Performability Evaluation of Networked Storage Systems Using N-SPEK

Ming Zhang and Qing Yang
Electrical and Computer Engineering
University of Rhode Island
Kingston, RI 02881 USA
{mingz, qyang}@ele.uri.edui

Xubin He
Electrical and Computer Engineering
Tennessee Technological University
Cookeville, TN 38505, USA
hexb@tntech.edu

Abstract

This paper introduces a new benchmark tool for evaluating performance and availability (performability) of networked storage systems, specifically storage area network (SAN) that is intended for providing block-level data storage with high performance and availability. The new benchmark tool, named N-SPEK (Networked-Storage Performability Evaluation Kernel module), consists of a controller, several workers, one or more probers, and several fault injection modules. N-SPEK is highly accurate and efficient since it runs at kernel level and eliminates skews and overheads caused by file systems. It allows a SAN architect to generate configurable storage workloads to the SAN under test and to inject different faults into various SAN components such as network devices, storage devices, and controllers. Available performances under different workloads and failure conditions are dynamically collected and recorded in the N-SPEK over a spectrum of time. To demonstrate its functionality, we apply N-SPEK to evaluate the performability of a specific iSCSI-based SAN under Linux environment. Our experiments show that N-SPEK not only efficiently generates quantitative performability results but also reveals a few optimization opportunities for future iSCSI implementations.

***Key words**: Performability, benchmarking, networked storage, block level access*

1. Introduction

One of the primary concerns of storage area network (SAN) is high data availability besides performance. To ensure high availability, a typical SAN has built-in redundancies at various levels including storage devices such as RAID, controllers such as HBA and NIC, and network components such as switches, bridges, and connecting cables. Software mechanisms are employed to bypass failed components to provide continued data availability. Different topological architectures and fault-tolerant mechanisms exist for SAN and new ideas and technologies emerge rapidly [20]. It is highly desirable to have efficient benchmark tools to quantitatively evaluate performance and availability of various SAN architectures. While there are benchmark tools for file system and disk performance evaluations such as PostMark [9], IoMeter [8] and others [3, 4, 14], there is little work done on benchmark tools for evaluating availability of a networked storage system. One exception is the research work done by Brown and Patterson [2] who were the first to advocate for availability benchmarking with a case study on evaluating the availability of software RAID systems.

In this paper, we present a kernel level benchmark tool for evaluating the availability of SAN systems named N-SPEK (Networked-Storage Performability Evaluation Kernel module). N-SPEK measures performance levels under various fault conditions in terms of performability (performance + availability) over time instead of using an average percentage of "up" time as an availability metric. N-SPEK allows a user to generate workloads to a SAN, to inject faults at different parts of the SAN such as networks, storage devices, and storage controllers. By generating configurable workloads and injecting configurable faults, users can grab the dynamic changes of potentially compromised performance and therefore quantitatively evaluate system performability of a measured SAN. To demonstrate how N-SPEK works, we carried out performability tests on an iSCSI SAN system.

The paper is organized as follows. Next section presents the structure and behaviors of N-SPEK. A case study on iSCSI will be presented in Section 3. We conclude the paper in Section 4.

2. Structure of N-SPEK

Figure 1 shows the overall structure of N-SPEK. N-SPEK contains a controller, several workers, one or several probers, and different types of fault injection modules.

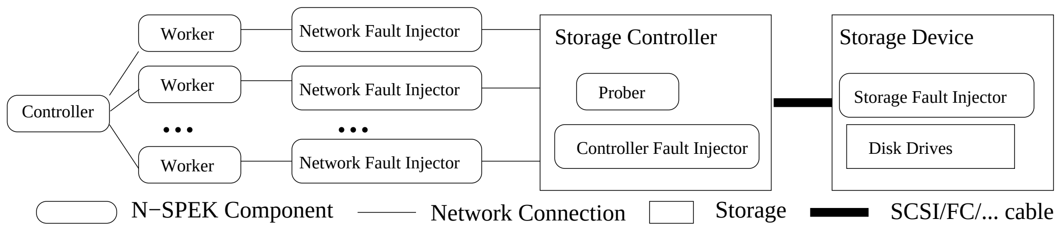


Figure 1. N-SPEK Structure

An N-SPEK Controller resides on a controller machine which is used to coordinate N-SPEK Workers and Probers. It can start/stop N-SPEK Workers and Probers, send commands and receive responses from them. One N-SPEK controller controls several workers and each worker generates I/O requests to storage targets independently. A Java GUI interface of the controller allows a user to input configuration parameters such as workload characteristics and to view measured results. The controller also has a data analysis module to analyze measured data.

There is one N-SPEK Worker running on each testing client to generate I/O requests via the low level device driver and to record performance data. As a Linux kernel module, each N-SPEK Worker has one main thread, one working thread, and one probe thread. The main thread receives instructions from the N-SPEK Controller, controls the working thread to execute actual I/O operations, and reports results to the controller. The working thread keeps sending requests to SCSI layer that are eventually sent to remote targets by a lower level device driver. The probe thread records system status data periodically and reports to the N-SPEK Controller once test completes.

	Ext2	Ext3	Ext2 with Sync flag
PostMark	2.375	1.351	0.511
IoZone	65.228	61.676	2.136
Bonnie++	61.495	56.773	3.236

Table 1. Measured throughput (MB/s) of PostMark, IoZone, and Bonnie++ with changes of file system options

By running in the kernel space, N-SPEK minimizes overheads caused by system calls and context switches compared to other benchmark tools running in the user space. It also eliminates overheads and skews caused by file system layer. Such overheads and skews may give quite different performance results for the same measured system. Table 1 shows measured random write performance results of a same SCSI disk under different file systems (Ext2, Ext3, and Ext2 with Sync flag) using PostMark, IoZone, and Bonnie++, respectively. It is clearly shown that although our measured storage is exactly same, these benchmark tools produce completely different performance results because of different file systems. Figure 2 plots the measured random write performance of a Seagate SCSI disk using N-SPEK and IoMeter, respectively. It is interesting to observe that throughputs produced by IoMeter fluctuate dramatically between 0 and 300 IOPS (IO per second) while those produced by N-SPEK are fairly consistent over time. The fluctuation of the throughputs produced by IoMeter results mainly from the buffer cache. Because of the existence of the buffer cache, throughputs are high at times. However, during a dirty cache flushing period, measured throughput approaches to zero because the system is busy and not able to respond to normal I/O requests. Our N-SPEK module, on the other hand, produces accurate and stable throughput values over time.

The accuracy of N-SPEK is also evidenced when we measure the sequential read throughput of a SCSI disk as shown in Figure 3. In this figure, throughput changes periodically between 55MB/s and 39MB/s and the total data accessed in each period is 18GB which approximately equals to the formatted disk size. With Zoned Constant Angular Velocity (ZCAV) scheme, a modern SCSI disk has more sectors on outer tracks than inner tracks. As a result, accessing sectors on outer tracks is faster than inner tracks giving rise to the periodic throughput change as shown in the figure.

To evaluate system availability, we need to inject faults at various parts of the system. Fault injection is commonly used in fault-tolerance community to verify fault tolerant systems or to study system behaviors [1, 5]. It is also adopted for the analysis of software RAID system availability [2] and measurement of networked service availability [10]. There are three types of fault injection modules in N-SPEK to support availability evaluation. By using these modules, users can introduce different types of faults to different parts of a networked storage system under test and measure the performability of the tested system at degraded modes. These modules are:

Network fault injector. It resides on a network bridge

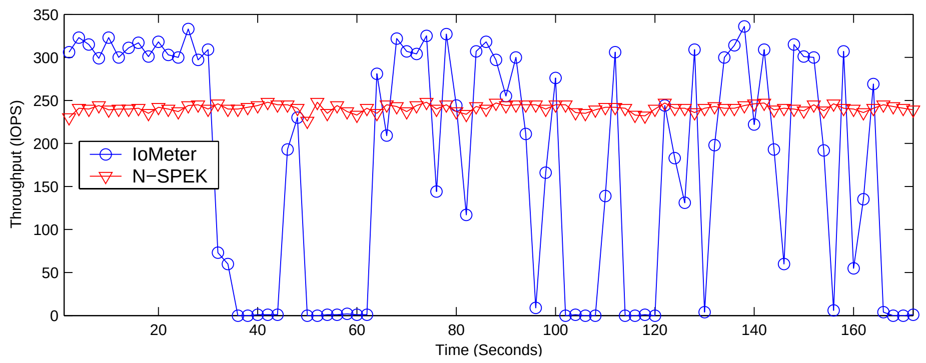


Figure 2. Measured throughputs for random write with block size being 16KB. The average IOPS of IoMeter is 175 while that of N-SPEK is 240. And the dynamic result of IoMeter fluctuates between 0 and 300 because of buffer cache effects while that of N-SPEK keeps consistent.

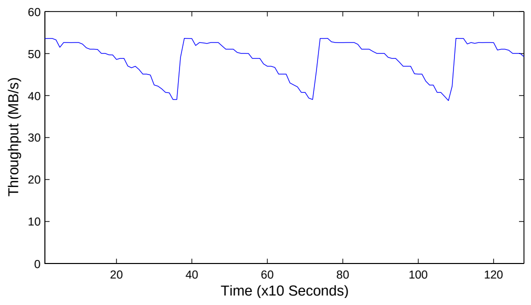


Figure 3. Measured throughput on the Seagate SCSI disk using N-SPEK.

on the network path between a worker and the measured storage target. It injects unexpected events to network traffic traveling through the bridge by adding excessive delays and dropping packets with a configurable packet loss rate. Note that TCP provides reliable transport over the Internet through flow control, time-out, and retransmission mechanisms. Many network faults including hardware failures and software failures in the network result in excessive delays at the transport layer. Injecting excessive delays at TCP layer mimics various network faults. We call this type of faults *delay fault*. Our fault injector makes use of a program that controls the existing dummynet [15] package in FreeBSD, a network traffic control and shaping software that was previously used by other researchers [13].

Storage fault injector. Its main purpose is to emulate a normal SCSI disk that can be used directly by storage systems, and to generate some kinds of transient and sticky SCSI disk errors that compromise system availability. Previous researchers [2, 6] have also used disk emulation techniques to do fault injections and performance evaluation. Our storage fault injection module is a RAM based virtual SCSI disk residing on the storage target. It exports itself as a normal SCSI disk and is utilized by the target under test.

Controller fault injector. Besides hardware failures of a storage controller, major source of faults of a controller can be attributed to malfunction of CPU and RAM. Normal operations of a controller can be compromised if needed CPU and/or RAM resources are unavailable. Directed by configurable parameters, a controller fault injector can take away most of CPU and/or memory resources from normal storage controller operations by adding unrelated CPU loads and memory loads to the controller.

All these injected faults are user configurable and can be set before a test experiment. Users can set them as sticky or transient. A sticky fault will influence tested systems during the entire measurement period while a transient fault occurs to the system in a short period. For example, users can set a network delay fault to 1ms during an entire test process as a sticky fault or add a packet loss rate of 0.0005 only in the third minute as a transient fault. By introducing these faults individually or simultaneously into a system, users can well simulate different failure situations and obtain performability of the measured system.

Unlike performance evaluation that has well-established metrics such as throughput and response time, there is no such a well-established performability benchmark available. Traditionally, availability has been measured as an average percentage of system "up" time. Brown and Patterson defined availability as a fluctuation of performance dynamics as opposed to a binary value to reflect different degree of performance degradations caused by component failures. Other researchers define performability as the average throughput multiplied by a measure of availability for cluster-based services [12]. Tsai [17] uses a time-dependent quality of service metric that is also adopted in [2]. Our method is to provide dynamics of performance over time under different fault conditions, and leave the performability definition and computation to users. In this way, user can obtain enough data and flexibly use their own definition of their interest, which is more informative to their research.

3. A Case Study: Performability Evaluation of iSCSI

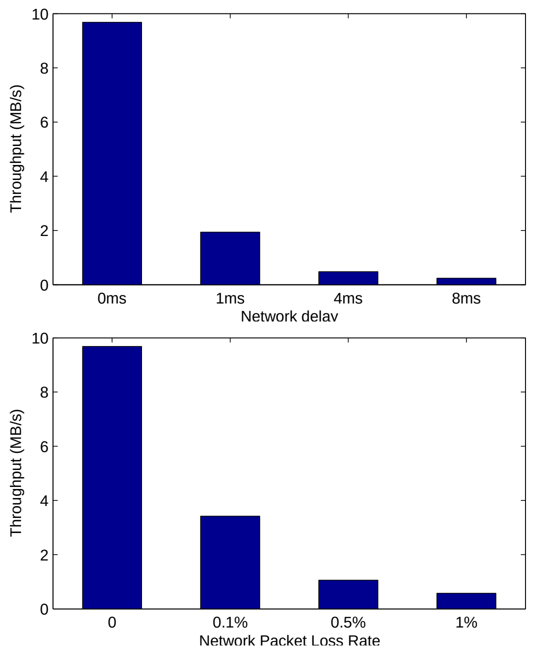
iSCSI is an emerging standard [16] to support remote storage access via encapsulating SCSI commands and data in IP packets. It was originally proposed by IBM, Cisco, HP, and etc, and has recently become an industry standard approved by IETF. It enables clients to discover and access SCSI devices directly via the matured TCP/IP technology and existing Ethernet infrastructures. Previous work on iSCSI mainly concentrated on its performance evaluation and potential improvement [13, 19, 11, 7]. By using our N-SPEK, we evaluate the performability of a popular iSCSI implementation and observe that the performance of iSCSI degrades dramatically in case of faults, and it can rapidly recover to normal status once such faults are removed or corrected.

3.1. Experimental Environment

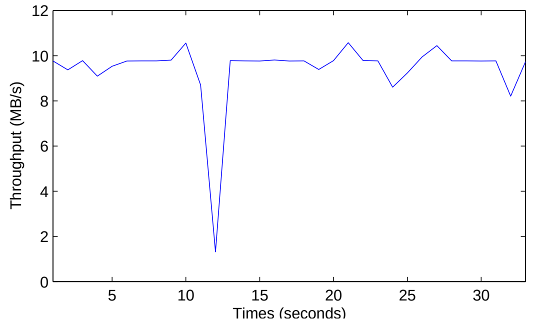
In our experiments, we use four PCs acting as N-SPEK Controller, N-SPEK Worker, network bridge, and iSCSI storage target, respectively. The network fault injection module is installed on the bridge and the controller fault injection module resides on the iSCSI target. The iSCSI target uses an N-SPEK storage fault injection module, an emulated disk, as a storage device. All PCs are equipped with one Pentium III 866MHz CPU, 512M PC133 memory, and one Intel Pro1000 Gigabit NIC except for the bridge that has two NICs. The N-SPEK worker is connected to a NetStructure 470T Gigabit switch through the bridge using a crossover cable while other three PCs are connected to the switch directly. All PCs run Redhat Linux 7.3 with recompiled standard 2.4.18 kernel except for the bridge that uses FreeBSD 4.6. The iSCSI implementation we choose comes from University of New Hampshire [18].

3.2. Performability Results

We measured performability under different faults with different request patterns and found that for different request patterns, the performability shows similar tendency under same fault conditions. Because of the page limit, we report here only the performability results under sequential read with 8KB block size. We mainly measure the performability with single fault injections to check effects of different faults. In all experiments, we let workers generate next request only if it successfully receives a response for the previous request. If it gets a response indicating that the previous request failed, was timeout, or finished with errors, it will retry the previous request.



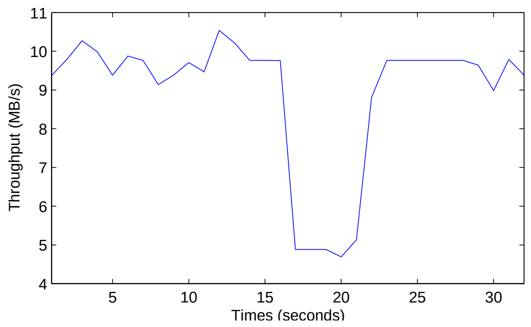
(a) Sticky Delay Faults and Packets Loss Faults Injected



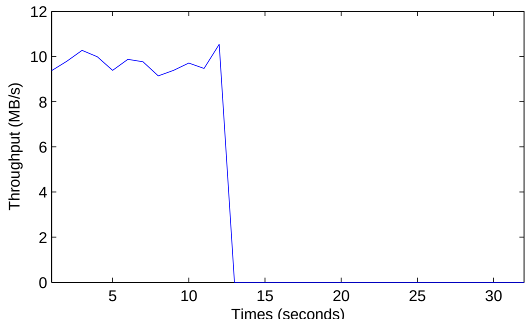
(b) Transient Packets Loss Faults with 0.005 Packet Loss Rate Injected

Figure 4. iSCSI Performability under Network Faults.

We plot throughput results of iSCSI under different network faults in Figure 4. From Figure 4 (a), we observe that iSCSI performance degrades rapidly with increase of delay faults and packet loss rate. For example, when the network delay fault increases from 0ms to 1ms, the iSCSI performance drops from 9.68 MB/s to 1.94 MB/s, a 398% reduction. And a 0.001 packet loss rate fault will degrade system performance from 9.68 MB/s to 3.42 MB/s. Since many iSCSI deployments share the same network with other applications, such network congestion can greatly impair the performability of the iSCSI storage system. Figure 4 (b) is the measured instant throughputs of the iSCSI target under transient packet loss that sustains for about 2 seconds. During the 2 seconds, system suffers from a low throughput. In general, we find that system runs with a degraded performance during the period that network faults are injected and it rebounds back to the normal status rapidly once the faults are removed.



(a) Transient Disk Errors Injected



(b) Sticky Disk Errors Injected

Figure 5. iSCSI Performability with Disk Faults

The iSCSI performance under transient and sticky disk faults are shown in Figure 5 (a) and (b), respectively. During the transient fault injection period, we let the storage fault injector reply each I/O request with a successful response or with a correctable error response equally likely, i.e. 50% chance of errors. Such transient faults result in a nearly 50% performance drop during the period of transient errors injected. To understand why there is such a big performance drop, we analyzed the source code of iSCSI implementation. We noticed that in this iSCSI implementation, the storage controller simply returns response of a request back to an initiator without checking its result. Therefore for a failed request, the iSCSI controller sends the response containing error message back to the client and the client simply retries this request via network again. A better policy would be to let the iSCSI controller retry the failed request directly at controller side and return a response with successful message or with failed message after predefined trials. In this way, an iSCSI controller can find and handle most transient errors locally minimizing unnecessary network traffic and thus improving iSCSI performability. With sticky uncorrectable disk errors injected, iSCSI performance reduces to zero as shown in Figure 5 (b). It can be seen that iSCSI performability is greatly influenced by the performability of storage devices it uses. A storage device that has some kinds of redundancy or mirroring such as RAID can greatly improve its performability and also enhance the performability of higher level services.

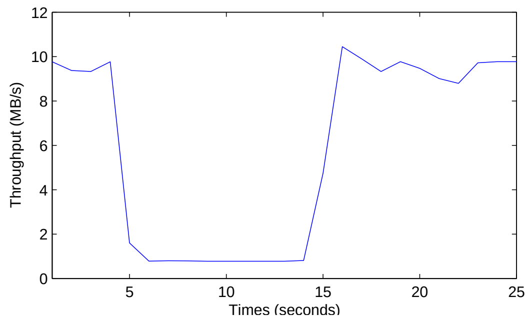


Figure 6. iSCSI Performability under Transient Controller Faults

We also measured iSCSI performance under transient controller faults and show the results in Figure 6. Our controller fault injector takes over 98% of the CPU time during the fault-injection period and iSCSI only has 10% throughput compared with normal case. During a fault injection period, the controller CPU becomes the bottleneck although Linux scheduler still gives the iSCSI process some time slices to run. Once the CPU fault is removed, iSCSI returns to normal performance rapidly.

Figure 7 shows the performance results of iSCSI under multiple faults. With both network delay faults and controller CPU faults injected, iSCSI only gets a throughput of around 0.39 MB/s, much lower than the case of any single fault. After the network delay fault is removed, the performance of iSCSI recovers to around 0.79 MB/s, almost identical to the performance it achieves with single controller CPU fault. Its performance recovers to the normal value rapidly after the CPU fault is removed.

4. Conclusions

In this paper, a new benchmark tool has been presented for measuring performability (performance + availability)

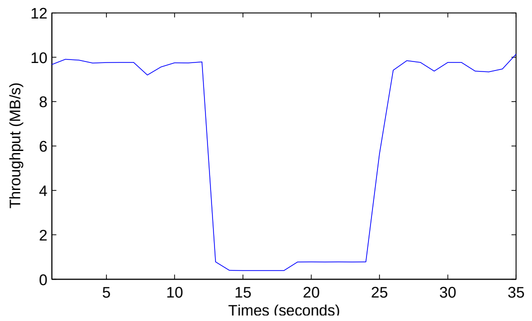


Figure 7. iSCSI Performability under Transient Network and Controller Faults

of SANs. The benchmark tool, referred to as N-SPEK (Networked Storage Performability Evaluation Kernel module), enables a SAN architect to measure available performance of a SAN under different failure conditions and to analyze performance changes as result of various component faults. In its current version, N-SPEK tool can generate user configurable storage workloads, network faults, disk failures, and controller faults. It then collects and records performance results based on the workloads and fault injections. A prototype N-SPEK has been tested on a simple iSCSI based SAN to demonstrate its functionality. Our future work is to support more network protocols and controller OS, to introduce more realistic faults, and to perform measurements on various SAN systems.

Acknowledgments

This research is sponsored in part by National Science Foundation under grants MIP-9714370 and CCR-0073377. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.The third author's work is partially supported by the Center for Manufacturing Research at Tennessee Tech University.

References

[1] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, and D. Powell. Fault injection and dependability evaluation of fault-tolerant systems. *IEEE Transactions on Computers*, 42(8):913–923, 1993.

[2] A. Brown and D. A. Patterson. Towards availability benchmarks: A case study of software RAID systems. In *Proceedings of the 2000 USENIX Annual Technical Conference*, pages 263–276, San Diego, CA, June 2000.

[3] D. Capps and W. D. Norcott. Iozone filesystem benchmark. http://www.iozone.org/.

[4] R. Coker. Bonnie++ benchmark tool. http://www.coker.com.au/bonnie++/.

[5] S. Dawson, F. Jahanian, and T. Mitton. ORCHESTRA: A fault injection environment for distributed systems. Technical Report CSE-TR-318-96, University of Michigan, 1996.

[6] J. L. Griffin, J. Schindler, S. W. Schlosser, J. S. Bucy, and G. R. Ganger. Timing-accurate storage emulation. In *Proceedings of the Conference on File and Storage Technologies (FAST)*, pages 75–88, Monterey, CA, Jan. 2002.

[7] X. He, Q. Yang, and M. Zhang. Introducing SCSI-To-IP cache for storage area networks. In *Proceedings of the 2002 International Conference on Parallel Processing*, pages 203–210, Vancouver, Canada, Aug. 2002.

[8] Intel. Iometer, performance analysis tool. http://www.intel.com/design/servers/devtools/iometer/.

[9] J. Katcher. PostMark: A new file system benchmark. Technical Report 3022, Network Appliance, 1997.

[10] X. Li, R. Martin, K. Nagaraja, T. Nguyen, and B. Zhang. Mendosus: A SAN-based fault-injection test-bed for the construction of highly available network services. In *Proceedings of 1st Workshop on Novel Uses of System Area Networks (SAN-1)*, Feb. 2002.

[11] K. Meth. iSCSI initiator design and implementation experience. In *19th IEEE Symposium on Mass Storage Systems*, Adelphi, MD, Apr. 2002.

[12] K. Nagaraja, N. Krishnan, R. Bianchini, R. Martin, and T. Nguyen. Evaluating the impact of communication architecture on the performability of cluster-based services. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA 9)*, Anaheim, CA, Feb. 2003.

[13] W. T. Ng, B. Hillyer, E. Shriver, E. Gabber, and B. Ozden. Obtaining high performance for storage outsourcing. In *Proceedings of the Conference on File and Storage Technologies (FAST)*, pages 145–158, Monterey, CA, Jan. 2002.

[14] A. Park and J. C. Becker. IOStone: a synthetic file system benchmark. *Computer Architecture News*, 18(2):45–52, June 1990.

[15] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[16] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner. iSCSI draft standard. http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-18.txt, 2002.

[17] T. K. Tsai, R. K. Iyer, and D. Jewitt. An approach towards benchmarking of fault-tolerant commercial systems. In *Symposium on Fault-Tolerant Computing*, pages 314–323, Sendai, Japan, June 1996.

[18] UNH. iSCSI reference implementation. http://www.iol.unh.edu/consortiums/iscsi/.

[19] K. Voruganti and P. Sarkar. An analysis of three gigabit networking protocols for storage area networks. In *20th IEEE International Performance, Computing, and Communications Conference*, Phoenix, Arizona, Apr. 2001.

[20] J. Ward, M. O'Sullivan, T. Shahoumian, and J. Wilkes. Appia: Automatic storage area network fabric design. In *Proceedings of the Conference on File and Storage Technologies (FAST)*, pages 203–217, Monterey, CA, Jan. 2002.