# Fault Recovery Designs for Processor-Embedded Distributed Storage Architectures with I/O-Intensive DB Workloads*

Steve C. Chiu
*Benedict College*
*chius@benedict.edu*

Alok N. Choudhary
*Northwestern University*
*choudhar@ece.northwestern.edu*

Mahmut T. Kandemir
*Penn State University*
*kandemir@cse.psu.edu*

## Abstract

*Fault recovery has become an essential capability for systems that process large data-intensive workloads. Processor-embedded distributed storage architectures offload user-level processing to the peripheral from the host servers. Our earlier work investigated the performance benefits of such architectures for disk- and MEMS-based smart storage devices. In this paper, we focus on the issue of fault recovery. We propose recovery schemes for TPC-H based workloads, and evaluate several recovery scenarios applicable to both disk- and MEMS-based smart storage architectures.*

## 1. Introduction

Apart from meeting the challenges of latency, bandwidth and storage capacity, effective storage architectures must also address the issue of reliability. For optimal bandwidth and load balancing, data in a storage system are usually partitioned across all storage nodes in ways that exploit the access patterns of the target workloads. While many methods exist in providing fault recovery, the basic principles used by these methods are similar. Existing fault tolerance methods enable a storage system to withstand failures with minimal interruption of processing; and the system is said to be operating in the degraded mode. This entails a real-time data recovery method to be deployed to reconstruct the data lost in the failed node; and the reconstructed data must be placed on another spare node before the system returns to its normal operation, when the failed node can be repaired or replaced offline at a later time.

This paper proposes and evaluates data reconstruction designs for distributed smart storage architectures. These proposed designs are intended to balance the demand for the best possible performance under the constraint of required level of fault tolerance. To present our designs and methodologies, Section 2 briefly reviews the work related to smart storage architecture, and the background work for fault recovery. In Section 3, we propose our data recovery schemes. Section 4 explains the workloads and scenarios for our evaluation. Section 5 presents our experiments as well as simulation platform and model. Conclusions and future work are discussed in Section 6.

* This work was supported in part by NSF Grants CNS-0406341, CCF-0444405, Sandia National Laboratory Contract 28264, and DOE Award DE-FC02-01ER25485; and conducted at the USCL Laboratory of Northwestern University.

## 2. Background

### 2.1. Redundant Array of Inexpensive Disks

To improve data bandwidth and reliability, RAID was proposed in the mid-1980s for high storage performance by clustering multiple disks to appear as a single and large disk [6]. The methods of data striping and mirroring of RAID provided techniques that are valuable to a distributed storage environment. Configuration of a RAID also depends on the characteristics of the workload, impacting system size, data partition and data replication for fault recovery. The primary objective of RAID is two-told: fault recovery and data availability – for normal as well as degraded operating modes of the storage system.

### 2.2. Processor-Embedded Smart Storage

Our previous work [2,3] proposed distributed disk- and MEMS-based smart storage systems, and provided the performance and processing models to evaluate their potential benefits. Such a system takes advantage of the increasing processing power on the storage device by performing I/O-intensive operations directly at the device and in parallel across multiple devices. This approach helped reduce both the system and storage network traffic, and offloaded the host processor, thus increasing the aggregate system power and I/O bandwidth. The objectives of the system are to: (1) maximize parallelism for data-intensive portion of the workload, (2) use primitives for resource availability, and (3) maximize (application-level) code isolation at each smart storage node. Figure 1 depicts

Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005)
0-7695-2318-8/05 $20.00 © 2005 IEEE

the high-level architectural design of the smart storage system. A disk-based smart node (SN) is also called a smart disk (SD). Likewise, a disk-based smart storage group (SSG) is also called a smart disk group (SDG). For MEMS-based systems, the terms SM and SMG are used, respectively.

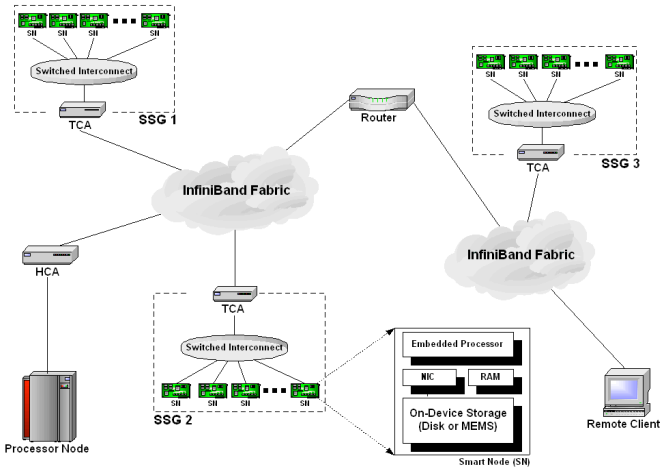


**Figure 1. Smart storage architecture with smart nodes (SN) organized into multiple smart storage groups (SSG) for disk- and MEMS-based systems on the InfiniBand [4] storage network.**

## 2.3. Operations Bundling and Processing Model

The work in [5] proposed the "operations bundling" technique to combine elementary database operations that share common data dependencies. For example, the *Sort*, *Group* and *Aggregate* nodes executed consecutively in the query plan depicted in the upper portion of Figure 2 can be combined into a single "*AggGrpSort*" node (thus called a bundle). This new execute node is semantically equivalent (in SQL) to the original sequence of nodes, and can enable the entire query to be processed with a higher efficiency. For distributed storage architectures, operations bundling is a logical approach to improve performance. With software architecture specified in [2], processing model of a smart storage system will be as illustrated in Figure 3, using disk-based systems as an example. An application inputs a query or executable code through the parser that decides the input as either valid (and passes it to the optimizer) or invalid (and passes it to the error handler). The optimizer takes database catalogs and system parameters as input, and constructs cost models to optimize the query. In the case of executable code, only the system parameters are used for code optimization. The optimized query or code is then distributed to the SNs of the involved SSGs, and execution commences. Processing leading up to this stage is performed by the client (host) before the control is handed to the designated SN within each SSG. Upon completion, the system responds to the user with the final results through the designated SNs. The software architecture proposed in [2] provides an on-device kernel (SD OS) to support the execution control, data access, memory buffer management, signaling, code/data recovery, etc. to facilitate this processing model.

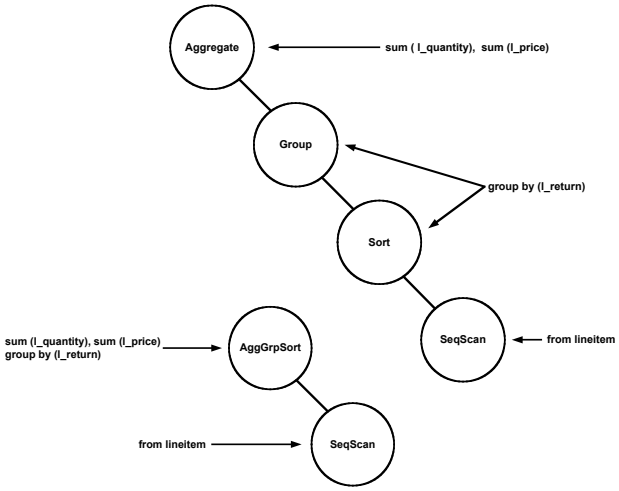


**Figure 2. Two user queries showing operations bundling in SQL syntax.**

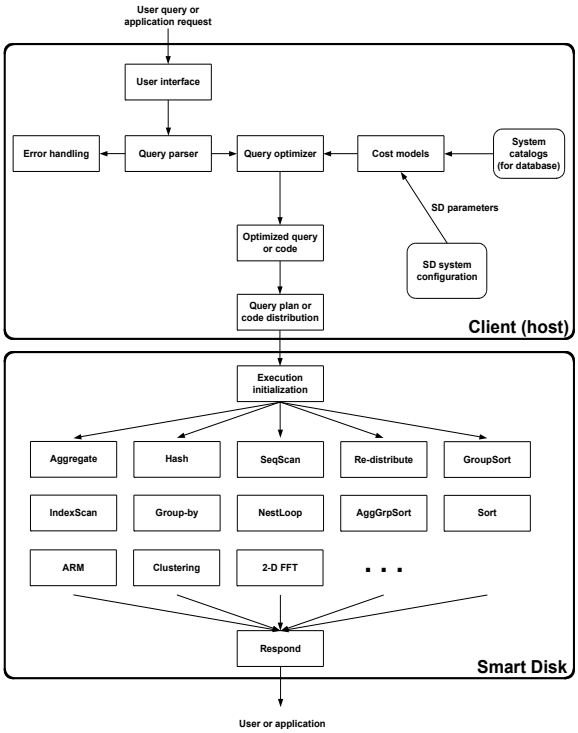


**Figure 3. Processing model for the smart storage system for various DB and non-DB workloads.**

### 2.4. MEMS-Based Storage Devices

MEMS are micron-sized devices fabricated from photolithographic processes. Instead of using a rotating spindle and dividing a disk into sectors, tracks and cylinders, MEMS-based storage devices consist of a moving rectangular media sled and an array of read/write tips. The media sled is spring-mounted over the tips and can be moved by actuators in the X, Y and Z directions. While the system "seeks" in the X direction and reads/writes data in the Y direction, it also moves in the Z direction to control the distance between the tips and the media sled. For example, in Carnegie Mellon University's CHIPS [9] storage device, the MEMS-based system contains an array of 80 x 80 tips with each tip accessing a region of 2,500 x 2,500 bits. The characteristics of MEMS-based storage necessitate re-evaluation of several design issues, including data layout and I/O scheduling.

### 2.5. TPC-H Benchmark

Developed by the Transaction Processing Performance Council, The TPC-H benchmark [8] consists of a suite of ad-hoc queries as well as concurrent data modifications. The queries and data populating the databases have been designed to represent decision support systems (DSS) that manipulate large data sets and execute complex queries. The performance metric used by TPC-H reflects a DSS's processing power and query throughput. Each TPC-H query is composed of basic database operations such as *scan*, *sort*, *join*, *group-by*, *aggregate*; and may also include data re-distribution functions such as *hash*. The queries together cover both intra- and inter-SDG communication and data filtering in their access patterns.

## 3. Proposed Data Recovery Schemes

### 3.1. Mirror Scheme with Spare Storage

Using disk-based SN (i.e. the SD) and SSG (i.e. the SDG) as an example, Figure 4 proposes our first scheme. SDG A holds row-wise data and SDG B, its mirror, holds the column-wise data, on their SDs: D0, D1, D2 and D3. Each SDG processes its respective application, application A and application B, each with different I/O access patterns. The fault condition occurs when D1 of SDG A malfunctions midstream during processing. To recover the data 1-5-9-13 originally held by D1 onto disk *Sp* of SDG A, a parallel *read* of D0, D1, D2 and D3 of SDG B is performed. The recovered *Sp* then joins with D0, D2 and D3 of SDG A to resume the processing.

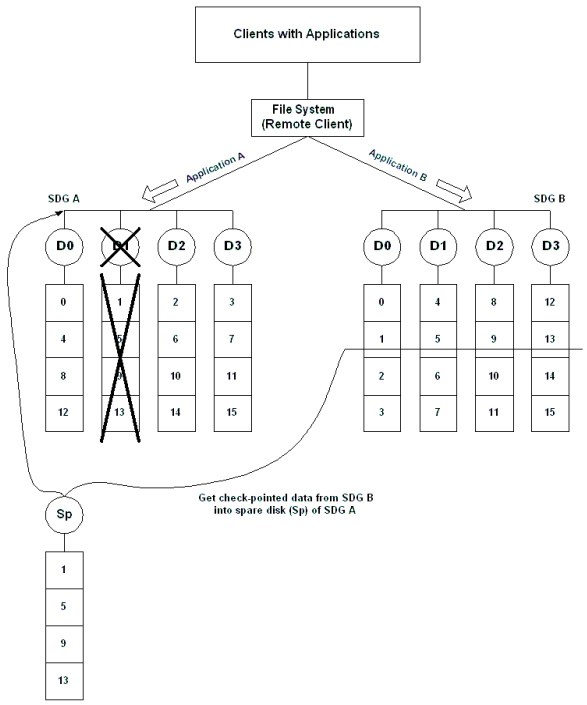


**Figure 4. Mirror recovery scheme with dedicated spare storage in SDG A.**

### 3.2. Mirror Scheme with Load Migration

In this scheme, the workload (application code only) is migrated from SDG A to SDG B, and both applications are processed on SDG B after the recovery. Since SDG B holds the needed data already, fault recovery operation overhead is reasonably reduced. However, overall system performance will be negatively impacted since: (1) SDG B must now process both workloads, and (2) the migrated workload on SDG B can no longer exploit the data access pattern provided by SDG A for maximum bandwidth and I/O throughput.

### 3.3. Mirror Scheme with Load Re-distribution

For large data-intensive applications, e.g. scientific simulations or DSS databases, it is common to perform data dumps or backups of intermediate results. Check-pointing stores the results produced from the intermediate steps to enable the processing to resume without complete restarts. In Figure 5, SDG A mirrors the intermediate results to SDG B. Then, D1 of SDG A fails during an intermediate step. The most recent check-pointed data are distributed between the remaining SDs, and the processing of application A is continued. The objective is to amortize the cost of load re-distribution with all available SDs in post-recovery processing.

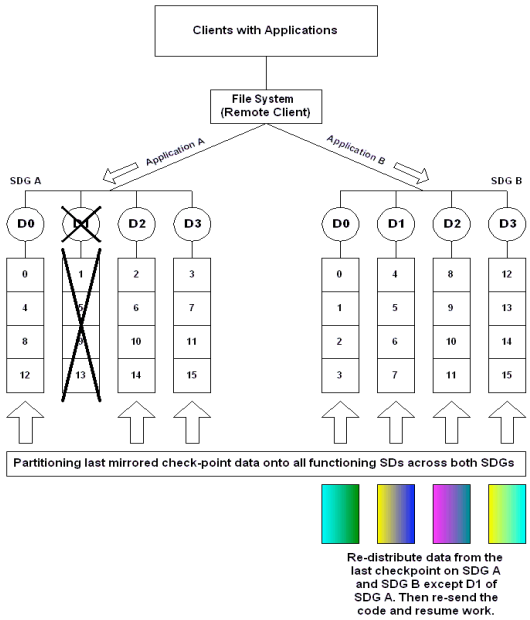


**Figure 5. Mirror recovery scheme with load re-distribution.**

### 3.4. Parity Scheme with Single Parity Device

Figure 6 shows a scheme based on the level 4 RAID [6] configuration by providing block-level data striping with a dedicated parity SD. The Figure contrasts the scheme with that of a centralized I/O server, using a 100-block I/O request as an example. When a SD fails, data recovery is performed by XORing the last check-pointed data on the remaining SDs with those on the parity disk, e.g. SD *P* for the SDG.

## 4. Workloads and Scenarios

We used the following tasks as our workloads: the database *Scan* and TPC-H $Q_1$. Again, using disk-based systems as examples, scenario 0 for each workload represents normal mode of operation, and was used as the basis for comparison with the other recovery scenarios.

### 4.1. Scenarios 0, 1 and 2 for the *Scan* Workload

In scenario 0, a *Scan* (i.e. SQL SELECT) operation is performed to select tuples from the input table, and is allowed to complete without disruption, denoting the normal operation of the *Scan*. Scenario 1 simulates the same *Scan* operation, except that a fault is induced on one of the SDs within the main SDG. To recover the data lost in the failed SD, data (the original partition of the input table) are sent from its mirroring SDG to the spare SD within the main SDG. Then the operation is resumed from the beginning. For scenario 2, a dedicated "parity" SD is used to reconstruct the lost data, as discussed in Section 3.4.

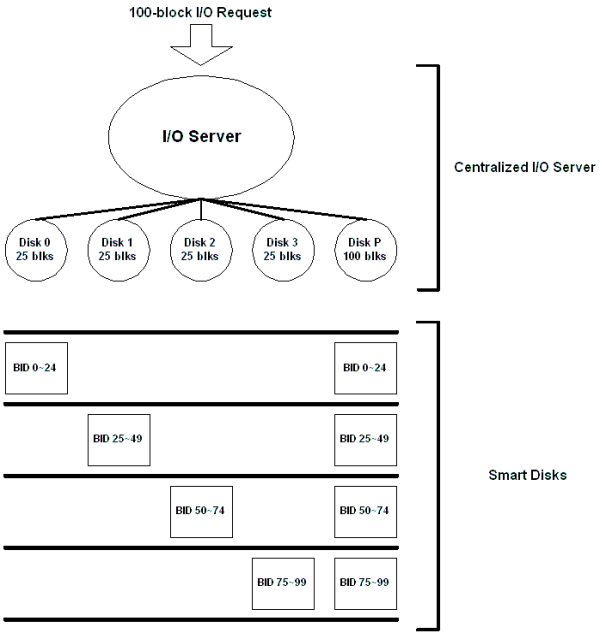


**Figure 6. Parity-based scheme with a 100-block I/O request.**

### 4.2. Considerations for the TPC-H $Q_1$ Workload

TPC-H $Q_1$ can be processed with the following query plan: *Scan*→*Sort*→*Group-by+Aggregate+LocalSort*, where *Group-by+Aggregate+LocalSort* represents an "operations bundle". While the scenarios for $Q_1$ are similar to those for *Scan*, they are more complex due to the multiple primitives (nodes) involved. Our $Q_1$ scenarios incorporate the design to update the mirror (or parity) SDG with intermediate results at the granularity of a primitive. Hence post-recovery processing need only be backtracked by one primitive (or one operations bundle).

### 4.3. Scenarios 0, 1, 2, 3 and 4 for the TPC-H $Q_1$ Workload

In scenario 0, the processing of $Q_1$ is to be completed without interruption. Scenario 1 assumes that the main SDG, i.e. the SDG that is performing the $Q_1$ query, has a spare SD. Processing of $Q_1$ continues through *Scan* and *Sort*, then a fault condition occurs just before the *Group-by+Aggregate+LocalSort* is to commence. The lost data is recovered by sending the data from the mirror to the spare

SD node. Post-recovery processing resumes with *Group-by+Aggregate+LocalSort* within the main SDG. Note that unlike the scenarios for *Scan*, only the most recent check-pointed data are required to be sent during data recovery for these scenarios, not the original input data.

For scenario 2, the remaining processing load is migrated to the mirroring SDG after one of the SDs in the main SDG faults. Note that only code is needed because the mirroring SDG already has the intermediate results from the last check point. The mirroring SDG will also be responsible for completing the processing. Scenario 3 attempts to evaluate the tradeoffs of having all but the faulted SD participate in post-recovery processing. As the fault condition is induced, the check-pointed data are re-distributed onto the SDs across both the main SDG and the mirroring SDG, except for the failed SD. The processing of $Q_1$ then resumes with all the remaining SDs from both SDG A and SDG B until completion.

Scenario 4 for parity-based recovery scheme works as follows. After *Scan* is executed, parity blocks are built and stored onto the parity SD. Likewise, new parity blocks are built and stored after *Sort* is completed. Hence, when the fault condition occurs, the lost data on the failed SD are reconstructed using last check-pointed data from the remaining SDs and the parity SD. The post-recovery processing is similar to the other scenarios. Note that this scenario also assumes the existence of a spare SD to hold the reconstructed data for post-recovery processing.

## 5. Experiments and Simulation Model

We evaluate the recovery schemes for both disk- and MEMS-based systems using the workloads and scenarios discussed in Section 4. Our platform consists of 9 Pentium III Linux PCs running at 500 MHz, each with 9 GB of disk space; and the PCs are interconnected via fast Ethernet. A 64-MB maximum on-device memory has been configured for our simulation environment. We used the TPC-H database generator [8] to populate synthetic data into input tables with scale factor of 1.0, denoting a 1-GB size for the input data (i.e. the *Lineitem* table).

### 5.1. Simulation Environment and Structure

A simulation environment was developed to simulate computation, communication and I/O behaviors of our system. The environment employs the *DiskSim 3.0* [1] storage simulator to estimate the cost of accessing storage devices, and is programmed to include the algorithms and data needed to process the given workloads. Message-passing interface (MPI) was used for data and signal communication during processing. Figure 7 depicts the simulation structure for a single SN, where processes representing the on-device embedded processor, storage controller, DMA engine and storage mechanism interact with one another to service the requests by a workload. The Figure also shows two *DiskSim* validated device models for MEMS and disks, which are used to obtain accurate I/O access times for the respective devices during our simulations.

### 5.2. Disk- and MEMS-based Device Models

A validated *DiskSim* module for the HP C2490A hard disk is used for our comparison between MEMS-based and disk-based active storage devices, and the performance characteristics of HP C2490A disk are as specified in [3]. The device-level models for our MEMS-based storage are from those described in [3] and [7]. Of the three MEMS-based storage devices G1, G2 and G3, the G3 device represents the most advanced device with a 1000 Kb/s data rate and a maximum throughput at 320 MB/s. These MEMS devices have been shown to exhibit a significantly higher I/O performance than disks in device-level simulations, and we are interested in their impact at the system level as well. Table 1 specifies the essential parameters for the MEMS-based storage devices as in [7].

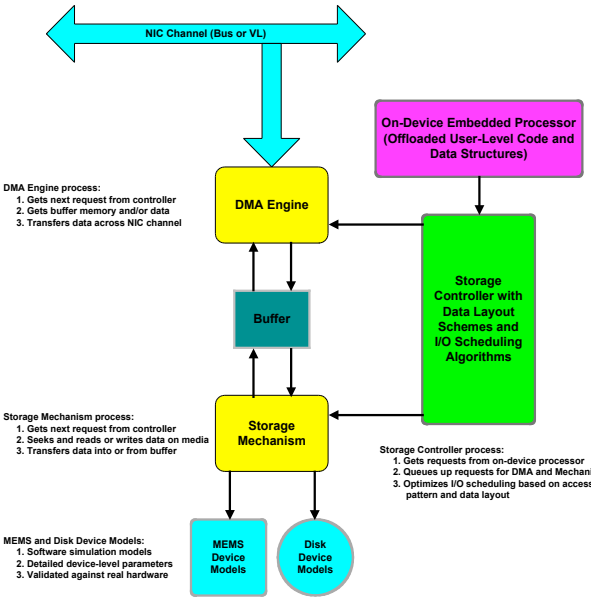


**Figure 7. DiskSim simulation model.**

#### 5.2.1. The DiskSim Storage Simulator

Developed by G. Ganger *et al.* originally at the University of Michigan, *DiskSim* includes modules that simulate disks, controllers, buses, device drivers, request schedulers, disk block caches, and disk array data layouts;

and has been accurately validated against several production disks. *DiskSim* also included accurate MEMS-based storage device modules. Together with its disk drive modules, *DiskSim* was used as the comparison platform for disk- and MEMS-based storage systems investigated in this paper, and our previous work in [2] and [3].

**Table 1. Parameters for three types of MEMS-based storage devices**

| Parameter | G1 | G2 | G3 |
|---|---|---|---|
| Bit width (nm) | 50 | 40 | 30 |
| sled acceleration (g) | 70 | 82 | 105 |
| access speed (Kbits/s) | 400 | 700 | 1000 |
| X settling time (ms) | 0.431 | 0.215 | 0.144 |
| total number of tips | 6400 | 6400 | 6400 |
| number of active tips | 640 | 1280 | 3200 |
| max throughput (MB/s) | 25.6 | 89.6 | 320 |
| Number of media sleds | 1 | 1 | 1 |
| per-sled capacity (GB) | 2.56 | 4.00 | 7.11 |
| bi-directional access | no | yes | yes |

### 5.3. Recovery on Disk-based Storage Systems

Figure 8 shows the execution times for the three scenarios for *Scan*. It is noted that while the parity-based scheme (scenario 2) required a smaller system size as compared to the mirroring scheme (5 vs. 9 total number of SDs), it incurred a higher recovery time due to the I/O cost associated with building and re-constructing data using parity blocks. This parity-based recovery scheme also incurred a higher data communication cost due to the bottleneck at the single parity device.

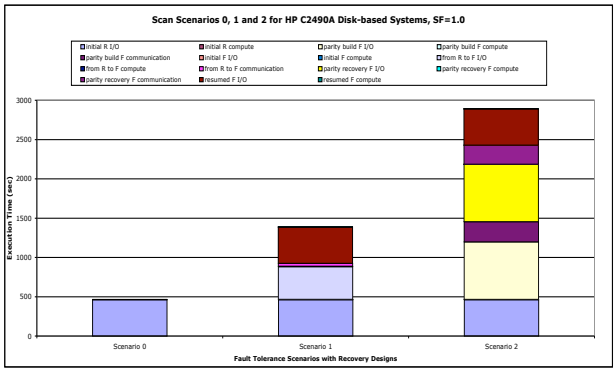


**Figure 8. Scenarios for *Scan* on disk-based storage at scale factor 1.0.**

Figure 9 presents the execution times for the scenarios evaluated for TPC-H $Q_1$. Most noticeable is the communication cost for scenario 3, where load re-distribution is performed after the fault. *Group-by* I/O, the dominant component of post-recovery processing, is reduced by approximately 75% to 80% in scenario 3 when compared to the other scenarios. However, such saving was overwhelmed by the communication cost of workload re-distribution, which was exacerbated by the serialization of I/O requests by the SCSI protocol implemented for our platform. These results suggest strongly for non-blocking switched interconnects, such as InfiniBand.

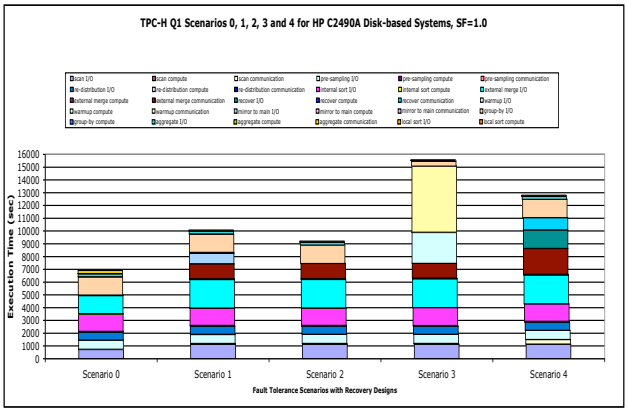


**Figure 9. Scenarios for TPC-H $Q_1$ on disk-based systems at scale factor 1.0.**

### 5.4. Recovery on MEMS-based Storage Systems

Figure 10 presents the execution times for scenarios 0, 1 and 2 for the *Scan* on G3 MEMS-based systems. Note the diminishing I/O dominance with scenario 2, instead the communication cost attributed to building and using the parity blocks took over as the dominant component due to the reduced I/O cost by the MEMS devices.

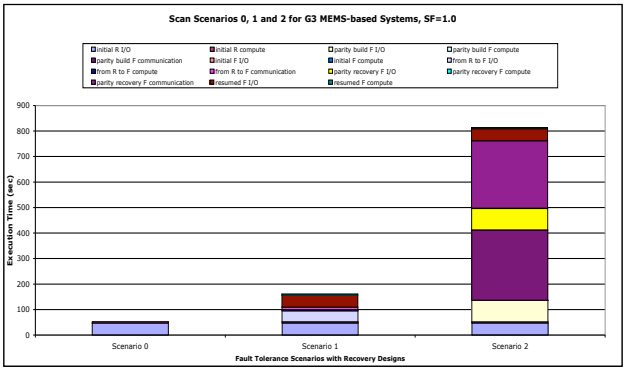


**Figure 10. Scenarios for *Scan* on MEMS-based (G3) storage systems at scale factor 1.0.**

Figure 11 continues with the trends as we observed in Figure 9, i.e. the dominating communication component attributed to the load re-distribution phase of scenario 3. This suggests that to fully benefit from the post-recovery load re-distribution design, a low-latency high-bandwidth interconnect is essential. On the other hand, scenario 2 incurred the least overall execution time, since the scheme only required the code to be migrated to the mirroring SDG. For data-intensive workloads such as *Scan* and $Q_1$, the benefit of this approach is significant. As the underlying storage devices shifted from slower disk-based devices to faster MEMS-based devices, scenario 1 became increasingly attractive. This is because in scenario 1 only one mirroring SD is involved in recovery and no mirroring SD is needed in post-recovery processing, whereas in scenario 2 all the mirroring SDs are involved in post-recovery processing. Should the mirroring SDG be also processing another workload simultaneously, its performance will be worse as that workload will be contending for the same SDG resources.

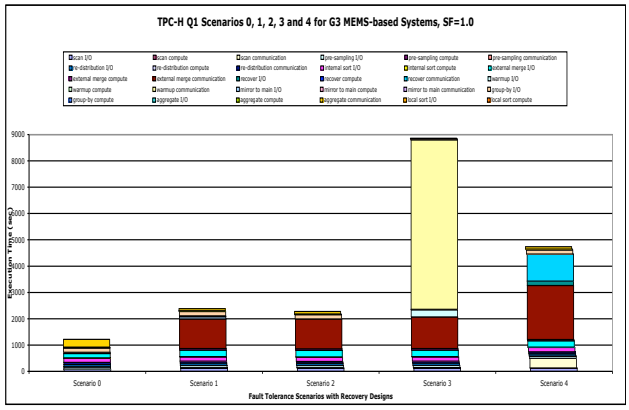


**Figure 11. Scenarios for TPC-H $Q_1$ on MEMS-based (G3) storage systems at scale factor 1.0.**

### 5.5. Additional Design Considerations

The fault tolerance schemes evaluated in this work must also be considered from the performance perspective of processor-embedded distributed storage, the hardware and software architectures of which were investigated in details in [2] and [3]. With new and emerging storage and interconnect technologies such as InfiniBand and MEMS, the offloading of application-level code to the storage device is becoming attractive and feasible. Ultimately this could result in advanced computation-storage integration becoming a viable alternative when configuring high-performance computing systems.

Figure 12 shows a higher scalability limit after an original system is changed by employing additional or new hardware, e.g. from disks to MEMS-based storage. The upgraded system can now handle a larger throughput, and still maintain the same acceptable execution time. A new improved limit is reached at the intersection between the new curve and the acceptable execution time line. Such consideration applies to both performance as well as reliability issues.

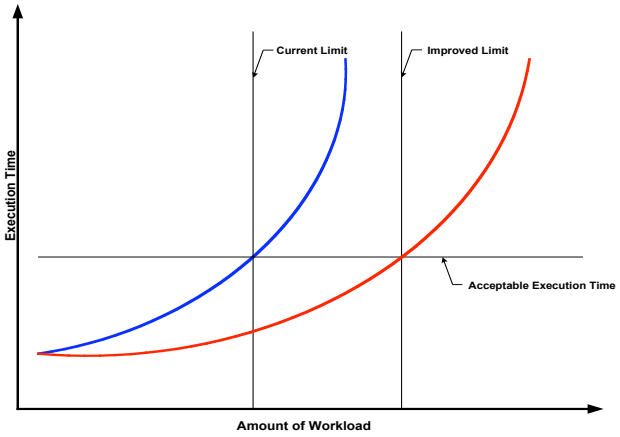


**Figure 12. Throughput vs. execution time before and after device changes.**

## 6. Conclusions and Future Work

For storage systems that process large data-intensive workloads, having effective and scalable fault tolerance capabilities is no less critical than high processing power and throughput. Furthermore, to be able to continue processing after a fault condition occurs is of significant importance, especially for workloads that require a tremendous amount of time to process, and would be inefficient to be resumed from the beginning each time a system fault occurs. In this paper, we presented several preliminary designs for fault recovery from the perspective of a distributed processor-embedded storage system, using I/O-intensive workloads.

This paper also suggests the need to exploit emerging storage technologies, such as MEMS, to cope with the performance issues that arise during the processing of I/O-intensive applications. We found it necessary to balance the tradeoffs among various fault recovery schemes. Schemes proposed in this work, when coupled with appropriate middleware, can be extended to computational or storage grids to further improve data access, management and analysis. We plan to investigate the impact of novel interconnect protocols, combined with emerging storage device designs, both at the device level and the memory system level (i.e. memory organization),

in the near future. While our investigation was conducted based on simulation, due largely to the unavailability of MEMS-based storage devices at the present time, it will be desirable to construct a hardware platform with the components discussed in this paper. Construction of the individual smart nodes (SN) along with their InfiniBand-based I/O interconnects should provide our future studies in active storage with further improved precision.

## Acknowledgement

We acknowledge the use of the Chiba City Cluster located at Argonne National Laboratory, the Distributed Optical Testbed, and the Linux prototyping platform in the Ultra-Scale Computing Laboratory at Northwestern University. We also thank Dr. George K. Thiruvathukal of Nimkathana Corporation for his support with the test data for our simulation work.

## References

[1] J. Bucy, G. Ganger *et al.*, *The DiskSim Simulation Environment Version 3.0 Reference Manual*, Technical Report No. CMU-CS-03-102, U.S.A.: Carnegie Mellon University, January 2003.

[2] S. C. Chiu, W. Liao, A. Choudhary and M. Kandemir, *Processor-Embedded Distributed Smart Disks for I/O-Intensive Workloads: Architectures, Performance Models and Evaluation*, Journal of Parallel and Distributed Computing, Vol. 64, No. 3, pp. 427–446. U.S.A.: Elsevier Inc., March 2004.

[3] S. Chiu, W. Liao and A. Choudhary, *Processor-Embedded Distributed MEMS-Based Storage Systems for High-Performance I/O*, Proceedings of the 18th International Parallel and Distributed Processing Symposium. U.S.A.: IEEE, April 2004.

[4] InfiniBand Trade Association (IBTA), *InfiniBand Architecture Specification Volume 1 Release 1.0.a*. U.S.A.: IBTA, June 2001.

[5] G. Memik, M. Kandemir and A. Choudhary, *Design and Evaluation of Smart Disk Architecture for DSS Commercial Workloads*, Proceedings of the International Conference on Parallel Processing. Taiwan: IEEE, September 2000.

[6] D. Patterson, G. Gibson and R. Katz, *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, Proceedings of the International Conference on Management of Data (SIGMOD). U.S.A.: ACM, June 1988.

[7] S. Schlosser, J. Griffin, D. Nagle and G. Ganger, *Designing Computer Systems with MEMS-based Storage*, Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). U.S.A.: January 2000.

[8] Transaction Processing Performance Council, *TPC Benchmark Suites and the TPC Soft Appendices for TPC-H and TPC-R Benchmarks*. U.S.A.: 1998.

[9] The Carnegie Mellon University CHIPS Project URL: http://www.lcs.ece.cmu.edu/research/MEMS.