Streaming Data from Disk Store to Application

Roger D. Chamberlain
Berkley Shands

Roger D. Chamberlain and Berkley Shands, “Streaming Data from Disk Store to Application,” in *Proc. of 3rd Int’l Workshop on Storage Network Architecture and Parallel I/Os*, September 2005, pp. 17-23.

Streaming Data from Disk Store to Application

Roger D. Chamberlain*† and Berkley Shands†
*Exegy, Inc. and †Washington University in St. Louis
roger@cse.wustl.edu, berkley@cse.wustl.edu

Abstract

Filesystem read performance is a critical issue for many applications. On the other hand, storage systems are typically configured with little or no knowledge of the properties of the applications that will use them. Here, we present empirical performance measurements of data throughput from disk store to application. A number of different disk system configurations are considered, and the properties of the applications are used to guide the design process. Particular attention is paid to the issues that arise when a large number of small files are present in the data set.

1. Introduction

It is not at all uncommon for highly data-intensive applications to be performance limited by the I/O subsystems of the execution platforms on which they are executing. It is also very common for applications to be uninterested in a large fraction of the data actually delivered to them. Both of the above situations call for the availability of a smart disk subsystem that is capable of high-bandwidth data delivery and is also has the capacity to perform application-driven data thinning.

In this paper, we provide empirically measured performance results for a system that does all of the above. The Exegy K•Appliance™ is a network appliance that incorporates both magnetic storage (up to 6 TB in a single appliance) and specialized hardware for examining the data stored in it. The fundamental idea is that if we position the processing closer to the data and aggressively exploit the inherent parallelism that exists in the processing requirements of applications, we can achieve significant performance gains as a result. The original concept was presented in [1], and the system is now operational.

Here, we report on two particular aspects of the operational system. First, within the K•Appliance, a number of design options are available for the disk subsystem. This includes decisions as to type of drives (SCSI vs. SATA) as well as disk controllers and RAID organizations. Empirical performance results will be presented for a number of options in this area. Second, when reading multiple files of varying sizes, it is often the case that data throughput is dominated by file lookup and seek times rather than data transfer operations. We will present mechanisms by which these overheads can be somewhat effectively masked and high throughput maintained.

The performance results are presented in the context of a pair of applications. The first is a text search application, in which a corpus of unindexed files is searched for one or more keywords. The second is the familiar triple DES encryption algorithm, where plaintext is converted into cyphertext. The first application has the property that while a large volume of data must be examined, only a small fraction is actually of interest to the application. This is a classic data thinning application. The second application has the property that the data volume out of the encryption algorithm is the same as into the algorithm. Here, we are interested in write performance as well as read performance for the disk subsystem. These two applications have very similar access patterns to the larger set of applications we are pursuing. The data is primarily write once, read many times with limited alteration once written. In addition, individual executions frequently access a

significant fraction of the data store (i.e., single runs access GB to TB of data).

The paper is organized as follows. Section 2 describes the experimental platform, including the hardware architecture, storage system(s), and software organization for managing data movement out of the storage system. Section 3 presents the empirical performance results for a wide variety of configurations, and Section 4 concludes.

2. Experimental Platform

The logical organization of the K•Appliance node architecture is shown in Figure 1. Reconfigurable logic is positioned between the disk controller(s) and the processor(s), enabling computation to be performed on data within the reconfigurable logic prior to access by the general-purpose processor. Our system node is built using either a Sun K85AE motherboard (the same one used in the 2100z workstation) or a Tyan S2882 motherboard. Each is configured with a pair of AMD 2.4 GHz Opteron processors and 8 GB of memory. The reconfigurable logic is in the form of a Virtex II series field-programmable gate array (FPGA) from Xilinx mounted on an AvNet PCI-X card. The various disk subsystems are described below.

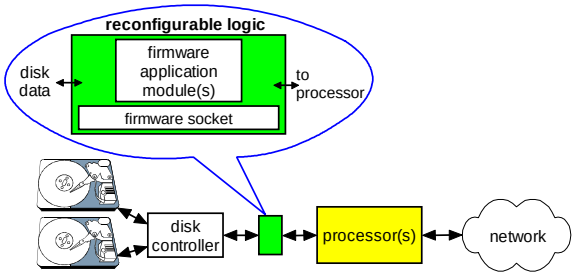

Figure 1. K•Appliance node architecture.

Within the reconfigurable logic, there is a firmware socket that manages the data flow in and out of the FPGA plus one or more firmware application modules that perform computations on the data that are flowing through the FPGA. Examples of firmware application modules that we have built include exact and approximate text search [2], biosequence search [3], signature hashing (i.e., MD5 [4]), encryption/decryption (i.e., 3DES [5]), etc.

The overall system architecture is illustrated in Figure 2. A set of appliance nodes are interconnected via a network, and the disk drives that store the data set are partitioned across the nodes. Each appliance node contains one or more traditional processors, memory, one or more FPGAs, disk controllers, and the network interface that provides connectivity to the other compute nodes.

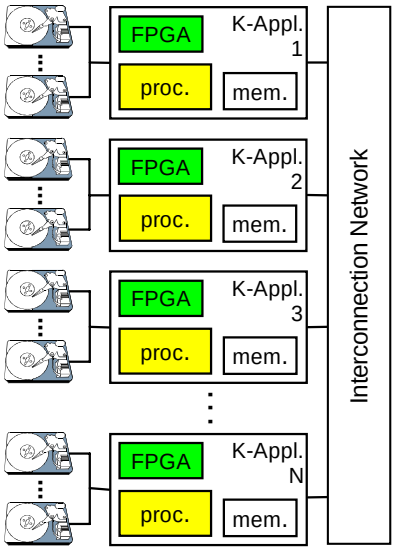

Figure 2. Parallel K•Appliance configuration.

This parallel configuration is exploiting one of the important properties of the applications of interest. The data can be partitioned across the system, and the individual compute nodes can work on their assigned subset of the data without compromising the quality of the computation. While we *are* querying very large data sets, we are *not* attempting to perform a complex join operation across a set of distributed relational database tables.

2.1. Disk Subsystems

In the next section, we will present empirical performance results for a number of potential disk subsystem configurations (on a single K•Appliance node). Here, we describe the configurations themselves. Our SCSI disk subsystem is comprised of 24 drives (Seagate

146 GB each) physically connected to 4 SCSI channels on 2 controllers. This results in only 6 drives per SCSI channel (illustrated in Figure 3). Two manufacturer's controllers have been investigated, the LSI MPT Fusion 22320-R and the Adaptec 39320A-R. The subsystem is then organized in 3 alternate configurations: the first is a single 24-drive RAID, the second is three 8-drive RAIDs, and the third is 6 4-drive RAIDs. Each RAID has an independent file system mounted on it, and the striping is across controllers, not within the drives attached to an individual controller (i.e., vertically in the figure). For example, the stripe order for the 4-drive RAIDs is: A1, B1, A2, B2. All of the SCSI RAIDs are RAID0.

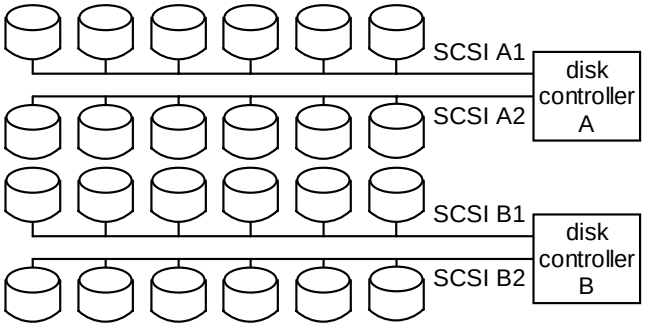

Figure 3. SCSI disk subsystem configuration.

Our SATA disk subsystem is also comprised of 24 drives (Seagate NCQ 200 GB each) physically connected to 3 controllers (8 drives per controller). Two manufacturer's controllers have been investigated, the Highpoint RocketRaid 2220 and the Broadcom 4852. The same 3 alternate configurations are used as in the SCSI system, with the added item of both RAID0 and RAID5 configurations being supported. In the SATA systems, the striping is within an individual controller, as the link between controller and each disk drive is dedicated rather than shared.

2.2. Data Movement

The software organization for reading data from the disk subsystem and delivering it to the FPGA is illustrated in Figure 4. A number, *k*, of independent threads are allocated to associated silos. The silos are FIFO buffers in system memory that are used to stage data from the disk subsystem prior to delivery to the FPGA. The number of silos, *k*, is normally equal to the number of independent RAIDs in the storage subsystem configuration.

It is important to point out that the data movement indicated in the figure is under the control of an execution thread on one of the general-purpose processors, but the actual data transfers are all DMA transfers commanded either by the disk controller (for data moving from disk to silo) or by the firmware socket within the FPGA (for data moving from silo to FPGA). Other than during transient bursts at program startup (e.g., for file lookup, etc.), the processors are generally 80% idle during most of the application execution.

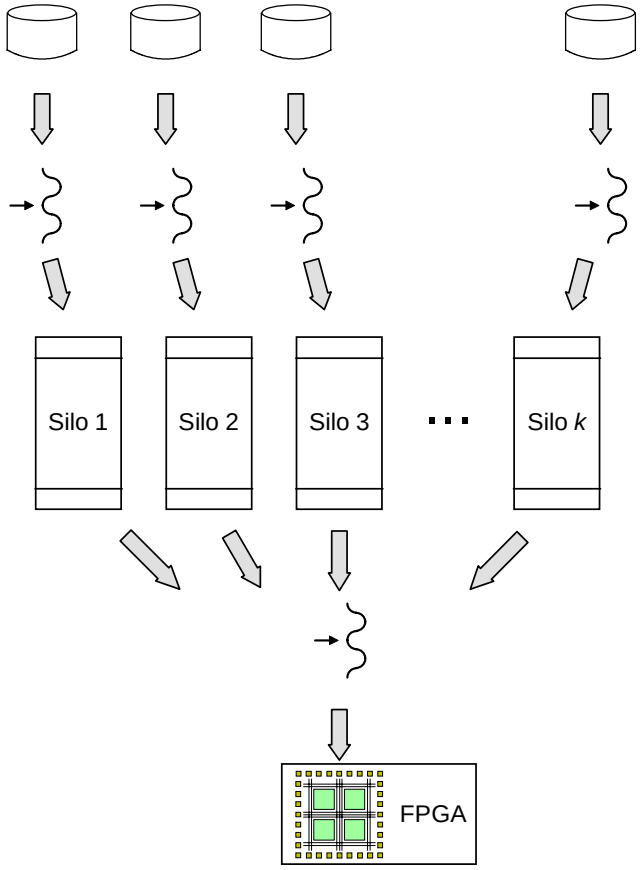

Figure 4. Read silos.

The majority of the applications we are dealing with can be characterized as write once, read many applications. Individual files are typically generated, stored, and then queried multiple times,

while updates of preexisting files are relatively infrequent. In addition, queries on the accumulated data set often are quite large, accessing a significant fraction of the entire data set. When an application is initiated, the set of files to be read are first queried for their size. Then *k* threads are invoked, each of which reads the files from an independent RAID and fills its assigned silo. The final thread reads from the silos and delivers the data to the FPGA. For applications that cannot be multitasked on the FPGA (e.g., encryption), entire files are read from a silo and sent to the FPGA intact. For applications that can be multitasked on the FPGA (e.g., text search), the read process from an individual silo does not need to terminate on a file boundary (i.e., it reads across silos). In either case, use of the independent threads for filling the silos supports greater parallelism across the disks (e.g., multiple reads of separate RAIDs). It also improves the overlap between disk reads and data delivery to the FPGA.

We explore two mechanisms for addressing the particular issues that arise with large numbers of small files. Specifically, for small files, file lookup operations and seek times can completely dominate the file access time, dramatically dropping the overall data throughput. The first mechanism attempts to mask the overheads associated with small files by processing the filesystem lookups and individual reads concurrently with the bulk of the remaining data to be accessed. This is accomplished by allocating an additional silo dedicated only to small files (i.e., those below a specified threshold in size). The original *k* silos operate as before, while silo *k*+1 reads all of the small files. Once the large files have completed processing, the collection of small files are output from silo *k*+1 and delivered to the FPGA.

If the total number of small files is sufficient, the above mechanism is unable to mask the lookup and seek times associated with small files. The second mechanism we explore is the explicit concatenation of sets of small files into a single large file for the purposes of data movement to the application. An index into the constructed concatenation is built, which allows the identification of the original file ID within the larger constructed file. This mechanism clearly is more appropriate with the write once, read many times property that our systems exhibit.

3. Empirical Performance Results

To exercise the system and collect empirical performance results, we have constructed a file set comprised from a variety of sources. The data includes an entire (abandoned) Windows XP filesystem, Shakespeare's collected works, a variety of song lyrics, the publicly available Enron emails, the full Red Hat ES 4.0 distribution (unpacked and in ISO), a body of commercial fixed-length record data, and several other miscellaneous items. The total data size comprises over 58 GB in 124,751 distinct files.

Figure 5 shows the distribution of file sizes in our experimental data set. Note that there are over 16,000 files that are empty (all from the XP filesystem) and over 72,000 non-empty files that contain less than or equal to 8 KB of data.

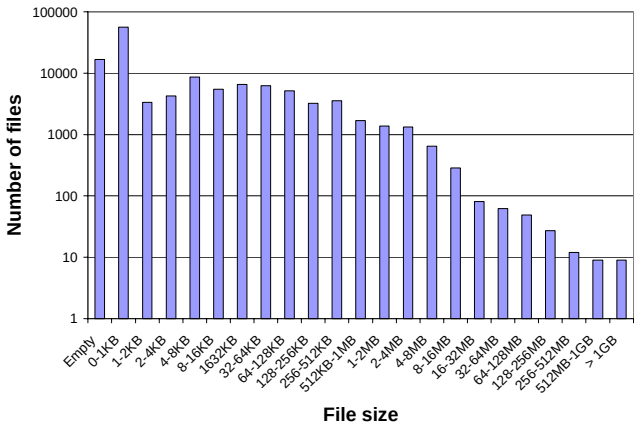

Figure 5. File size distribution.

Experiments were run across the following dimensions: SCSI RAID0 vs. SATA RAID0 vs. SATA RAID5; two manufacturer's controllers (for each of SCSI and SATA); three configurations (i.e., 1x24, 3x8, 6x4 as described above); and three applications (read, search, and encrypt). This results in a total of 54 distinct experiments. However, not every experiment is

worthy of full treatment. In the paragraphs below, we examine several general trends present in the above dimensions of the experimental system, limiting the need for exhaustive coverage.

First, across the board, SATA RAID0 and SATA RAID5 performance was virtually identical. The technique of performing RAID5 processing within the controller is an unqualified success. While the literal numbers presented below are from the SATA RAID0 executions, RAID5 performance did not vary in any significant way.

Second, the two SCSI controllers (one from LSI and one from Adaptec) were also indistinguishable, performance wise. The reported results come from the LSI runs. For the SATA disk systems, there was a performance issue with the Broadcom controller that inadvertently limited its performance. When more than one controller is present in the system, the driver utilizes a single kernel lock that is common across the set of controllers. As a result, the performance doesn't effectively scale up with the number of controllers. This artifact is a limitation of the current driver, not the hardware, and we have communicated our findings to Broadcom. So as to not present an unfairly biased comparison, we will only report results from the Highpoint SATA controller.

Third, of the 3 disk configurations (1x24, 3x8, and 6x4), the 1x24 configuration was wholly unsuited to the task at hand. Read throughputs were consistently less than 200 MB/s, limited by the constrained parallelism of the configuration. The detailed reported results are for the 3x8 and 6x4 configurations.

Table 1 shows the measured I/O throughput for reading the entire data set of Figure 5. Note that these numbers do not include the time required to perform file lookups (generally about 10% of the time for data transfer in this data set). It is envisioned that in a production environment the lookup for one query can be pipelined with the data transfer for the previous query, thereby masking this latency. In addition, the use of cached directory entries dropped this overhead by a factor of 5, further minimizing its performance impact.

Two trends are immediately apparent that persist in subsequent experiments. First, with a common configuration, the SCSI systems consistently outperform the SATA systems. Second, the 6x4 configurations consistently outperform the 3x8 configurations. The increased parallelism is being exploited for performance gain.

Table 1. Read throughput.

Disk Technology	Configuration	I/O Throughput
SCSI	3x8	224 MB/s
SCSI	6x4	360 MB/s
SATA	3x8	217 MB/s
SATA	6x4	318 MB/s

The next set of reported results is from the search application. The entire data set is searched for a pair of keywords (ensuring that the data volume out of the search application is negligible). Here, we are interested in the rate at which data can be read from the disk and delivered to the application.

Figure 6 plots the throughput from disk to application for 16 different experiments. The disk technology is either SCSI or SATA, the configuration is either 3x8 or 6x4, and the size threshold refers to the second mechanism described in Section 2.2 for mitigating the impact of small files (concatenating files that are smaller than a given size threshold).

The first observation is that the trends described above with respect to read performance continue to be exhibited in the search application. Generally, the SCSI system outperforms the SATA system and the 6x4 configuration outperforms the 3x8 configuration. A second observation is that delivering the data to the FPGA does not appreciably impact the achievable throughput. The performance results indicated in Figure 6 are roughly equivalent to the performance presented in Table 1. A third observation is that, in most cases, there is direct performance benefit to be had by aggregating

small files and processing them all together, rather than handling each file independently.

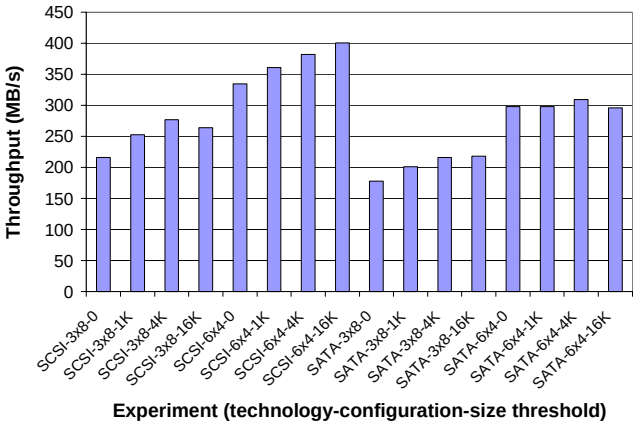

Figure 6. Search throughput.

The other mechanism described for mitigating the impact of small file sizes was to add an additional silo dedicated solely to small files. Performing this optimization on experiment SATA-6x4-0 moves throughput up to 353 MB/s, a significant performance gain.

While the experimental data set was chosen to be quite general, our applications of interest typically do not contain anywhere near the quantity of small files present above. To further explore the performance impact of these small files, we executed the search application only on files of a given minimum size. The results of these experiments are presented in Figure 7. Here, the performance jumps appreciably as soon as a 1 MB minimum file size is present.

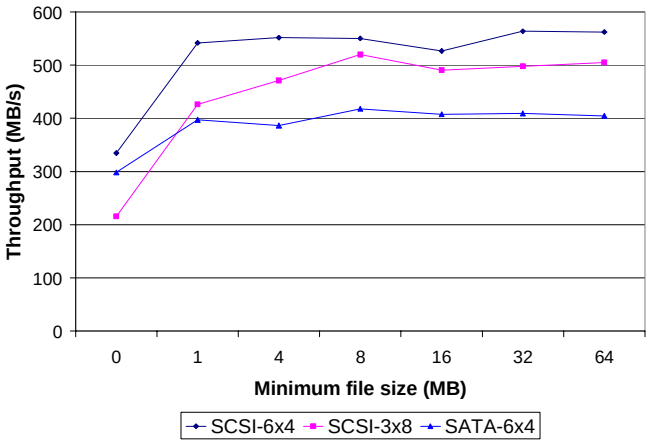

Figure 7. Impact of minimum file size.

At the top end, our SCSI system is capable of reading at 729 MB/s, given a small number of large (GB) files to process.

The 3DES encryption application has the property that the output data volume is equal to the input data volume. Here, we are not just interested in read performance from the disk subsystem, but in write performance. For the experimental data set of Figure 5, the large number of small files is a particularly difficult task, and none of the disk systems performs well on it. When given a workload more suited to its capabilities, however, the 3DES application can perform well. We have achieved 300 MB/s end-to-end throughput when performing encryption operations on large files. This includes reading the plaintext from disk, encrypting it into cyphertext, and writing the cyphertext to disk.

4. Conclusions

This paper has introduced the K•Appliance architecture and described empirical performance experiments for a number of disk subsystem design options. Design dimensions that were investigated include disk technology (SCSI RAID0, SATA RAID0, and SATA RAID5), disk controller, and RAID configuration (1x24, 3x8, and 6x4). The applications utilized include read, search, and encrypt).

A number of performance trends are noted. First, with the same number of physical drives, the SCSI system consistently outperforms the SATA system. We have achieved SATA performance that matches the best SCSI performance reported here, but it requires a larger number of SATA drives to reach that performance level.

Second, the larger number of independent RAIDs available in the 6x4 configurations consistently performed best. This can be attributed to the increased parallelism available due to the additional silos and the parallelizable nature of the data access in the applications.

Third, the performance of the search application (which primarily consumes data and generates few results) closely matches that of the read

application. We are consistently delivering data from the disk store all the way to the application without degrading performance.

Finally, the size of the files in the data set (especially small files) has a dramatic impact on achievable performance. Two mechanisms were proposed to mitigate this impact, and both had a positive influence, but maximum throughput for the disk system (both read and write) is achieved when file sizes are large.

An additional technique for minimizing the seek times associated with reading large data sets is to arrange the data so that within files the information is laid out contiguously. If one has access to application-specific knowledge, a directed layout of data to the disk system is possible. Without such knowledge, it is still possible to diminish the fragmentation of data through the use of the filesystem described in [2]. We are currently building a prototype of this filesystem for use with the K•Appliance.

5. Acknowledgments

The authors would like to acknowledge the financial support of Exegy Inc. and National Science Foundation grant number CCF-0427794.

6. References

[1] Roger D. Chamberlain, Ron K. Cytron, Mark A. Franklin, and Ronald S. Indeck, "The *Mercury* system: Exploiting truly fast hardware for data search," in *Proc. of Int'l Workshop on Storage Network Architecture and Parallel I/Os*, September 2003, pp. 65-72.

[2] Roger Chamberlain and Ron K. Cytron, "Novel Techniques for Processing Unstructured Data Sets," in *Proc. of IEEE Aerospace Conference*, March 2005.

[3] Praveen Krishnamurthy, Jeremy Buhler, Roger Chamberlain, Mark Franklin, Kwame Gyang, and Joseph Lancaster, "Biosequence Similarity Search on the Mercury System," in *Proc. of the IEEE 15th Int'l Conf. on Application-Specific Systems, Architectures and Processors*, September 2004, pp. 365-375.

[4] Ronald L. Rivest, *The MD5 Message Digest Algorithm*, Internet RFC 1321, April 1992.

[5] ANSI, "Triple Data Encryption Algorithm Modes of Operation," American National Standards Institute X9.52-1998, July 1998.