The Impact of Failures on Large Distributed Storage Systems

Technical Report UCSC-SSRC-07-10
August 2007

Maximiliano Mehech
maxmehec@soe.ucsc.edu

Storage Systems Research Center
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
http://www.ssrc.ucsc.edu/

This paper was filed as an M.S. project report in August, 2007. Max's M.S. advisor was Prof. Ethan Miller.

The Impact of Failures on Large Distributed Storage Systems

CMPS-296 – Master Project

Maximiliano Mehech
University of California, Santa Cruz
maxmehec@soe.ucsc.edu

1 Abstract

Distributed storage systems that can support high throughput and capacities in the petabytes range, are being built using standard Off the Shelf Components disks and networks to lower costs. Both the large number of components and the quality of these components contribute to a high probability of failures. The performance and complexity of storage networks built using standard topologies with data network components has been studied before, but the analysis of the effect of network and disk components failures on the performance and availability of the system has received limited attention. We analyze the effects of network components, links and disk failures on storage data networks connected with several common network topologies in the 4,000 nodes range using a simulation program. We compare both the average numbers and the peak numbers. Our results indicate that it is possible to build storage networks with high level of resiliency using several of these topologies and we analyze the possible trade-offs among them.

1 Introduction

Storage systems that can have up to thousands of disk drivers and storage capacity in the order of petabytes are quickly becoming a common presence in both research and commercial institutions. Cost is still an issue, although the use of standard of the shelf components coupled with the use of ubiquitous Ethernet networking technology has brought down the cost of building such systems to very affordable levels. But as the cost to build the system decreases, the use of thousands of off-the-shelf components brings the reality of multiple component failures in the system, whether they are network or storage related.

While disk capacity has been constantly increasing and price has fallen, performance and reliability have not accompanied the same trend of improved efficiency. Coupled with the use of powerful and ever faster networking technologies, now it is possible to design and build storage systems comprised of large clusters of disks with increasing data capacity and overall data throughput. On the other side, the time consumed on regenerating the data lost in case of a disk failure has also increased.

Recent research suggests that using data networks is an effective way to create a resilient and robust architecture for data storage systems that can hold large amount of data and provide high throughput access. We propose to analyze the effects of component failures on this storage networks in terms of performance impact both on average and on extreme scases. We will use a simulation program to collect statistics about how different failure scenarios affect the performance of different network topologies. We will see that many topologies can be built in ways to minimize the impact of failures both on the average performance and on the extreme cases using a simple load based, locally determined routing algorithm.

2 Related Work

Lustre [12] was one of the first clustered file system implemented that uses Object Oriented Storage and can scale into thousands of nodes. Its networking approach is a modular one that can be used with TCP/IP, Quadric Elan and Myrinet in addition to Ethernet. The effects of network or disk failures on the performance and reliability of Lustre have not been addressed yet.

Panasas offers a commercial implementation of an OSD with its ActiveScale Storage Cluster [9]. Panasas utilizes a standard TCP/IP over Gigabit Ethernet implementation to connect both clients and servers to the OSD system. Servers are connected using 4 aggregated gigabit connections to a single gigabit Ethernet switch where also the clients are connected. While this paper presents a real case study of the file system performance, it is clear that such performance is also limited by both the network design and the protocol configuration. The use of TCP/IP allows the use of other standard protocols such as iSCSI, RPC, DNS and NTP while at the same time permitting the use of other networking technologies like Myrinet and Infiniband. Their approach is directed to the study of the performance of the system and not on how failures would affect it.

Other industry solutions include arrays using SAN with Fibre Channel (IBM [23], EMC [21], Network Appliance [24]), Infiniband [7] (Mellanox) or Gigabit Ethernet (e.g. EqualLogic [22], Network Appliance [24]). These approaches rely on the inherent fault tolerance and failure detection methods from the

particular network technology or from the use of standard TCP/IP protocols [4].

Xin *et al.* studied the effects of failures on large interconnection networks for storage [20] but only analyzed a few topologies. This paper built on a previous study from Hospodor and Miller [6] that proposed the use of switched data networks to interconnect storage components and studied the cost, throughput and complexity of several network topologies but did not analyze the effects of component failures. Xin and Miller studied the effects of failures of OTSC disks on large storage arrays but did not look at network failures [19].

The study of resilience of networks includes routing algorithms and failure detection as the main focus to provide resilience but most of the studies are limited in the size of the network being analyzed. Many studies focus on fault tolerance and failure recovery for generic systems like Castro *et al*. in Byzantine fault tolerant systems [2] or for multiprocessors systems such as Vaidya *et al*. that studied how routing algorithms can deal with failures in an efficient way [14].

3 Basic Concepts

The increasing use of storage capacity in current supercomputer or high performance computing has created a paradigm shift in how to address and solve the storage problem. The advent of affordable and cost competitive storage area networks that are easy to install and maintain have pushed the industry into the direction of using specialized networks to achieve larger density and faster access to data. Technologies such as Fibre Channel, SCSI and lately IP SCSI (iSCSI) have helped mature the industry and the solutions available. But these technologies have limits on how much storage capacity they can manage efficiently and safely. Even at the current growth rate of single disk data capacity, the demand for additional storage can increase even faster.

The deterrents for using common data network technologies to connect data storage networks traditionally have been throughput, latency, reliability and cost, where the first three are the main components. With Gigabit Ethernet and 10 Gigabit Ethernet, the performance seems now acceptable. The cost has come down and will continue to follow the same trend. The standardization of iSCSI as a protocol to access storage across any IP network has allowed a more reliable way to build storage networks. Conversely, the availability of large of the shelf disks and switches not only allows for building large networks but also increases the probability of failures that could affect the overall availability and performance of the system.

3.1 Networks and topologies

Several topologies have been proposed and analyzed to implement storage networks. Miller and Hospodor explored several of them for use in large storage networks [6] and compared their cost, network throughput and complexity but not the impact that failures would have on their performance.

The topologies and designs used to create fault tolerance networks are very similar to the designs used to achieve large throughput. The use of redundant elements for connections, switches, routers, disks and servers to achieve high throughput are indeed very similar. The more resources that are available in the network the greater the performance the system will achieve. A single component failure (or a limited number of components) will have less effect on the overall performance and availability of the whole system.

Note the use of "overall" as a comparison metric. It could be argued that even if the average performance has not been degraded with a certain failure, some accesses will experience such long delays that could render the system unusable or the performance unacceptable. Thus it should be necessary not only to analyze the overall performance but also the extreme cases, to see how robust and resilient to failures the system really is. We analyze both the average values as well as the distribution over time or over quantity of components to watch for extreme cases.

3.2 Specific topologies studied

Some of the topologies have already been analyzed for their performance under failure [20]. We will analyze both additional topologies and also ones that were studied to compare the results. There are several basic network interconnect topologies that can scale into the thousands of nodes such as meshes, trees, hypercubes, butterflies and other variations of hierarchical networks.

The simplest connectivity would be achieved by connecting all servers directly to the storage but this will neither scale efficiently nor will it be resilient enough. The next alternative is a butterfly topology where links are distributed across an intermediate network in between routers and disks in order to create a single path between each router and each disk. In this case we are showing a butterfly topology with 4096 disks and 256 routers and an intermediate network composed of 1024 8-port switches with the last 256 switches having 12 additional ports to connect the disks. The main drawback is the lack of redundant routes from routers to disks. This has to be addressed by adding redundant

links in between switches in the same level to be used only in case of failure.

All the other topologies have redundant links to create redundant routes. Starting with Mesh 2D where every node is a switch plus a disk and is connected to its immediate top, down, left and right neighbors. This configuration uses 4 port switches plus a disk for each node and provides additional routes both for load distribution and for fault tolerance. The nodes are arranged in a 64x64 2D mesh and there are 8 routers that will connect to all nodes on two sides of the mesh. Drawbacks are first, that there are not enough links to allow great throughput and second, that the average length of paths are over 30 hops.

We analyze 4 architectures using Torus from 2D, 3D, 4D and 5D. A Torus is similar to a mesh with the addition of a link that wraps around the connection from the end of a row in a dimension to the beginning of that row in the same dimension. This allows for half the average length as the Mesh. All the Tori topologies that we analyze are composed of 3968 nodes with disks and 128 routers. The routers are distributed randomly inside the Torus because this provides better performance [10]. The 2D Torus is again a 64x64 mesh, the 3D is 16x16x16, the 4D is 8x8x8x8 and the 5D is 4x8x4x8x4. As the number of dimensions of the Tori goes up, the number of links also increases and the average length of the routes goes down. See Table-1.

The last two topologies studied are the ones with the most links. The purely hierarchical Fat Tree requires large switches and routers but uses only about half the number of links as the Torus 5D. We analyzed a fat tree with 4096 disks connected to 128 routers each with 64 ports connecting 32 disks and 32 switches. To create the Fat Tree we used 96 switches each one with 128 ports.

The topology with the largest number of links is the 12D Hypercube. Again we used a similar arrangement as the Torus where the routers connected to the servers are randomly distributed in the hyper cube. There are 3968 nodes each one with 12 ports and one disk and 128 additional routers with 12 ports. Each node in the hypercube has one connection to another node in each one of the other dimensions. This gives plenty of redundant links for load balance and fault tolerance.

3.3 Routing messages in the network

Since almost all the topologies mentioned above provide multiple physical paths between two given nodes we need to have a method for determining which path a message should follow. The first approach would be to use a deterministic routing algorithm where the path is determined by the source and destination nodes. The disadvantage is that this method does not take full advantage of the redundant links and it can not adapt to variations in network utilization patterns or component failures.

Adaptive routing uses the network resources more efficiently to detect congestion patterns or sense faulty components at every hop in the path. This data is used to decide where to forward the message next. Minimal adaptive routing limits the path available for selection to the ones providing the shortest path. This might not be possible in all cases, so a non-minimal adaptive routing, that can use routes that are longer than shortest path, is used in our case for routing when failures occur in the path of a message.

More sophisticated routing algorithms such as wormhole flow control or virtual channels are outside the scope of this study and can be done as further refinement.

3.4 Failures

We can divide the failure scenarios in a large storage network in three types:

- **Link Failure:** A link is a connection between two components of the system. When a link fails the greater problem is that there might not be another route to reach a certain point and the network might become partitioned. Good fault tolerance requires enough redundant links to allow several link failures without partitioning the network.
- **Node failure:** A node can be a router, a server, a switch or a storage concentrator. When a switch or router fails all of the links connected to it will also fail. In some topologies, a node failure could also partition the network. In general, it will cause an overload on adjacent nodes and on nodes along alternate routes.
- **Disk failure:** A disk failure will prevent the disk from replying to requests and to server data. Requests sent to that disk will not be replied and will timeout. Furthermore this will increase the load on other disks. A good fault tolerance depends not only on redundancy but also on an intelligent data distribution and duplication policy. Once a disk failure has been detected, if there are spare disks in the network, a recovery mechanism will be triggered that will re-image one of the spare disks. The replication process will in turn create additional load on the system beyond the normal redistribution of the requests destined to that failed disk.

In case a failure is detected, the requests will need to be rerouted around the specific component that has failed (except when it is a disk failure). This requires a routing algorithm that can discover alternate paths depending on failures and network status in general. For

example, if an alternate path is getting congested or if this alternate path has been used more than a different alternate path then the routing algorithm has to be able to choose which one is better for the overall system performance. Routing has to be done independently of other components in the network that have discovered the failure and based only on local information from the immediate accessible point to where the failure is.

4 Experimental Setup

4.1 Simulation description

We used a simulation program to test the behavior of traffic patterns in the different network topologies under different types of failures. The program was written in C++ in a modular way to allow the creation of different topologies with their proper interconnections and routing algorithms. The main building block is the node object that is the central part of the simulations. Nodes simulate switches and are connected to other nodes using link objects. Nodes can have up to two different types of objects attached: routers or disks. Nodes without any object attached are just switches that pass traffic like the ones used on the Butterfly and Fat Tree topologies. Routers are responsible for forwarding requests from servers into the network and to the appropriate disks. Our simulation does not include the servers, so routers will generate traffic simulating requests that should have come from servers. Disks, on the other hand, are responsible for processing these requests and sending the results. Some nodes can act as switches with disks attached like those needed on the Mesh, Torus and Hyper Cube topologies. Other nodes only act as a network interface for disks like those needed for Fat Tree and Butterfly topologies.

The system was programmed to simulate 8 different types of network topologies although many others could easily have been added:

- Hypercube up to 13 dimensions,
- Butterfly on 3 levels with configurable number of ports per switch and number of switches per level
- Fat Tree on 3 levels with configurable number of switches per level
- Mesh 2D with configurable number of switches
- Torus 2D through Torus 5D. Also the number of switches per dimension are configurable

The simulator accepts as a parameter which topology to run. The sizes of each topology are precompiled in advance. Each topology has its own procedures for initialization, link creation, requests creation and packet forwarding (routing).

There is a global queue for events and each link has its own queue for processing messages. Each disk has two queues, one to process read requests and another to process write requests.

4.2 Determining the right network load

Before running the failure simulations we needed to establish a load base to run on the network. We ran several iterations of simulations with no failures at different loads. We were looking for a load that would utilize the disks around 90% or that would utilize the links at a point when the collisions in the network did not exceed 50% of the messages. Some topologies can accept larger network loads but have to wait for the disks to respond. Other topologies can not carry enough load to keep the disks busy without saturating the links. In these cases increasing the load would not yield larger overall network throughput.

Another parameter that needed to be determined was the amount of time that each router would wait for a request to be completed. The request timeout had to be long enough as not to generate false lost messages (messages that would arrive normally but after the timeout triggering undesired retransmissions) and short enough as to be smaller than the period of time being simulated. The router timeout would become more important for the failure cases but we determined it using the base case with no failures.

4.3 Network topologies simulated

These are the configurations that were simulated (see Figures 1 through 6). Table 1 shows the number of switches, disks, routers and links for each topology.

Topologies	# switches	# disks	# routers	#ports	#links
Butterfly	1024	4096	256	16384	8192
Fat Tree	4192	4096	128	24576	12288
Hyper Cube	3968	3968	128	49152	24576
Mesh 2D	4096	4096	8	18160	9080
Torus 2D	3968	3968	128	16384	8192
Torus 3D	3968	3968	128	24576	12288
Torus 4D	3968	3968	128	32768	16384
Torus 5D	3968	3968	128	40960	20480

Table 1 - Number of components per Topology

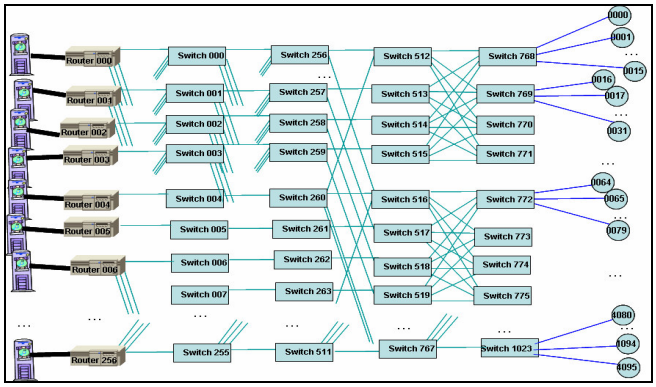

Figure 1 - Butterfly Network

Butterfly: 4096 nodes with disk on a switch with a single gigabit Ethernet link, 1024 switches with 8 gigabit Ethernet links, 256 routers with 4 gigabit Ethernet and 2 10 gigabit Ethernet links.

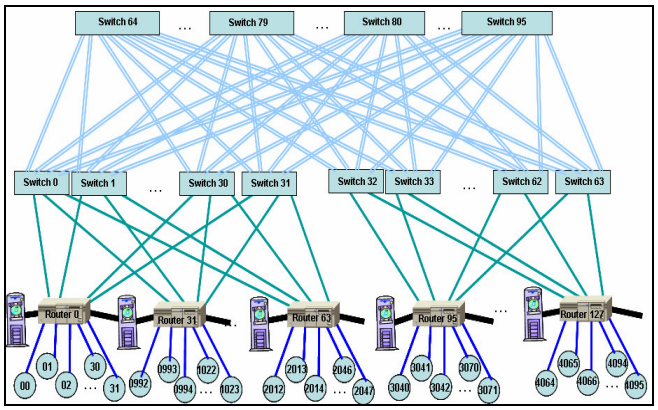

Figure 2 - Fat Tree network

Fat Tree: 4096 nodes with disk on a switch with a single gigabit Ethernet link in the first layer; 128 routers on the second layer with 32 gigabit Ethernet links to 32 node with disk in the first layer, 32 gigabit Ethernet links to switches in the layer 3 and 2 10 gigabit Ethernet links to servers; 64 switches with 64 gigabit Ethernet links to the switches in the layer 2 and 64 gigabit Ethernet links to switches in the layer 4; and 32 switches with 128 gigabit Ethernet links to the switches in the layer 3.

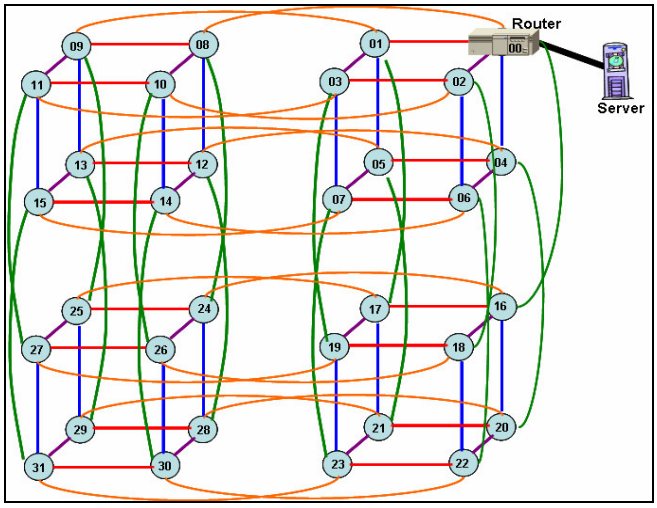

Figure 3 - Hypercube Network

Hypercube 12-dimension: 3968 nodes with disks on switches with 12 gigabit Ethernet links, and 128 routers with 12 gigabit Ethernet links to other nodes and 2 10 gigabit Ethernet links to servers. Routers are positioned randomly in the topology to minimize the impact of routers too close in the topology as demonstrated in [10].

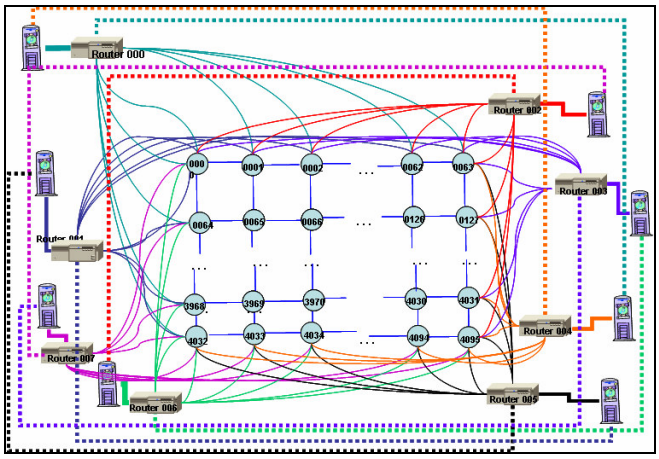

Figure 4 - Mesh 2D network

Mesh 2-dimensions: 4096 nodes with disks with up to 4 gigabit Ethernet links connected to other nodes and the border switches with 4 gigabit Ethernet links to routers, and 8 routers with 128 gigabit Ethernet links to nodes with disks and 2 10 gigabit Ethernet links to servers.

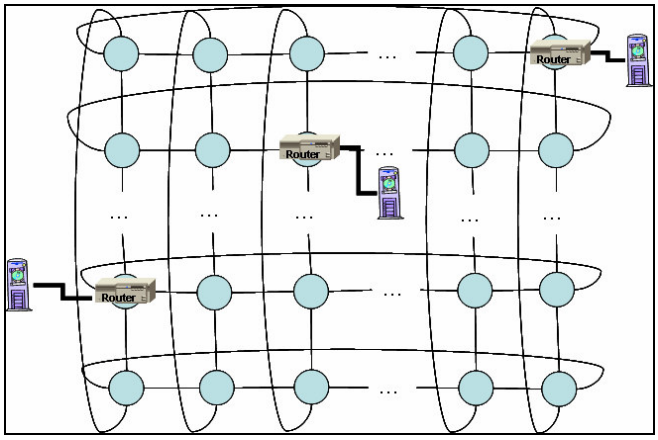

Figure 5 - Torus 2D network

Tori in 4 different dimensions: Each one with 3968 nodes with disk and switches and 128 routers all with the same level of connectivity depending on the dimension. The 128 routers always have 2 additional 10 gigabit Ethernet links to servers and are placed randomly in the topology:

- **2 Dimension**: 64x64 with each switch and router having 4 gigabit Ethernet links
- **3 Dimension**: 16x16x16 with each switch and router having 6 gigabit Ethernet links
- **4 Dimension:** 8x8x8x8 with each switch and router having 8 gigabit Ethernet links
- **5 Dimension:** 4x8x4x8x4 with each switch and router having 10 gigabit Ethernet links

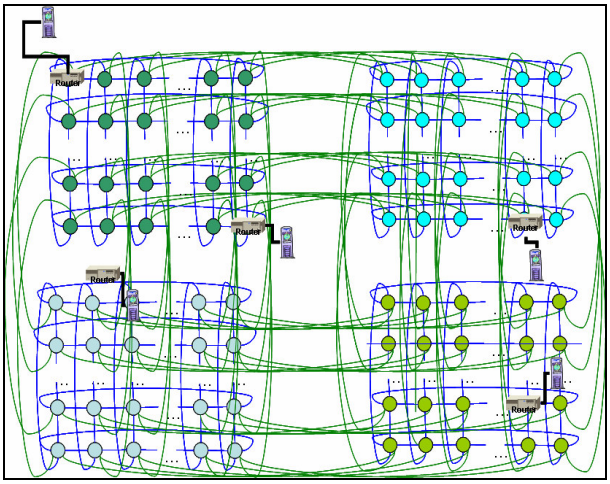

Figure 6 -Torus 3D network

4.4 Simulation assumptions

In order to model systems with these level of complexity in a manner that allows them to be programmed in a reasonable amount of time and run on limited time and computing power, we had to make several assumptions both to simplify the system and to limit the number of variables that would need to be simulated and randomized. This also allowed for a clearer and sharper vision of the results. Some of the assumptions are:

The behavior of the links follows the rules of Ethernet in terms of inter-packet gap and minimum size. All links are considered full duplex which is the standard for both gigabit Ethernet over fiber and gigabit Ethernet over twisted pair [IEEE 802.3]. This will create in effect two separate logical links for each physical link. Statistics were compiled separately and combined at the end of the simulation to calculate averages. For link failure considerations we assume that the physical link has failed and declare both logical links as failed.

The behavior of the switches will not simulate the 802.1D standard of bridging [IEEE 802.1D] but instead will follow a more liberal approach of a routing algorithm similar to Infiniband [7,15] where messages are redirected only to the next node that offers the shortest path to the destination. The shortest path is determined locally based solely on the topology being used, the destination address, and sometimes on the source address. There is no routing protocol, route discovery or routing information exchange between any of the nodes. When there are multiple shortest paths available from a single node to a particular destination, the path is chosen based on the following tie breakers in this order: which link will be available first for transmission, which link has been used the less and unbiased round robin order.

When the routing algorithm encounters a failure in the path chosen most of the time it will be taken care by the algorithm mentioned above. Only when there is no more than one link to the destination do we need a different approach to forward the message. In this case the algorithm will look at alternative routes that are not the shortest one and will use the same algorithm to decide which one of those routes to choose.

A node can not forward a message to the node from which it has received it. This is to avoid a loop when a message will be trapped back and forth between two nodes. This algorithm does not prevent a message from being caught in a loop with more than two nodes but that is statistically very unlikely and we did not observe this behavior in our simulations.

The size of the packets is restricted to a few options and not all of them match the available standards for Ethernet [IEEE 802.3]. In bytes those sizes are 64, 1024, 1500, 2048, 4096, 8192, 16384, 32768 and 65536. Any arbitrary sized request data can be subdivided in packets of these sizes and, although it does not seem as efficient as when using specific sizes, for sufficiently large messages usually utilized when reading or writing data to and from disks, the difference in performance is

negligible. But the simplification of the simulation's performance and programming are significant.

We calculated the amount of time taken to transmit each one of these different size packets on a regular gigabit Ethernet using the standard calculations for Gigabit Ethernet. The minimum packet size is 64 bytes. We add the size of the preamble (8 bytes) and the equivalent size of the minimum inter-gap time (12 bytes) and that gives us a total size of 84 bytes. The nominal throughput for gigabit Ethernet is 10,000,000,000 bits per second. Doing the math we reach 1,488,095 packets per second. That yields a delay of 67ns for a 64 bytes packet.

Our simulation time unit is fixed as 1 µs (1 micro second). The amount of time required to send a 64 bytes packet on gigabit Ethernet was approximated to 1 µs. The amount of time for a 64K bytes packet, if sent continuously as a single packet, would be 525 µs. If sent in segments with size 8K bytes, allowed under commonly accepted Jumbo Frame standards, the transmission would take 527 µs.

We introduced a further simplification of the model for the simulation of large networks; we only used packets of either 64 bytes or 64K bytes. The first were used to transmit read requests and write replies, and the second were used to transmit blocks of data for read replies and for write requests. On a comparison run, using 64K bytes and 8K bytes packet sizes, the difference in terms simulated was non perceptible but the amount of time taken by the simulation using 8K bytes was 10 times longer than when using 64K bytes.

Lastly the size of a read or write requests from a server to a single disk were fixed at 512 Kbytes. This is a reasonable assumption if we consider that the server will break data requests in smaller chunks to send them in parallel to different disks.

The disk access delay was consider to be 22ms which is derived from the seek time and data transfer of a 15000 rpm SCSI-II disk or of a SAS-1 disk. As a base, we used the Seagate Cheetah 15Krpm SCSI-II [25]. It has a seek time of 3.5ms for reads and 4.0ms for writes and a sustained data transfer of 73MB/s. If we assume one seek for every 128KB of data, and use the worst case of 4ms we arrive at a disk delay of 21.41ms.

On read requests, the disk will wait for 22ms before starting the transmission of the 512KB. The read data will be divided on packets fitting the maximum packet size (64KB for this simulation) and sent one after the other. In the case of write requests, the disk will buffer all the segments of the write request and will send a confirmation to the router that sent the request as soon as it has received all segments. Then it will process the write and keep the disk busy for 22ms.

If the disk is busy when a request arrives, the request will be queued. Each disk has two queues to hold requests: one for reads and one for writes. The disk will empty the read queue first before checking the write queue since the write has already been acknowledged and the read still has to be sent. This could in theory prevent writes from being committed to disk, but the random nature of the request generation prevented it from happening during the simulation.

The switches were considered to be crossbar switches where traffic from one port to another does not cause interference with other traffic in the switch. Since the times for transmissions on the links are measured from when the first byte is sent on the transmitting end until the last byte arrives on the receiving end, including the preamble, postamble and minimum inter-gap packet, the switch only has to determine the outgoing port and queue the packet in the link. Most crossbar switches have latency between 500ns and 20ms. For our simulation we considered a store and forward switch with a low latency of 6 ms.

Packets are sent immediately by the switch into the link after the switch delay unless the link is already busy. There is a separate queue per port on each switch and for our simulation we consider the queue size to be infinite and the time spent on enqueueing and dequeueing packets to be negligible.

The simulator accepts 3 parameters to determine the load of the system:

- Number of simultaneous requests per router. This parameter indicates how many outstanding requests each router can have at a certain time. When a request is fulfilled a new request is generated. When a request's timeout expires a new request of the same type (read or write) is generated to a different disk. Late replies for expired requests are discarded.
- Randomization interval for requests: This parameter is used to determine the interval when all routers will generate their first round of requests. At initialization time each router will generate the maximum number of simultaneous requests. Each request will be randomly selected as a read or a write request and will be scheduled randomly between 0 and the randomization interval. When a request is fulfilled and a new request is generated to a random disk, it will also be randomly sent within the same random time interval starting from the current time. This mechanism generates a self throttling loaded system. When the delay to complete a request is long, there are fewer requests generated per time unit in the system. When the delay to complete a request is short, there are more requests generated per time in the system because it is capable of completing them.

- Request timeout. This parameter is the amount of time that a router will wait to receive a reply from a disk before declaring the request lost and generating a new request to another disk.

There is a startup time fixed in the simulation to be double the time of a regular disk access. When the simulation time reaches the defined startup time, all statistics are cleared. This procedure is used to avoid distortion in the measurements during the "warm up" of the system when it is empty, response time is much faster and collisions are rare.

The simulator also needs the amount of time being simulated. This parameter has to be larger than the request timeout and the startup time described above.

We simulated three types of failures with a different number of components failing simultaneously. All failures were generated randomly but could easily also have been generated deterministically.

- **Disk failures.** We only simulate failure of 4 disks simultaneously. When disks fail, all requests sent to them are going to be lost since there is no disk to respond. The switch to where the disk is connected is still working so will not affect the network performance and the network will not be able to detect this failure.
- **Link failure**: We simulate failure of 8 links, 16 links, and 24 links. There is the possibility of network segmentation in some cases but that was not observed. A failed link will return -1 as the time when it will be free and also as the network utilization effectively removing it from the routing algorithm used in the nodes.
- **Switch failure:** We fail one switch and two switches. A switch failure assumes that all the links in the switch will fail. In turn all the links connecting to this switch will fail (this is necessary because we consider all the links to be full duplex). We did not fail switches with only one link since that would be equivalent to failing the disk attached to that switch. But in the case where the switch with the disk has more than one connection, the switch can be failed and the disk also can not be accessed. This will cause lost messages for the hyper cube, Mesh and Tori topologies.

5 Results and analysis

5.1 Determination of the simulation load

We run the simulation using different loads. All the loads used the same interval randomization value equal to the disk delay access. This was done in order to bring the system faster to a steady state. Since different topologies have different number of routers, we parameterized on a total number of simultaneous outstanding requests for the whole system and divided that number by the number of routers to achieve the requests per routers.

We simulated 6 different loads expressed by total number of requests: 1280, 2560, 3840, 5120, 6400 and 7680. We then divided the number of requests by the number of routers and arrived to the following loads: for topologies with 128 routers, it was 10, 20, 30, 40, 50 and 60 respectively. For topologies with 256 routers it was 5, 10, 15, 20, 25 and 30 and for topologies with 8 routers it was 160, 320, 480, 640, 800 and 960.

From the results we can see that above 6400 requests the disks are almost over saturated in the topologies with more links such as Hypercube, Fat Tree, Torus 4D and 5D (Figure 7). On the topologies with less links such as Butterfly, Mesh and Torus 2D, we see that most links are saturated without being able to keep the disks busy (Figure 8).

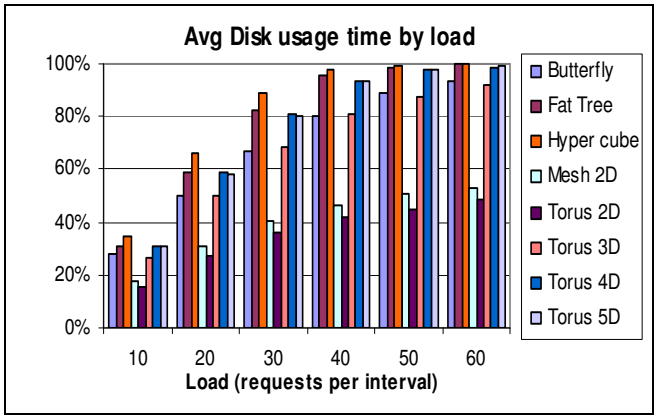

Figure 7 - Average disk usage time by number of requests per interval

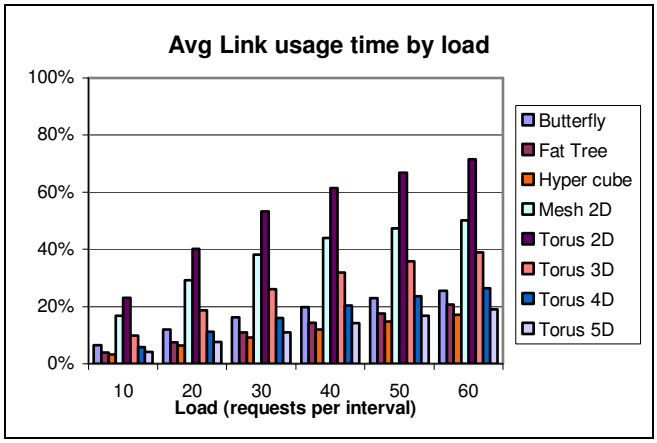

Figure 8 - Average link usage time by number of requests per interval

On Figures 9, 10 and 11 we can see the network throughput, request delay and link collision percentages for each run. The network throughput is the aggregated sum of all the bytes received by routers and disks. The

request delay is the average time taken by a request since it is sent from the router until it finishes receiving the replies from the disk. The link collision is the percentage of all packets that when arriving at a link, found that the link was busy transmitting another message and had to wait in queue. We can see that in the topologies where the links have over 50% utilization, the number of collisions also has climbed to over 50% and the I/O delay has increased significantly at the same time that the network throughput has decreased.

Based on these results we decided to conduct our simulations at a steady 5120 requests per interval which would allow most of the topologies to top 100GB/s of aggregated throughput and still have some room to increase the disk utilization and keep adequate I/O requests delay.

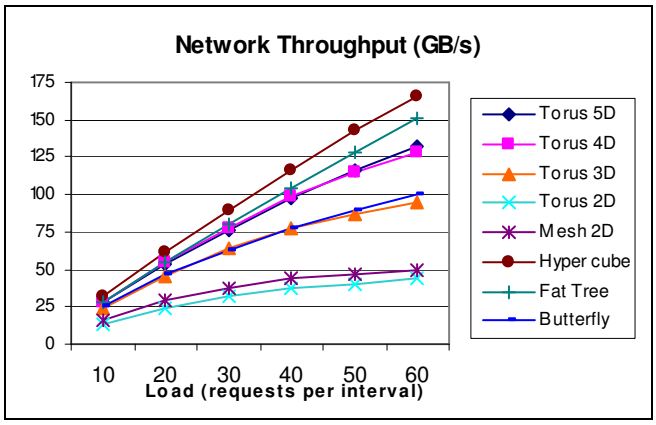

Figure 9 - Network throughput by system load

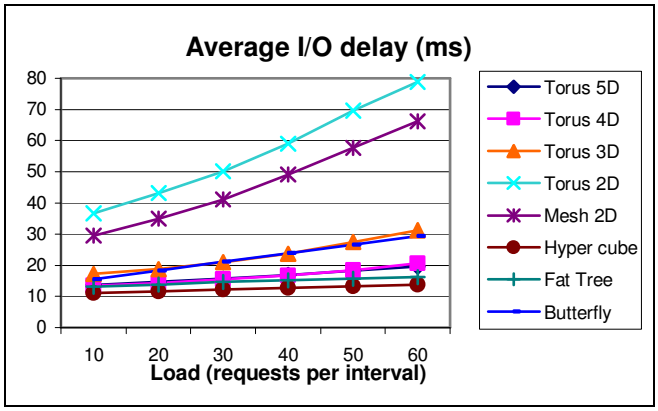

Figure 10 - Average I/O delay by system load

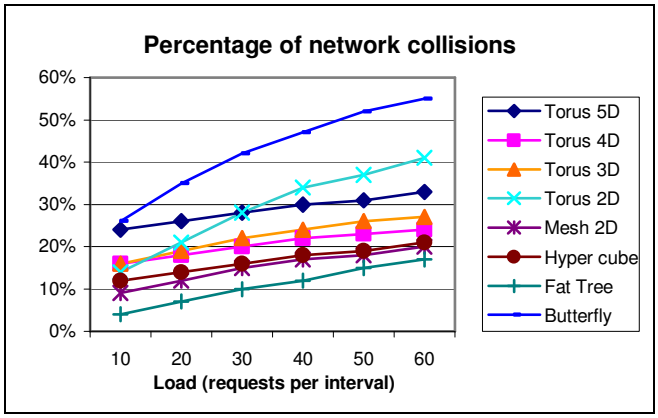

Figure 11 - Percentage of messages in links that were queued because of collisions

5.2 Effects of failures on number of hops per message

The number of hops that a message has to traverse to arrive at its destination is a measure of the connectivity of the topology and also an indication of the delay that it will face. We counted the hops for all messages and divided them by the total of messages in the links. When failures occur some messages will need to deviate and take longer paths in order to avoid failed devices. This is not necessary for disks. Figure 12 the average hops count for each topology in the base case and Figure 13 shows the maximum variation due to failures.

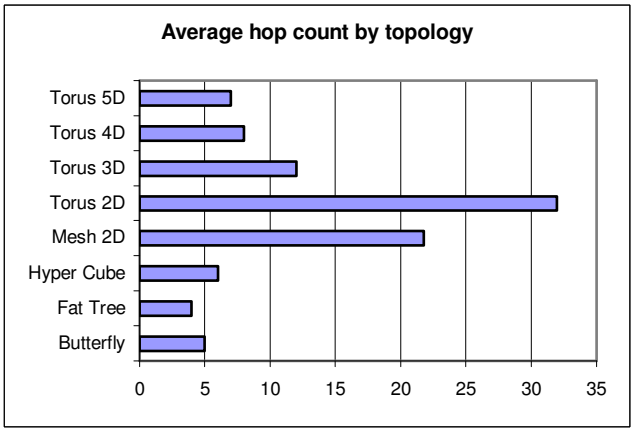

Figure 12 - Average hop count by topology

We can see that for topologies with many links per node there is almost no variation on the number of hops. The less links per nodes, the greater the variation on the number of hops, but even in the cases like Torus 2D and Mesh 2D the actual percentage variation over the absolute total is very small (0.38% for Torus 2D and 0.39% for Mesh 2D). In general hop count on all topologies behave well under failures.

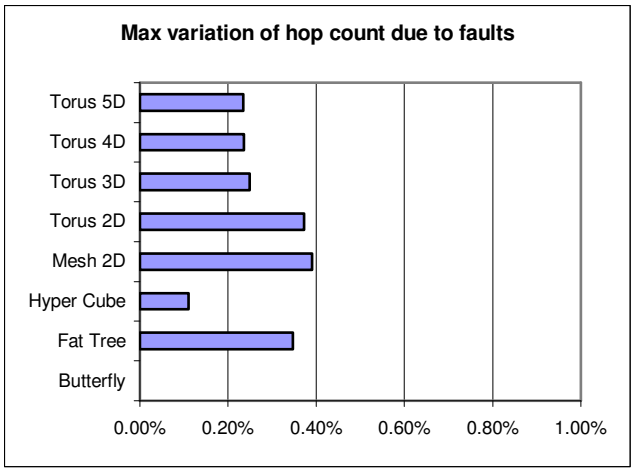

Figure 13 - Maximum variation of hop count due to failures

5.3 Effect of failures on I/O requests delay

The delay for each request will directly affect the overall performance of the system and it is an important parameter when designing a network and evaluating its final throughput. In our case we examine both the average I/O request delay in ms as shown on Figure 14 and 15 and the distribution over time for number of requests per millisecond as shown in Figure 16. In the first metric we are looking for a trend that would indicate that the network is slowing down in average. In the second metric we are looking for indications of hot spots that could be masked in the average. Additionally, this information should be correlated with the number of lost messages as seen in the next item. The loss of a message indicates that the I/O request it was carrying will timeout and will need to be retransmitted therefore increasing the delay.

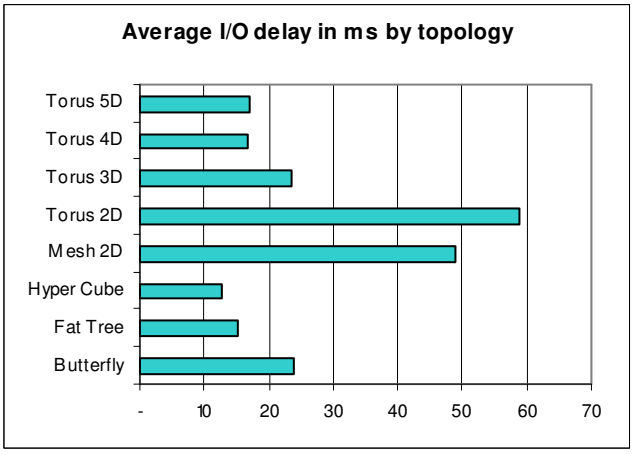

Figure 14 - Average I/O delay by topology

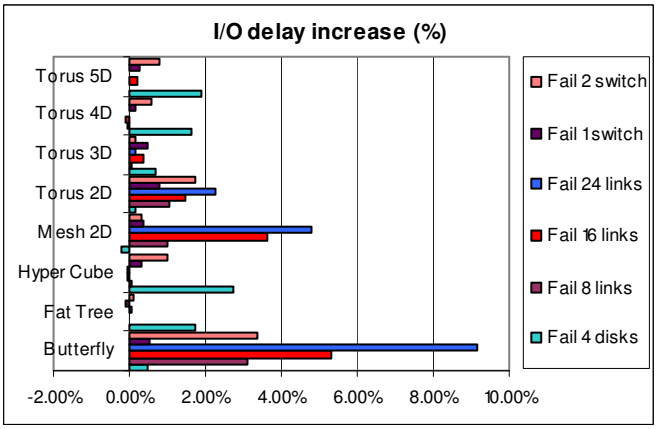

Figure 15 - I/O delay increase by failure and topology

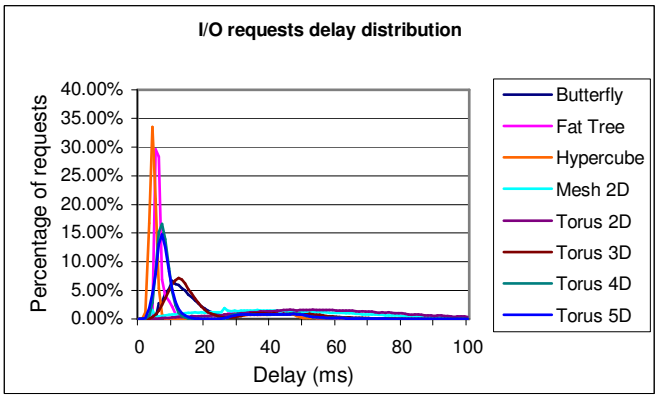

Figure 16 - I/O requests delay (in ms) distribution for base case for each topology

The I/O request delay increases in all cases when we fail disks because requests directed to those disks will timeout and will need to be retransmitted, greatly increasing the I/O delay. Fat Tree is insensitive to other failures, especially since we are not failing links or switches connected to disks and there are many redundant connections. Hypercube, Mesh 2D and Tori are sensitive to switch failures since each switch is connected to a disk but the sensitivity to link failures varies greatly. Torus 2D and Mesh 2D are the most sensitive since there are very few links per node. Torus 3D, 4D and 5D are similarly insensitive to link failures, while Hypercube is not affected. Butterfly is the most affected due to its lack of redundant paths.

Looking at the distribution of delay per messages, we can see that Hypercube and Fat Tree have almost all messages delay in a very narrow interval between 3ms and 5ms, while the Tori and the Mesh get more dispersed as the number of links per node decreases. Only the Torus 2D, Mesh 2D and Butterfly show different distribution of delays under failures, while the other 5 topologies are virtually indistinguishable, reinforcing the conclusion that delay is not affected by failures.

5.4 Effect of failures on messages lost

Loss of messages indicate that the level of congestion has increased enough to cause the requestor to timeout the request or to partition the network in a way that there is no route to reach the destination, or the routing algorithm is incapable of finding a valid route. We experimented with different timeout values to determine the lowest timeout that would separate network congestion from lack of viable routes. We run the simulation on a Torus 2D and vary the timeout from 175ms to 275ms. The results are shown on Figure 17. We then measured the influence of messages lost on the number of requests completed without timeouts and no retries, as shown on Figure 18.

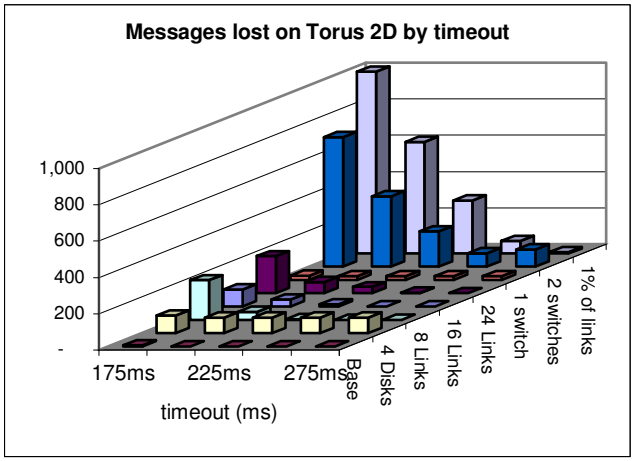

Figure 17 - Number of messages lost in a Torus 2D with different timeout values under failures

The first test showed that Torus 2D would not miss any messages if the timeout was long enough, showing that the network was not partitioned. The second test showed that other than disk failures, which would certainly cause messages to be lost, all topologies, with the exception of Butterfly, were able to receive all messages sent.

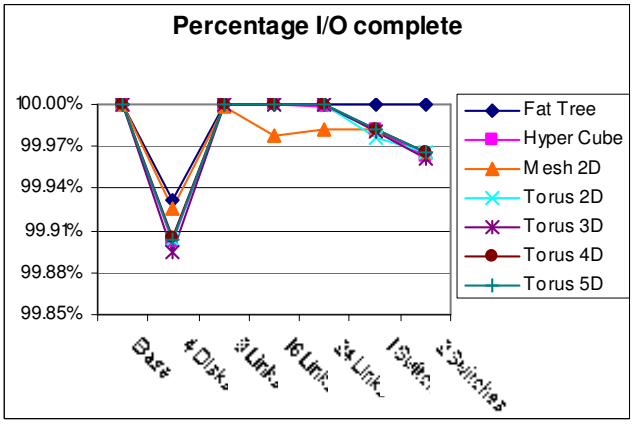

Figure 18 - Percentage of I/O requests completed without timeout

5.5 Effect of failures on network congestion and network throughput

Network congestion and network throughput are inter-related and are different measurements of performance of the overall system. Network congestion measures the percentage of times a packet arrived at a link in the network and the link was busy transmitting another message and the packet had to be queued. Since links are full duplex, a link was receiving behavior had no effect on the transmission, so congestion is measured only on the transmission side. We discounted the effects of limited buffers for queuing and assumed that there were separate buffers for transmission and reception. When links or switches fail in the network, adjacent links get additional traffic and that increases the amount of collisions that will be seen in the network overall. Average collision percentage per topology on the base case are shown on Figure 19, percent variation on the amount of collision for each type of failure are shown on Figure 20. Notice that in some cases the variation is negative, indicating that there are less collisions. Distributions of percentage of link usage time are shown on Figure 21 and 22. We show the total accumulated percentage of links usage in percentage of time for each type of network and each failure. In most cases the variation by failures is so small as to be considered null.

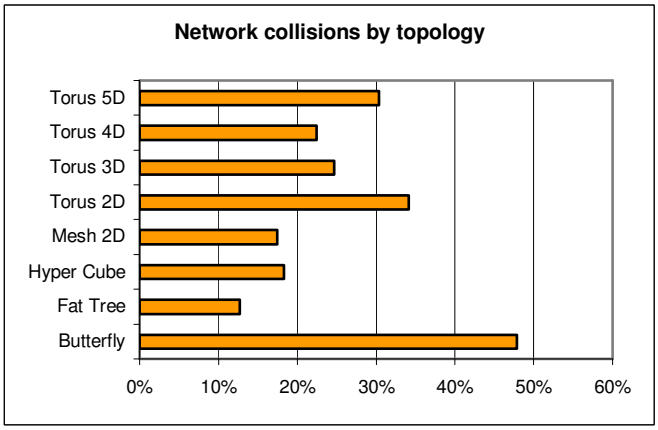

Figure 19 - Percentage of network collisions on links per topology on base case

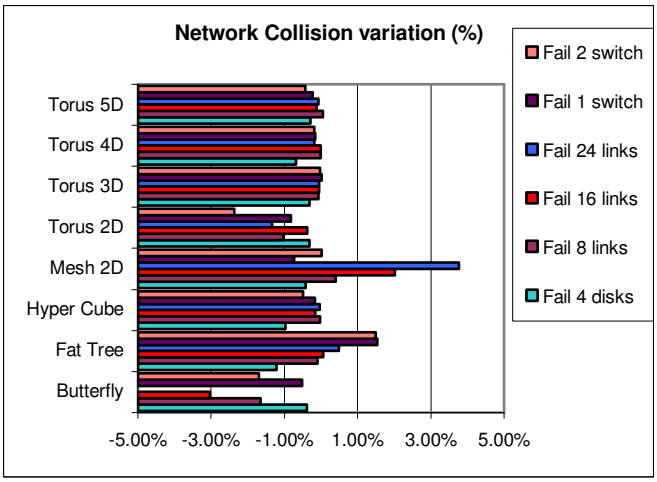

Figure 20 - Percentage variation from base on network collisions by topology and failure

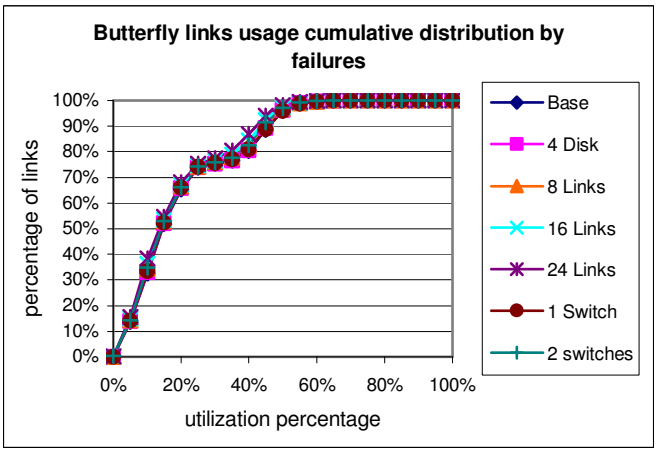

Figure 21 - Link usage cumulative distribution for Butterfly topology

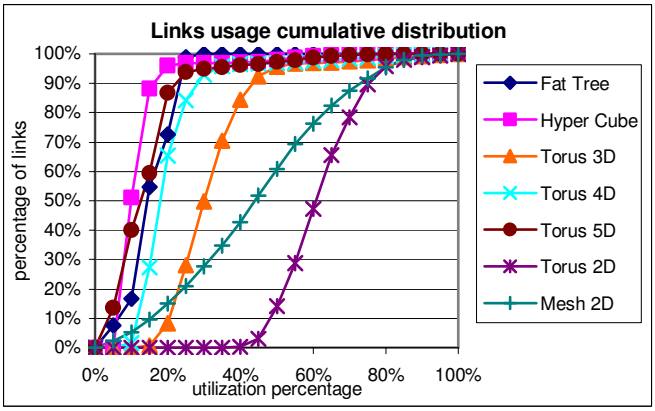

Figure 22 - Link usage cumulative distribution for base case for all topologies

Similarly, the aggregate system's overall network throughput indicates the capability of the system to transfer data in and out of the disks. The effect of failures on data throughput is due both to the increased congestion of the network and the need to re-request the data when an undetected disk failure occurs. Also, disk utilization changes slightly when disks fails due to the increased number of requests per disk. That can be offset by fewer requests being generated due to network congestion. Figure 23 shows the aggregate network throughput and Figure 24 shows the percentage drop in network throughput for each different type of failures. Similarly Figure 25 shows the disk utilization time average per topology in the base case and Figure 26 shows the disk utilization drop per topology for each failure case. Again note that in some cases disk utilization increases under certain failures.

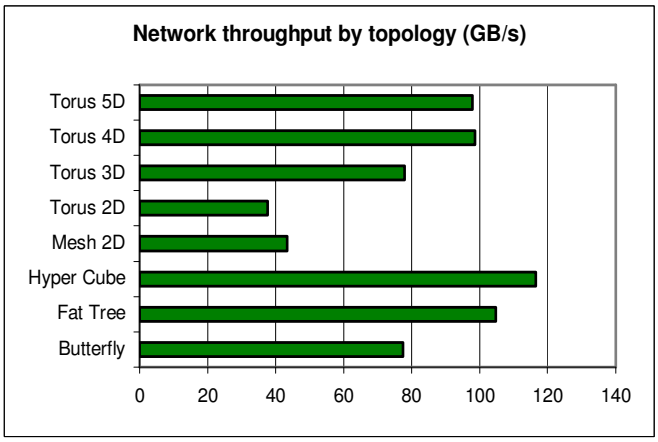

Figure 23 - Aggregate network throughput per topology on base case

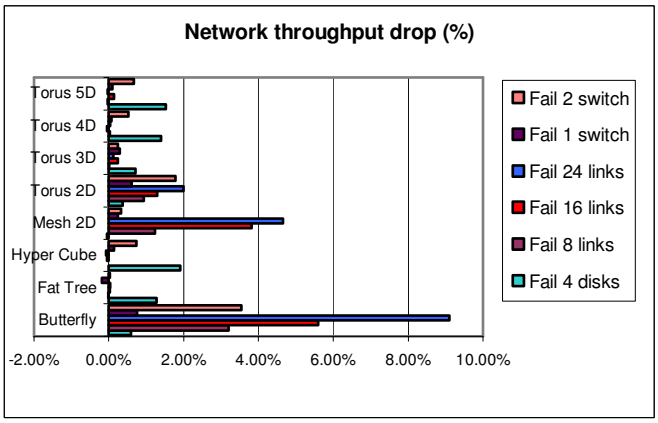

Figure 24 - Network throughput drop by topology for different failures

We can see that the number of network collisions increases only for Mesh 2D and Torus 2D and it actually decreases for Butterfly networks. The latter is due to the fact that the network becomes partitioned and there are many links that will not receive any traffic, much less collision and that would bring the average down.

The network throughput is extremely high on Hypercube and Fat Tree due to the large number of paths between routers and disks that keep the network congestion low. The fact that our results for Fat Tree show lower network throughput than the Hypercube

when the first has more redundant network connections is explained by the fact that our Fat Tree design uses only one network connection from the disk to the network compared with 12 connections for a disk in our design for the Hypercube. All segments of a data message are sent simultaneously in different packets. In a Fat Tree topology, all these packets will be queued on the single link to the disks while in the Hypercube they can be distributed among the many links of the switch. Torus 5D and Torus 4D see a similar performance slightly slower than the first two since they have fewer links on the network and fewer connections per switch.

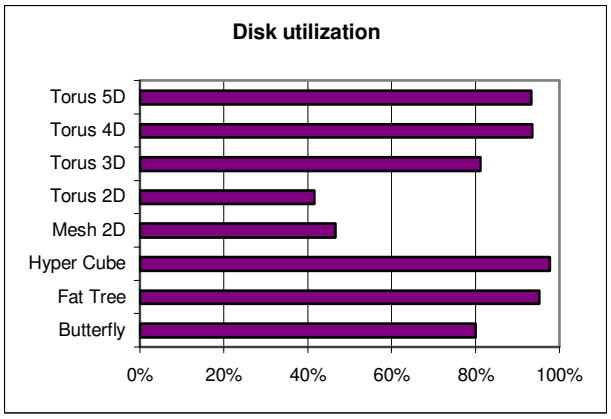

Figure 25 - Disk utilization per topology on base case. It includes rotation time, seek and access time.

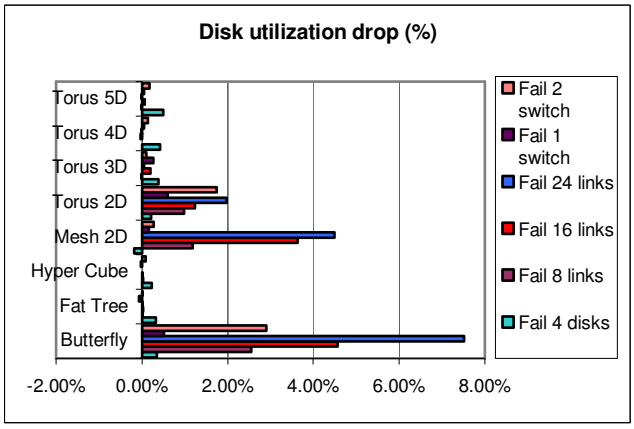

Figure 26 - Disk utilization drop per topology under different failures

Torus 3D and Butterfly have about the same middle of the ground network performance, but Butterfly is affected more by failures due to the network partition. On the slow side are Torus 2D and Mesh 2D with not enough links to maintain the throughput and more susceptible to failures.

Disk utilization does actually go down when there are fewer disks. This can be explained by the fact that requests to failed disks are still being generated but are not being processed. In turn this leads to fewer requests in the network leading to less disk utilization. This can also be observed when there are fewer switches or fewer links or anything that would lower the network throughput. For these topologies the disks are not a performance constrain, but actually reflect the capacity of the network in generate requests.

In general, higher throughput networks are more susceptible to disks failures due to the fact that requests can be delivered at the same rate to disks when there are network failures than on lower throughput networks. This can be observed when comparing the effect of 2 disks failures on Hypercube (significant) and Mesh 2D (not noticeable).

6 Future work

Some considerations for future optimizations are the use of more efficient routing algorithms while still maintaining the simplicity of local views with no routing protocols and the use of mixed topologies to try to take advantage of the best qualities of each topology to create a simpler, faster and more fault tolerant overall system.

In the topic of failures, the next step would be the implementation of failure detection algorithms and protocols. The effect of failures in the system can be traced to two different reasons: One is the fact that when devices fail there are fewer of them available to serve the same load: fewer routers to route requests, fewer switches to provide additional bandwidth, fewer links to provide alternative routes and fewer congested paths and fewer disks to serve the same amount of data. The other component is the fact that the devices in the system are not aware that one or more devices have failed and keep sending requests to that disk or using that switch or counting on that link to forward the message. This creates unnecessary errors, longer paths and eventual timeouts with its consequent retries. In order to understand the tradeoffs between implementing yet another protocol and processing additional messages and using additional bandwidth and the savings that could come from the fact that the system would not try to use devices that are not longer responding we need answers to several questions: how would its use affect the network performance? And the opposite too, how would the network topology and its inherent performance limitations affect the accuracy, responsiveness and effectiveness of the failure detection protocols? Which failure detection is more efficient and accurate for each different topology or to detect different failures? This topic has been explored by Tan in [16] but many questions remain unanswered.

This whole study is based on a theoretical premise that we can use Ethernet like connections and switches which provide understandable technology readily available and at a low price that a mass market creates. But we expect all the switches to behave more like routers than real switches and that is still a very difficult

obstacle to overcome. Thinking about alternatives that could be implemented is another area of interest to study.

Furthermore, the simulation can be improved to include the next layer in the storage access which would be the transport protocol and the data access protocol. Interactions at a higher level can have unintended consequences on the system performance when failures occur as demonstrated with the Panasas system [9] and routing in Slice [1].

7 Conclusions

When choosing the design of a high performance petabytes scale storage network we must consider not only the total data throughput, but also the sensitivity to different types of network failures. To guide such decision we evaluated the performance and tolerance to failures of 8 different network topologies - Butterfly, Fat Trees, Hypercube 12D, Mesh 2D, Torus 2D, 3D, 4D and 5D - using a simulation model.

We found that Hypercube and Fat Tree have the best performance both on the base case and also under any of the failures. The performance of the Fat Tree was hindered by our choice of design with a single connection from disks to network but still had an impressive performance with only half the gigabit Ethernet ports than the Hypercube and offering 128 more disks. What makes Fat Tree more difficult to build is the intermediate switches with 128 ports and the routers with 64 ports,. This will likely not be a restriction in the near future since new switches at more competitive prices are coming out in the market. In favor of Fat Trees is the fact that cabling and managing it is more straightforward and it is easier to visualize than Hypercube which uses switches with 12 connections to very different switches in the network and with no easy to discern pattern.

Torus 5D and 4D still had a good performance and a relatively good tolerance to failures and used fewer connections than Hypercube but more than Fat Tree. Again, the maintenance of the connection pattern needed for Torus is not trivial. Butterfly and Mesh 2D also have good performance under some parameters but tolerance to failures is not the best, especially on Butterfly. Mesh also has the added challenge of using just 8 routers with 128 gigabit Ethernet ports and several 10 Gigabit Ethernet connections for servers. These are still more difficult to build currently than the 64 ports plus 2 10GbE for the Fat Tree topology. Torus 2D and 3D have the worse performance to complexity ratio, with still too many links for not enough performance. Fault tolerance was still acceptable but these topologies are more sensitive to link failures.

8 References

[1] Darrell C. Anderson, Jeffrey S. Chase, and Amin M. Vahdat. Interposed request routing for scalable network storage. *ACM Transactions on Computer Systems (TOCS), Volume 20 Issue 1,* Publisher: ACM Press February 2002

[2] Miguel Castro and Barbara Liskov. Proactive Recovery in a {B}yzantine-Fault-Tolerant System. *Technical report, Massachusetts Institute of Technology*, 1999.

[3] M. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based Failure and Evolution Management. *In 1st NSDI*, 2004.

[4] Garth A. Gibson, Rodney Van Meter. Network attached storage architecture. *In Communications of the ACM, Volume 43 Issue 11*, November 2000

[5] R. Honicky and E. Miller. An Optimal Algorithm for online reorganization of replicated data. *Proceedings of the 17th International Parallel & Distributed Processing Symposium,* Nice, France, April 2003.

[6] A. Hospodor and E. L. Miller. Interconnection Architectures for Petabyte-scale High-performance Storage Systems. *In Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2003.

[7] InfiniBand Clustering: Delivering Better Price/Performance than Ethernet. *Mellanox Technologies White Paper.*

[8] Daniel A. Menascé, Odysseas I. Pentakalos, Yelena Yesha. An analytic model of hierarchical mass storage systems with network-attached storage devices. *ACM SIGMETRICS Performance Evaluation Review, Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems SIGMETRICS '96*, Volume 24 Issue 1, May 1996

[9] David Nagle, Denis Serenyi, Abbie Matthews. The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage. *Proceedings of the 2004 ACM/IEEE conference on Supercomputing SC '04,* November 2004

[10] Christopher Olson, Ethan L. Miller. Cryptographic storage security: Secure capabilities for a petabyte-scale object-based distributed file system . *Proceedings of the 2005 ACM workshop on Storage security and*

survivability StorageSS '05, November 2005. Publisher: ACM Press

[11] Renato John Recio. Promises and reality: Server I/O networks past, present, and future. *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications* NICELI '03, August 2003. Publisher: ACM Press

[12] P. Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, July 2003.

[13] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kosti´c, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. *In 5th OSDI*, 2002.

[14] A.S. Vaidya, C.R. Das, A. Sivasubramaniam. A testbed for evaluation of fault-tolerant routing in multiprocessor interconnection networks. *Parallel and Distributed Systems, IEEE Transactions on Volume 10, Issue 10*, Oct. 1999.

[15] A. Vishnu, A.R. Mamidala, H. Jin, D. Panda. Performance Modeling of Subnet Management on Fat Tree InfiniBand Networks using OpenSM. *In Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International,* April 2005

[16] C.Tan. Failure Diagnosis for Configuration Problem in Storage System. IBM Almaden Reseach Center

[17] Hong Tang, Aziz Gulbeden, Jingyu Zhou, William Strathearn, Tao Yang, Lingkun Chu. A Self-Organizing Storage Cluster for Parallel Data-Intensive Applications. *Proceedings of the 2004 ACM/IEEE conference on Supercomputing, Publisher: IEEE Computer Society* November 2004

[18] Q. Xin, E. L. Miller, T. J. Schwarz, D. D. E. Long, S. A. Brandt, and W. Litwin. Reliability mechanisms for very large storage systems. *In Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, Apr. 2003.

[19] Q. Xin, E. L. Miller, T. J. E. Schwarz. Evaluation of Distributed Recovery in Large-Scale Storage Systems. *In Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, June 2004.

[20] Q. Xin, E. L. Miller, T. J. E. Scharz, D. D. E. Long. Impact of Failure on Interconnection Networks for Large Storage Systems. *In Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005),* April 2005

[21] EMC Corporation, http://www.emc.com/index.jsp

[22]EqualLogic Incorporated, http://equallogic.com/

[23] IBM Systems Storage, http://www-03.ibm.com/systems/storage/index.html

[24] Network Appliance Incorporated, http://www.netapp.com/

[25] Seagate Technology LLC, http://www.seagate.com/products/enterprise/cheetah.html

[802.3] IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements. Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. *IEEE Std 802.3-2005,* December 9, 2005.

[802.1D] IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common requirements. Part 3: Media Access Control (MAC) Bridges. *ANSI/IEEE Std 802.1D 1998 Edition.* 1998.