

Terascale I/O Solutions

Nathan Stone, John Kochmar, Paul Nowoczynski, J. Ray Scott, Derek Simmel,
Jason Sommerfield, Chad Vizino

Pittsburgh Supercomputing Center, 4400 Fifth Avenue, Pittsburgh, PA 15213, USA
{stone, kochmar, pauln, scott, dsimmel, jasons,
vizino}@psc.edu

Abstract. PSC has architected and delivered the TCS-1 machine, a Terascale Computing System for use in unclassified research. PSC has enhanced the effective usability and utilization of this resource by providing custom I/O solutions in four key areas: high-performance communication, high-performance file migration, checkpoint/recovery and an updated hierarchical storage management system. These I/O solutions have a synergistic effect that is leveraged in their design, implementation and integration. Each successive enhancement builds on its predecessors, thereby exacting the highest performance (*e.g.* multi GB/sec file transfers) from the available hardware. This paper presents a technical overview of these solutions from design to integration to application.

1. Introduction

We have four well-developed and unique custom I/O solutions to suit the needs of TCS-1 users: high-performance communication, high-performance file migration, checkpoint/recovery and a new hierarchical storage manager. These solutions, although they address independent issues, have been integrated in ways that can and will allow TCS-1 users an unprecedented quality of service in these areas. The impact of these solutions enhances the performance both of the TCS-1 machine and of the applications that run there.

1.1. Machine Configuration

The TCS-1 machine is a cluster of more than 750 Compaq ES-45 quad-processor servers, which are built with 1.0 GHz EV68 generation Alpha processors. Thus there is a pool of more than 3000 processors available to the compute partition of the machine, which does not include “hot” spare nodes positioned for immediate and automated scheduling in the case of a compute node failure. The nodes are interconnected via two independent communication fabrics (or “rails”) called

“QsNet”,¹ by Quadrics. The TCS-1 is a Compaq AlphaServer SC² system. As such its software configuration is composed from three layers. The operating system on each node is Tru64. Nodes are grouped into sets of 32 by a cluster-management layer called TruCluster, which creates clusters from Tru64 nodes. In addition to providing high-availability of services that run within the TruClusters, this layer provides some advanced file system features. The third layer is called the Resource Management System (RMS). RMS provides for some level of hardware event handling, a basic job queueing interface, a parallel job launching mechanism and the ability to divide the available nodes into “partitions” for convenience. Many of our high-level site customizations are layered over RMS, from custom scheduling and job control to event monitoring and management, although some site customizations necessarily intercede at a lower level.

2. TCSCOMM

The first feature we present is a high-performance communication library called TCSCOMM. This is a user-level library that allows applications to use the QsNet to communicate outside of the RMS environment. By using TCSCOMM, system level daemons and user-level applications can transmit data at near Quadrics network rated speeds, with extremely low latencies.

QsNet is a high bandwidth, low latency network comprised of 2 major components: the Elan network cards and the Elite switch components. A full “fat-tree” switch topology allows point-to-point non-contending communication between any two nodes at full network bandwidth. On TCS-1 full bandwidth is approximately 250MB/s per QsNet rail with a 5 microsecond latency.

2.1. The Elan System Libraries

Two system libraries are available for communicating over QsNet using the Elan cards: the libelan³ library, and the libelan3 library. The libelan library is a high-level library that facilitates communications between processes locally and on other machines connected to the switch. This library requires RMS to set the memory layout, communications contexts and capabilities. This library is designed to be less architecture dependent, so that applications and libraries written against this library can be more easily ported to future versions of the QsNet interconnect.

The libelan3 library, a lower level interface, is specific to the elan3 card and interconnect and much of the libelan library is based on it. With this interface, applications can control the specific memory layout for communications, as well as create specific context and capabilities for the communications. In addition

¹ See: <http://www.quadrics.com/>.

² See: http://www.hp.com/techservers/systems/sys_sc.html.

³ Quadrics Supercomputers World Ltd. Elan Programming Manual, January 1999.

applications can access the lowest levels of the communication layer, providing the highest levels of performance at the expense of code complexity and portability to future versions of the interconnect.

2.2. A Custom Elan3-based API

It was our intent to use the QsNet for system applications and daemons (running outside of RMS), so we designed the TCSCOMM API to use the `libelan3` library rather than the `libelan` library. Using the `libelan3` library had the following benefits:

- It allowed us to provide communications between Tru64 and Linux (IA-32 and IA-64) hosts on the same QsNet network by ensuring the same memory layouts on the two architectures. By default, the `libelan` library chooses a layout that matches the process memory layout, which differed between the various architectures and OSes.
- It allowed us to create custom capabilities and contexts for the running applications. This allowed us to run outside of RMS, so that we could create system daemons and applications that didn't need to depend on RMS to run.
- Using `libelan3` also allowed us to develop a multi-rail communications scheme that we use to improve performance by striping communications over multiple independent QsNet rails, improving over-all performance.

At the moment, the library only supports point-to-point communications. There is no concept in this library of a multi-cast operation, even though the QsNet supports this type of operation in the form of a hardware broadcast.

2.3. Memory Management

Mapping the memory directly from the Elan card allows for the greatest performance, over 220 MB/s on a single rail, and better than 400 MB/s over 2 rails. This contrasts with 170MB/s and 270MB/s when not using the API's memory malloc routines, respectively. The maximum memory available to use off of the card is 32 MB. This size is configurable at library compile time, and in addition to the 32MB available for `tcscomm_malloc`, 16MB is used for send/receive buffers for the `tcscomm_send` and `tcscomm_recv` calls when using memory not allocated via `tcscomm_malloc`.

2.4. Message transfer protocol

The TCSCOMM library uses a two-stage method to transfer data from one client to another:

- Queued DMA operations are used to send data transfer requests from the sender to the receiver.

- The receiver uses the `elan3_getdma` function to transfer large blocks of data from the sender to the receiver, based on a request pulled from the message queue.

When one process wants to send data to another process, it stores the data in an area of memory known to the library, and then tells the remote process to fetch the data, allowing the receiver to synchronize the data transfers.

3. TCSIO

The second feature we present is a high-performance file migration utility called TCSIO. This is an object-oriented client-server toolkit. As such, we introduce the presence of an “I/O daemon” that runs on each node where disk resources are to be presented—both file servers and compute nodes, for the purpose of accessing their local disks. The I/O daemons are conversant in multiple transport protocols, including a reference implementation in TCP/IP and an expansion to the TCSCOMM library. The protocol for connecting to an I/O daemon begins with a TCP/IP socket connection and includes, among other things, a negotiation to the highest performance protocol for bulk data transport. Metadata are always transmitted over the TCP/IP connection. By separating functionality between the client and the I/O daemon we achieve several useful benefits, as follows.

3.1. High-Performance

All of the performance-oriented software is isolated within the daemons, minimizing exposure to the user’s environment. This client-server subsystem has been shown to achieve transfers at roughly 2 GB/sec aggregate, utilizing parallelism for the transfer of multiple files. Furthermore, distributed clients transferring to `/dev/null` on the file servers have been benchmarked at roughly 2.4 GB/sec, revealing that faster file systems on the file servers would improve the overall transfer performance. This is an area of ongoing work.

3.2. Bypass NFS Cross-Mounts

The presence of an I/O daemon at the point of origin of disk resources allows us to explicitly transfer files from source to destination via our custom TCSIO protocol. This has the immediate advantage that we can replace NFS cross-mounts with our higher-performance protocol, which is faster than NFS due to the use of TCSCOMM. Furthermore, in a system the size of TCS-1, the high node-count can lead to literally hundreds of cross-mounted file systems. This presents a scaling problem that can significantly degrade the performance of conventional files systems such as NFS.

3.3. Third-Party Copy

By passing a copy *request* from a client to the I/O daemon the client can leverage the I/O capabilities of a reliable third-party process. Thus a compute job can, for example, pass a non-blocking file migration request to an I/O daemon and return to CPU-intensive compute tasks while the I/O daemon (a separate process) handles the I/O intensive tasks involved with the migration.

3.4. Pluggable Transport Protocols

Communications in the I/O daemon are encapsulated in a “Connection” object. In this way, future transport protocols can be added with minimal code modification, in a manner completely transparent to user applications running on the system.

3.5. I/O Redirection

Some applications of TCSCOMM are being introduced directly into users’ applications. One of these instances will use the TCSCOMM library directly, not to migrate an existing file, but to redirect file-oriented I/O to a remote file server. This will be further described in the Checkpoint/Recovery section below.

3.6. Extensible Service Integration

The I/O daemon has already been instrumented with additional features, beyond file migration, for other custom TCS-1 services. Thus, the I/O daemon provides a foundation for building other integrated services, like checkpoint/recovery and our new HSM implementation, which we discuss below.

4. Checkpoint/Recovery (CPR)

We have created an application-level checkpoint/recovery (CPR) library to enable TCS-1 users to checkpoint and recover jobs in a way that makes optimal use of machine resources and provides additional features otherwise inaccessible to them. Of dominant concern in the design of this library was the minimization of “non-compute” time—time spent either in I/O or library overhead.

Some features of the CPR library are discussed in a previous publication,⁴ including the description of independent I/O schemata (also called “plans”) and how a user’s job selects the desired I/O schema for a particular job execution by the means of an

⁴ N. Stone, J.R. Scott, J. Kochmar, J. Sommerfield, R. Subramanya, R. Reddy, K. Vargo, “Mass Storage on the Terascale Computing System”, Proceedings of the 18th IEEE Symposium on Mass Storage Systems and Technologies, p. 67, 2001.

environment variable. Of note is the fact that each I/O plan is a complete implementation for generating and retrieving checkpoint data. As such, any of the available plans could be selected for any job execution, although certain job types may experience performance differences between the various I/O schemata. Furthermore, since all plans are accessed via the same fixed API, the CPR library is extensible to support new schema implementations. To access a newly added schema, users need only re-link their application to the updated CPR library and update the corresponding CPR plan environment variable.

The checkpoint implementation also supports a passive pre-emption mechanism. It provides users with a means for checking whether their job should write a checkpoint in anticipation of an impending job termination.

The recovery implementation is also dependent on the selected checkpoint schema. All plans rely upon some type of redundancy in storing checkpoint files, as a means of protecting against lost of files, storage media or file-serving nodes. Recovery plans range from retrieving full duplicates of lost files to regenerating lost files from parity files. In all cases the recovery is handled by a single call to the API.

4.1. Failover Syntax

When identifying files internally, the checkpoint system utilizes a failover syntax that expresses potentially redundant routes to a given file path. Failover paths are generally strings that include host and path notations with reserved keywords. Here are a few illustrative examples:

“node[4-10]:/scratch/” indicates the absolute path `“/scratch/”` that exists only on the hosts “node4, node5, ... node10” and is only accessible locally on those nodes;

“{local}:/local/checkpoint/” indicates the absolute path `“/local/checkpoint/”` that exists on all nodes but is exclusively accessible on the node where it is written;

“{cluster}:/usr/storage/” indicates the absolute path `“/usr/storage/”` that is mounted from the cluster file system and is thus accessible equally from all nodes within a given TruCluster set;

“node[0-9](n+1):/usr/speedy/” indicates the absolute path `“/usr/speedy/”` that exists on hosts “node0, node1, ... node9” and are pair-wise redundantly accessible (*e.g.* via multi-initiator SCSI), for example node0 and node1 can access each other’s `/usr/speedy`, and similarly for node2 and node3, node4 and node5, and so on.

Using this failover notation during the creation and registration of checkpoint files allows the propagation of this redundancy information to the recovery step, thereby providing alternative access routes to checkpoint data in the case of hardware or other failures.

Performance of the checkpoint I/O methods is internally monitored and reported to an external database for offline analysis and evaluation. In the event that this reporting

becomes burdensome either to the user's job execution or the system's recording resources this behavior can be deactivated by a switch in a system configuration file `/etc/tcsiiod.conf`.

4.2. Scheduler Integration

Checkpoint/recovery is of some usefulness without automation, but its full value is only achieved when it is integrated into the scheduling environment for automated restart and recovery. Our CPR system has been integrated into the Resource Management System (RMS) to automatically flag machine-failure conditions as distinct from job failure conditions. It has also been integrated into "Simon," our custom scheduler based upon OpenPBS.⁵ PBS, like most schedulers, provides hooks for scripts that can be run immediately before and after job execution; for PBS these are called the "prologue" and "epilogue" scripts. By providing a supplement to the epilogue script we ensure proper cleanup of CPR resources, re-running of failed jobs that will need to resume from checkpoint and notification of administrators and the user of noteworthy job-related conditions. The epilogue script even automates refunding of "lost" compute time, which we discuss further below. The precise steps executed within that script depend on both the job exit status, captured by RMS, and the state information, if any, held in the checkpoint system. In this way jobs that abnormally terminate because of machine failure conditions will be automatically recovered without intervention from the administrator or the user.

4.3. Accounting Integration

Many computing sites monitor their usage, charging users or grants directly for resources consumed by compute jobs. The accuracy of the records is imperative. Since CPR handling occurs within a user's job, CPR handling can cause extensions to the wall-time consumed by a user's job and thus, an inflated job charge. The integration of the CPR system and the scheduling system is made complete by a further integration of the accounting system. To eliminate charging the user for CPR handling, the CPR system keeps track of the time spent in CPR activities and reports this in a table that is available to our accounting system. Furthermore, by measuring the time between the last complete checkpoint and the end of a failed job one can determine the amount of time "lost" by a compute job that terminated due to machine failure and post that in a manner accessible to the accounting system as well. In our CPR system the PBS epilogue script records this automatically, as noted above. In this way, resources lost or consumed by CPR activities can be credited back to the user's allocation.

By policy, neither nodes nor processors within TCS-1 are time-shared. Thus, the formula for job charging on TCS-1 is as follows:

⁵ See: <http://www.openpbs.com/>.

$$C = \sum_{i=1}^{N_r} (UT_i - RT_i - \min(LT_i, MaxLT)) * N$$

where:

- C = total charge for user's job (node-hrs)
- N_r = total number of RMS resources created by OpenPBS job
- UT_i = wall-clock time (hrs) for each RMS resource
- RT_i = time spent in checkpoint file recovery for each RMS resource
- LT_i = "lost time" calculated for each RMS resource
- $MaxLT$ = maximum refundable "lost time", set by policy
- N = number of nodes requested

4.4. Examples

Encouraging users to make use of application-level checkpointing is, to a large degree, a matter of user education and is motivated by the direct advantages to the user. Aside from the obvious benefit of job recovery, users gain recovery automation, an assurance of checkpoint file availability, and access to the highest-performance storage resources on the TCS-1 by using our CPR library. As a first step toward achieving this user education we provide many examples of CPR-instrumented codes ranging from simple two-integer test cases to Laplace solvers, in both C and Fortran. In this way we hope to make it easier for users to understand how to properly utilize the library and its features.

5. Hierarchical Storage Manager

We have designed a new archiving system that will tightly integrate into TCS-1. The Scalable Lightweight Archival Storage Hierarchy (SLASH) is a heterogenous system comprised of two major components. The first component is the Linux Cache Node (LCN) cluster. LCNs are arranged as a loosely distributed caching cluster that is enmeshed in the QsNet network. The second component is XFS, a metadata file system that distributes and maintains the metadata for the distributed disk caches. The third component is DMF, a tape archiver that manages the tape archives and their front-end disk cache. XFS and DMF are tightly integrated, as discussed below.

5.1. Linux Cache Nodes (LCNs)

The Linux cache nodes (LCNs) have been designed to handle considerable amounts of I/O and house many disks. LCNs have the internal capacity to hold up to 32 commodity IDE devices. Given today's maximum disk density a single LCN's usable storage capacity is over 8 terabytes.

The software layer, SLASH-SWL, executes a number of tasks such as getting

permission from XFS for file system update operations and obtaining LCN residency information for specific file. SLASH-SWL also manages the local cache consistency, which offloads tasks from XFS.

SLASH-SWL exports an API that most file transport applications, *e.g.* the TCS I/O daemons, can use. When uploading data to the HSM, an I/O daemon running on an LCN obtains the target upload file descriptor from a SLASH-SWL API function (“hsmCacheUploadInit”) instead of directly from an `open()` system function. For HSM downloads another SLASH-SWL API call (“hsmCacheDownloadInit”) is used, which can either return a local read-only file descriptor or, if the local LCN does not have the most recent cached file, redirect I/O to another I/O daemon on the LCN holding the most recent version.

5.2. XFS/DMF

The XFS⁶ subsystem is an advanced Unix file system that incorporates capability for user-supplied information into its metadata. It runs on Silicon Graphics, Inc. (SGI) hardware as well as most Linux platforms. The Data Migration Facility (DMF)⁷ is an additional software package, also from SGI, that integrated HSM functionality into the XFS file system. DMF is currently most mature on the SGI hardware platform, though Linux options are emerging. The current architecture has a moderately sized and robust SGI server connected running the XFS/DMF file system. The design includes over ten terabytes of disk cache presented by several external RAID enclosures and petabytes of tape storage connected to the SGI machine. Failover paths to these storage devices are possible through a Fibre Channel switch.

SLASH uses the “user metadata location” of the XFS metadata to store its own internal consistency information. Noted information stored here is the identifier of the LCN, if any, which holds the most recent version of the file. SLASH requests from the LCNs are transmitted via an RPC layer to the XFS file system.

5.3. User Access Methods

The preferred access method for SLASH is the TCSIO subsystem. Used within the Terascale Computing system, TCSIO to SLASH will perform data transfers via the Quadrics network. When transferring groups of files, such as in the case of reading or writing checkpoint files, TCSIO can take advantage of SLASH's large distributed cache by batching many simultaneous requests to the LCNs. This method will yield bandwidths of order gigabytes per second.

A secure, interactive client, similar to 'ftp', will also be provided. This will be accessible via the wide or local area network. The interactive client will be built

⁶ See <http://oss.sgi.com/projects/xfs/>.

⁷ See <http://www.sgi.com/products/storage/software.html#dmf>.

with the SLASH-SWL API so that it will be able to take advantage of the large distributed cache.

6. Total Integration

We have presented a discussion of application services designed to augment both the performance of the TCS-1 machine and the applications that run there. Our innovations range from communications to file systems to job checkpoint/recovery to hierarchical storage, yet all of these areas have been integrated to extract the highest possible performance from this machine. The I/O enhancements described above are not mere research constructs. These are deeply integrated into one another and into the production services now used by many TCS-1 users. In addition, some of these services have been designed and written in a machine-independent or even platform-independent manner to facilitate portability of these facilities to additional platforms at PSC and elsewhere in the scientific computing community.