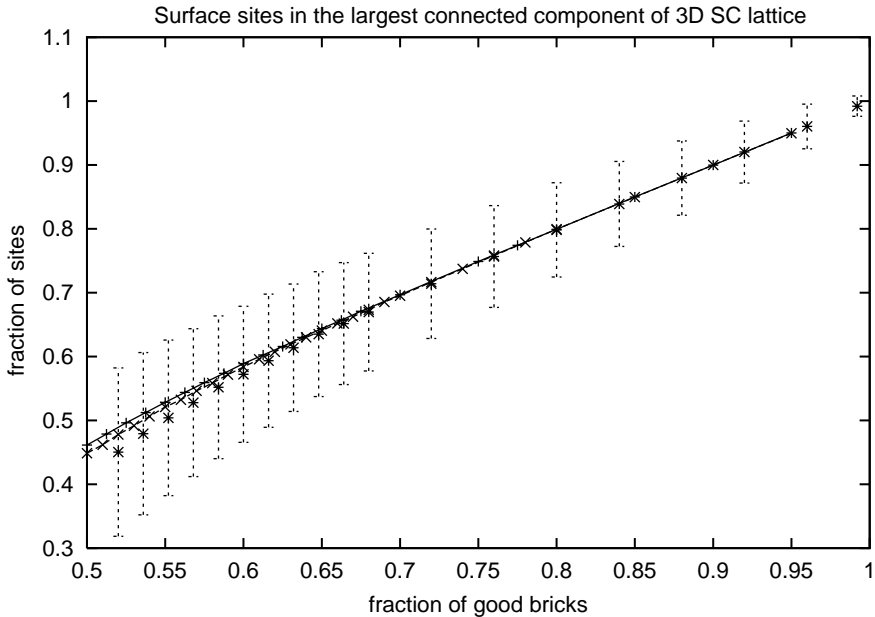


Fig. 3. Fraction of top surface bricks that lie in the largest connected cluster for various sizes of IceCubes. The data are $L = 20$ (upper line), $L = 10$ (lower line) and $L = 5$ (data points with errorbars). In this characteristic, the size dependence is small, but the fluctuations from one sample to the next are of practical import.

more significant. Not only must the average IceCube perform properly, but those in the tails of the distribution will also belong to paying customers! It appears, from Fig. 3, that it may be necessary to add extra I/O sites on more than one surface to serve the smaller arrays reliably, or to provide toroidal boundary conditions by linking opposite faces of the Cube in the directions normal to the face at which I/O takes place. Both effects, of course, are just the finite size corrections to scaling long known for percolation theory and other phase transitions.

A larger effect than the connectivity decrease is the decrease in overall bandwidth that comes from routing traffic around failed bricks, through fewer and fewer available paths as the number of failures increases [7]. To estimate this effect, we can calculate the maximum number of parallel paths that exist connecting opposite faces of the partly faulty array. By a duality argument, this is given by the minimum number of good links that a surface (not necessarily flat) crossing the sample must cut, or the min-cut. Fig. 4 shows the min-cut in 3D site percolation as a function of the fraction of good bricks remaining. When half of the bricks remain in operation, the bandwidth for data messages that must cross the array is reduced to only one tenth of its initial value, because of the crowding of messages through a few hot spots. A third effect, the increase in path length due to avoiding failed bricks, only becomes large close to the percolation threshold, and does not seem to be significant in our design regime with 50% or more good units [7].

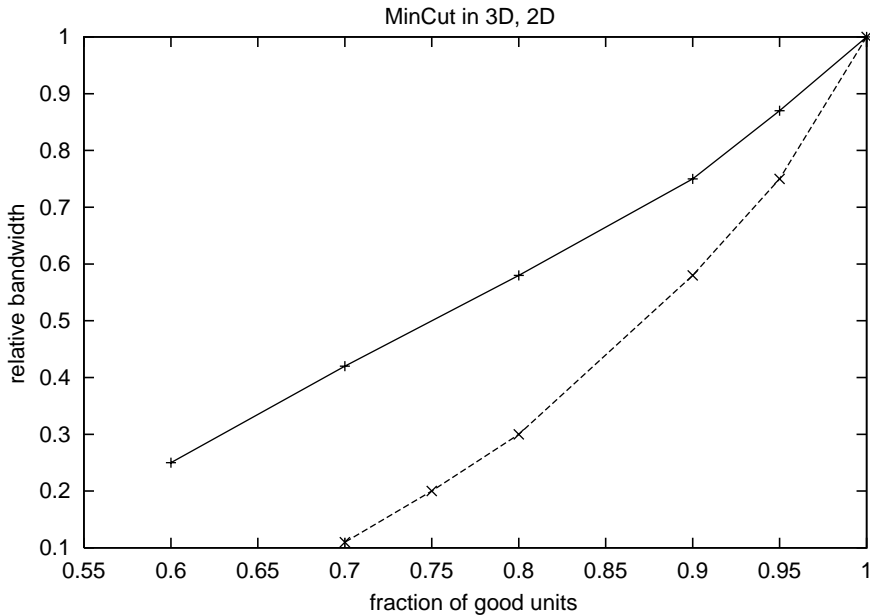


Fig. 4. Min-cut, the smallest fraction of links that must be cut by a continuous, but not necessarily flat, surface passing normal to one axis of a 3D SC lattice (upper line) or a 2D square lattice (lower line). This is a worst-case measure of overall bandwidth reduction due to brick failures. Actual observed bandwidth may be even lower because of hot spots in message traffic.

While fast flow management algorithms exist for centralized management of messages to an inhomogeneous network, this approach is not acceptable because it would expose a single point of failure, and it is not even likely that future switch chips will be managed in terms of messages. Most are designed for “destination routing”. As a packet begins to arrive, its header is scanned for a destination address. This is looked up in a forwarding table, which tells which output port should transmit or buffer the packet, and the packet bits then begin flowing to the output port. When they reach the next switch chip in the path, the process is repeated. All cleverness in the chips’ routing of messages to avoid bottlenecks has to be encoded in these forwarding tables.

A classic linear algorithm due to Dykstra [8] finds the shortest path, measured in nearest neighbor steps, from a source brick to any other brick in the site that can be reached. It is simply breadth-first search: in the first step, we list all sites that can be reached in one hop from the source; in the second, all sites that can be reached first in one more step from the first list, and so forth. As the paths are discovered, we build a list of first steps that lead along minimum-hop paths. These are not unique—for the perfect cube there can be up to three possible first hops; for the disordered cube up to six first hops may lead by a minimum-hop path to a distant site. We shall select one of these for each destination as our forwarding table entry. Keeping a memory of the alternatives permits fast distributed recovery in the frequent cases where a failed turtle can be avoided by switching some destinations’ first hops to alternatives which still

provide a minimum hop path. The forwarding tables can be filled independently of one another without causing infinite loops because all paths generated are minimum-hop—in effect the messages are always flowing “downhill”.

We have explored two natural ways of selecting one of the alternative first steps for the routing table. The *XYZ* rule, a canonical ordering, is to always move first in the *x*-direction if possible, then in the *y*-direction if possible, then in the *z*-direction. The random rule is to pick one of the possible first steps at random. The first rule is one traditionally used in non-adaptive mesh routing. The second appeared likely to handle high levels of missing bricks more gracefully.

To see how these two approaches compare in their sensitivity to disorder, we first define a traffic model. The description above of how a write operation would take place provides a good example. Let messages enter the IceCube at every brick on the top surface. From each entry point, we go once to each brick in the whole cube to write data, then to another brick in the cube to mirror the data, then back to the original surface brick for an acknowledgement. For an $L \times L \times L$ Cube, this model requires completing almost L^8 messages, and is a stylized simulation of writing to every brick in the array from all possible external sources. We simulated this traffic model on a $10 \times 10 \times 10$ IceCube array assuming various levels of failed bricks, using either the *XYZ* or the random rule to load the forwarding tables, and kept a record of the number of messages that had to pass through each buffer in the switching fabric. Our interest is in understanding the “hot spots” that form under this load.

We think that bottlenecks and worst-case loadings, not average loadings, will dominate the rate at which our array can store away this traffic load. Therefore Fig. 5

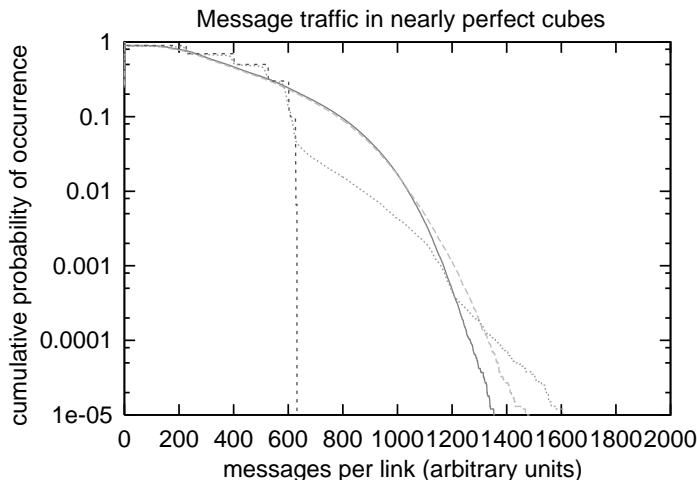


Fig. 5. Cumulative distribution of link loadings caused by the “write everywhere” traffic model in IceCubes with $10 \times 10 \times 10$ bricks and 0% or 1% rates of failure. The dashed line distribution stopping with a sharp step results from the *XYZ* model routing on the perfect cube. The dotted line is its generalization to 1% failure. This curve has its toe furthest to the right. The other two curves, which are smooth, are from using randomized routing with 0% and 1% failures.

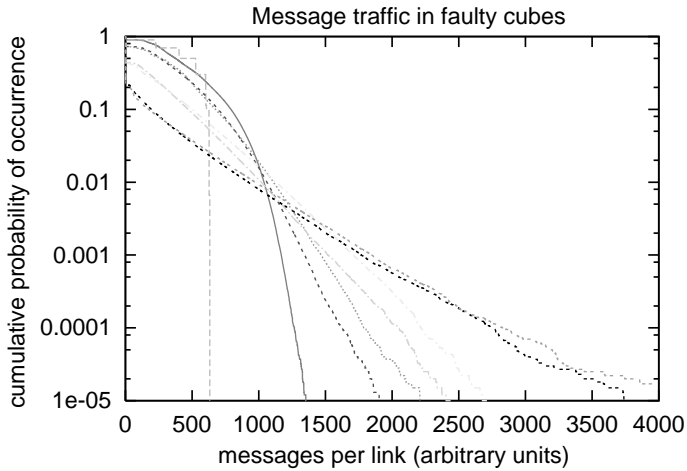


Fig. 6. Link loading distributions for 0%, 10%, 30% and 50% failed bricks, using both *XYZ* and randomized routing. The cases shown are (from left to right, at the bottom of the chart): *XYZ* (0%), Random (0%), Random (10%), *XYZ* (10%), Random (30%), *XYZ* (30%), Random (50%), *XYZ* (50%).

shows the cumulative distribution of buffer loadings when the fraction of good bricks is very close to 1. The *XYZ* routing approach produces a highly regular and balanced arrangement when routing on the perfect cube; its cumulative distribution function is a series of steps. The steps round a little for 1% failed bricks, and the *XYZ* routing develops quite a long tail of unusually loaded links. By contrast, the random routing strategy produces a tail of heavily loaded buffers even on an undamaged lattice, but with only 1% failed bricks, the random choice is better.

Continuing this analysis, Fig. 6 shows that the *XYZ* distributions of loadings are worse than the randomly generated tails for most levels of failure, except in the absence of any failures. At the other endpoint, 50%, the two routing strategies show similar performance, except deep in the tail, where the random assignment of first steps again does better. To convert these distributions (which were gathered on ten or more samples apiece) into actual performance estimates will take us beyond the aims of this paper.

One open problem at this point is what to do about deadlock. This is the concern that there may be cycles formed in which packets to unrelated destinations waiting for resources to be freed are in fact waiting for each other and will never get free. (Imagine a traffic intersection, with four cars stopped, each politely waiting for the car on its right to go first, etc.) There are methods for only slightly flawed networks to handle this problem, for example using the *XYZ* routing strategy with a sufficient number of virtual lanes but nothing that deals with the high levels of failure that we analyze here. There have been statistical studies [9] indicating that deadlocks in highly faulted networks are rare enough to not be a practical problem.

We thank Richard Freitas, Moidin Mohiuddin and Jai Menon for providing resources and stimulating discussions, and Harry Butterworth and Claudio Fleiner, as well as the entire Icecube team for their contributions.

References

- [1] See <http://www.almaden.ibm.com/cs/storagesystems/IceCube/>
- [2] See <http://www.infinibandta.org/specs>
- [3] D. Stauffer, A. Aharony, *Introduction to Percolation Theory*, Taylor and Francis, London, 1994.
- [4] S. Kirkpatrick, The geometry of the percolation threshold, in: J.C. Garland, D.C. Tanner (Eds.), *AIP Conf. Proc.* 40 (1978) 99.
- [5] B. Pittel, J. Spencer, N. Wormald, The sudden emergence of a giant k-core in a random graph, *J. Combin. Theory Ser. B* 67 (1996) 111–151.
- [6] J. Adler, *Physica A* 171 (1991) 453–470;
J. Adler, D. Stauffer, *J. Physics A* 23 (1990) L1119–24;
N.S. Branco, C.J. Silva, *Cond-Mat/9904239*, 1999.
- [7] S. Kirkpatrick, Percolation thresholds in granular films, in: S.A. Wolf, D.U. Gubser (Eds.), *AIP Conf. Proc.* 58 (1979) 79.
- [8] E.W. Dijkstra, A note on two problems in connection with graphs, *Numer. Math.* 1 (1959) 269–271.
- [9] T.M. Pinkston, S. Warnakulasuriya, Characterization of deadlocks in k-ary n-cube networks, *IEEE Trans. Parallel Distrib. Comput.* 10 (9) (1999) 904–921.