Efficient Parallel I/O on SCI Connected Clusters

Joachim Worringen

Abstract- **This paper presents a new approach towards parallel I/O for message-passing (MPI) applications on clusters built with commodity hardware and an SCI interconnect: instead of using the classic scheme of clients and a number of servers communicating via TCP/IP, a pure peer-to-peer communication topology based on efficient use of the underlying SCI interconnect is presented. Every process of the MPI application is client as well as server for I/O operations. This allows for a maximum of locality in file access, while the accesses to remote portions of the distributed file are performed via distributed shared memory techniques. A server is only required to manage the initial distribution of the file fragments between the participating nodes and to provide services like external access and redundancy.**

***Keywords*- cluster, SCI, parallel IO, MPI-IO, ROMIO, DSM**

I Introduction

As a cost-effective alternative to vendor-integrated systems, more and more high-performance computing systems are built as *components-off-the-shelf* (COTS) clusters. The standard components include the hardware and also the software, with the operating system as the basic building block. Common operating systems for COTS clusters are Linux, Windows NT/2000 and Solaris.

The currently most used programming interface to create portable parallel scientific or technical applications is MPI. Many implementations of MPI exist for COTS clusters, too. However, this kind of applications does not only require communication between the processes, but often also has to perform a significant amount of file I/O for various purposes [1].

The standard I/O interfaces offered by operating systems used in COTS clusters lack support of high-performance, parallel I/O services as they are required for typical I/O related load of scientific applications [2]. On the other hand, existing mechanisms for parallel I/O like dedicated parallel file systems have to be used via a proprietary interface which hinders portability. To overcome these problems at least for the scope of MPI programming, the MPI-2 standard includes a definition of a programming interface for parallel I/O called MPI-IO [3] to allow portable and efficient programming of MPI applications with I/O requirements. A number of MPI-IO implementations are available for COTS clusters, too.

To improve the performance of COTS clusters, high-speed interconnects like Myrinet or SCI are used instead of the ethernet-based TCP/IP network. However, the direct and thus efficient support of these interconnects for I/O is not very common. In this paper, we present the design and an implementation of a system for parallel I/O with an MPI-IO API on SCI connected clusters which directly utilizes the fast SCI interconnect.We have named this system *pioSC*^{3} which stands for *parallel I/O on SCI connected COTS clusters*. The next chapter describes the basic priniples of *pioSC*^{3}, while chapter III summarizes the results and gives some directions for future development.

Joachim Worringen is with the Lehrstuhl für Betriebssysteme, RWTH Aachen, Kopernikusstr. 16, D-52056 Aachen, Germany.
E-mail: contact@lfbs.rwth-aachen.de, WWW: http://www.lfbs.rwth-aachen.de .

II Principles of pioSC^3

Our current target is to design and implement a system for parallel I/O via SCI which will be used by MPI applications via the MPI-IO interface: a set of processes running on multiple nodes of a cluster is accessing a file for read or write operations via appropriate MPI-IO calls. This means that we can implement our design as a run-time library which operates in user-space. A system for parallel I/O which is designed to be used with MPI-IO is of course less general than a generic parallel file system. But it has the advantage of having semantic knowledge of how the data is structured (via the MPI type definitions) and allows collective access to a file and can thus be optimized towards its specific use.

The basic design idea of pioSC^3 is illustrated in figure 1. Each process has a global view on the whole file. This view is provided via the SCI shared memory, which hides the physical location of the underlying file fragments. Accesses to the file by the MPI-IO library are executed as memory accesses to the address regions in which the differerent file fragments have been mapped.

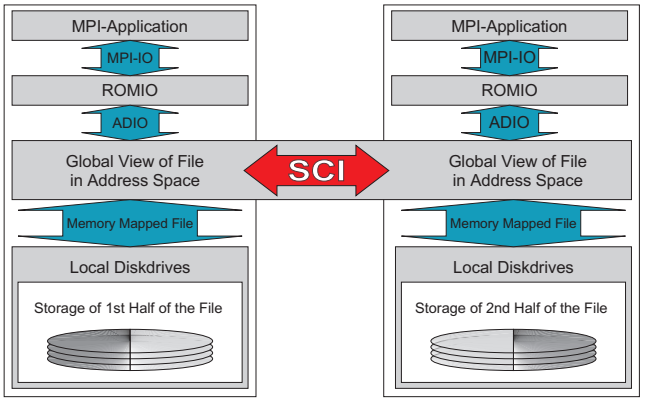

Fig. 1. Concept of MPI-IO via memory-mapped files and SCI

However, it is not possible to directly map the file into the SCI address space. Additionally, it is important to cache portions of the file which have been accessed, and to convert remote read accesses into remote write operations (writing to remote memory is an order of magnitude faster than reading from remote memory). This means that an additional software layer is required which

- uses SCI for communication without the need to map the whole file into SCI memory segments.
- converts remote-read accesses to remote-write accesses in the opposite direction.
- caches the remote portions of the file which have been accessed with a suitable consistency scheme.
- optimize locality of file accesses to reduce the amount of communication for remote accesses.

For this purpose, we use a slightly modified version of the SVMlib [4] which provides distributed shared memory by software means and is able to perform inter-process communication

via SCI. SVMlib offers different consistency models; for the given purpose, the *multiple reader - single writer* consistency model is the best choice. This results in a setup as illustrated in figure 2, again for the case of two proceses on distinct nodes accessing one shared file which is distributed over both nodes.

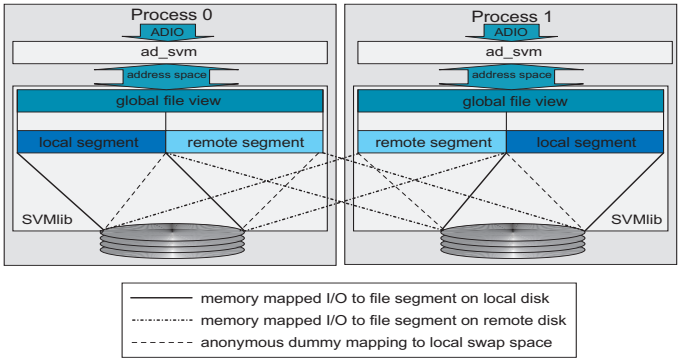

Fig. 2. Two processes map a distributed file

Next to the techniques to access a distributed file, an infrastructure to manage the distribution and collection of the file fragments is required. The approach that we have designed is illustrated in figure 3. Each node on which an MPI application process is running and performing I/O is a *peer*, since it is acting as a server and a client at the same time. Next to these peers, one *server* does exist. If a the I/O demon running on a peer needs a file fragment which is not locally available, it asks the server for this fragment which serves it itself or issues the transfer request to another *subserver* (not shown). This means the servers and subservers maintain a repository of all files in the system, while the peers may maintain redundant copies of the file fragments. The management of the files is done via a simple database on the server. If a process from outside the cluster (*client*) wants to access a file, it does this via the server which provides it via the standard TCP/IP connection.

III Summary & Conclusions

First measurements with our current pioSC^3 prototype show that the performance is vastly superior to any NFS server which is still the most common solution for I/O in COTS clusters. The SCI interconnect contributes in two ways to this performance: the communication for file access is performed via SCI, but also the inter-process communication of the MPI-IO library which is done via MPI function calls benefits from the high performance of the underlying SCI interconnect [5].

Of course, a number of tasks remain.While the current performance is good compared to NFS, we need to compare it to other distributed file systems for COTS clusters like PVFS. We also have located performance bottlenecks in the SVMlib; it might be substituted by a solution which has less overhead by reducing the number of context switches between user and kernel space and by maintaining the information for the distribution of the file in SCI shared memory.

References

[1] R. Oldfield and D.Kotz: *Applications of Parallel I/O*, Department of Computer Science, Dartmouth College, Hanover, Technical Report PCS-TR98-337, August 1998. http://www.cs.dartmouth.edu/pario

[2] R. Thakur, E. Lusk and W. Gropp: *I/O in Parallel Applications: The Weakest Link*, in International Journal of High Performance Computing Applications, (12)4:389-395, Sage Publications

[3] MPI Forum: *MPI-2 Standard*, chapter MPI-IO
http://www.mpi-forum.org/docs/docs.html

[4] K. Scholtyssik, M. Dormanns: *Simplifying the use of SCI shared memory by using software SVM techniques.* In Proc. 2nd Workshop Cluster Computing, Karlsruhe, March 1999.
http://www.lfbs.rwth-aachen.de/users/karsten/projects/SVMlib

[5] J.Worringen and Th.Bemmerl: *MPICH for SCI-Connected Clusters.* In Proc. SCI-Europe 1999, pp. 3-11, Toulouse, 1999
http://wwwbode.in.tum.de/events/sci-europe99/proceedings
http://www.lfbs.rwth-aachen.de/users/joachim/SCI-MPICH

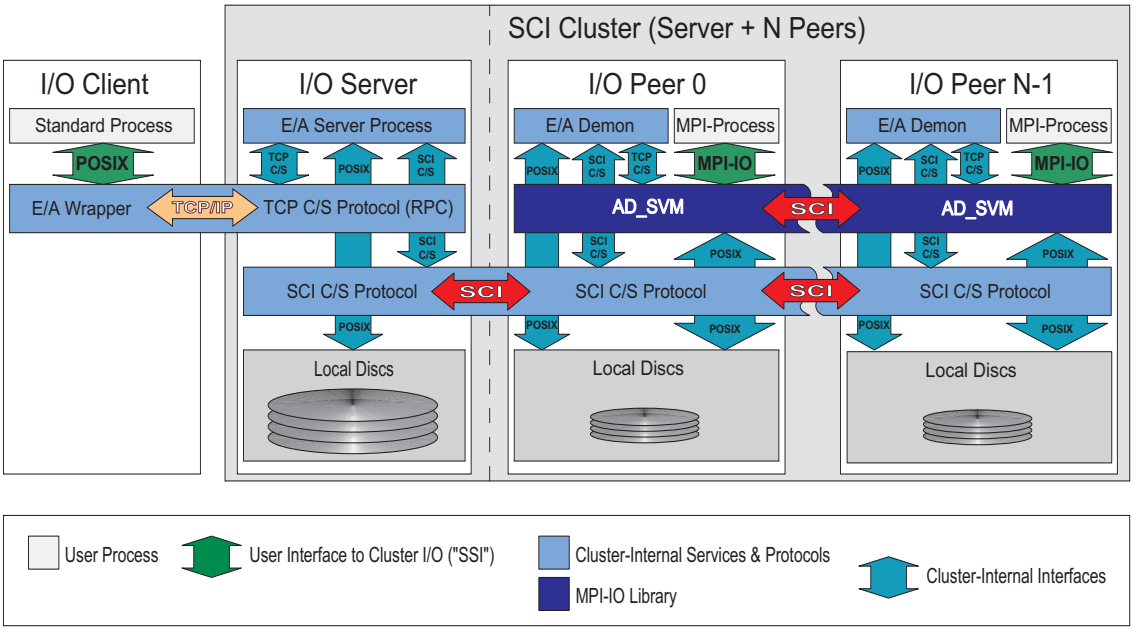

Fig. 3. Interaction of the components of the pioSC^3 system