ACTIVE I/O STREAMS FOR HETEROGENEOUS HIGH PERFORMANCE COMPUTING

FABIÁN E. BUSTAMANTE AND KARSTEN SCHWAN
College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA
E-mail: {fabianb, schwan}@cc.gatech.edu

We are concerned with the attainment of high performance in I/O on distributed, heterogeneous hardware. Our approach is to combine a program's data retrieval and storage actions with operations executed on the resulting *active I/O streams*. Performance improvements are attained by exploitation of information about these operations and by runtime changes to their behavior and placement. In this fashion, active I/O can adjust to static system properties derived from the heterogeneous nature of resources and can respond to dynamic changes in system's conditions, while reducing the total bandwidth needs and/or the end-to-end latencies of I/O actions.

1 Introduction

The high performance computing community has identified I/O as a key limiting factor in the performance of future parallel and distributed systems. This *I/O bottleneck* arises from several trends in both technology and applications. For instance, there is an increasing speed mismatch between processing units and storage devices, which is only exacerbated by the use of multiple processors operating simultaneously in parallel machines. Furthermore, new classes of applications like multimedia, collaborative visualizations of large data sets, and computational solutions to Grand Challenge problems, are imposing steadily increasing demands on I/O. Finally, the attainment of high I/O performance is additionally complicated by the heterogeneous nature of many target platforms, the dynamically varying demands on resources, and the run-time variations in resource availability. The dynamically varying demands on resources are due to applications' data dependency and/or users' dynamic behaviors, while the run-time variations in resource availability are a consequence of failures, resource additions or removals, and most importantly, contention for shared resources.

The wide recognition of this problem can be seen in the growing number of large-scale research efforts addressing high performance I/O (see [1] for an extensive list). We contribute to such research with the introduction of adaptive *active I/O streams*. Applications I/O streams are made active through the association of application-specific or system-level operators to them. These operators are logically invoked by accessing the associated streams. The resulting *active I/O streams* permit us to exploit available computational resources and whenever possible and beneficial, to move computation toward the data sources and across the I/O Bottleneck. The ability to change, at runtime, these operations' behaviors and of their placement help us deal with dynamic variations in resource demands and availabilities.

This paper introduces active I/O streams and describes the design and implementation of *Adios*, a library for active I/O targeting parallel/distributed, heterogeneous computing platforms. Experimental evaluations of the active I/O concept and its realization are performed in the context of the Distributed Laboratories (DL) project[2] at the Georgia Institute of Technology.

2 Abstractions, System Architecture, and Implementation

This section briefly defines active I/O, presents some examples of its use, and describes the abstractions supported by the Adios I/O library, its architecture, and implementation. Performance results appear in Section 3.

Active I/O. By active I/O streams we refer to application-specific or system-level functionality associated with application I/O streams. This functionality is embodied in what we term *streamlets* and is implicitly invoked by accesses to the associated streams.

Examples of useful activities associated with streams include data-based filtering, conversion of data formats, adaptive prefetching, and adaptive declustering[3].

Many high-performance applications [4,5] work with large multi-dimensional datasets representing chemical concentration in the atmosphere or astronomical readings by different instruments. These applications rarely need the entire data set but perform data-dependent filtering to extract those items they are interested on. Although doing such application-specific filtering through active I/O streams does not reduce disk I/O, it may result in substantially less network traffic.

When data is exported into files or shared between cooperating programs, formats must be chosen for the data's representation. At the lowest level, this may involve choosing appropriate machine-specific or portable binary data formats. At the application level, this involves choosing record-based data representations most likely to be efficient for the programs that share data. In either case, active I/O can help by applying streamlets that implement data transformations 'in place'.

Weissman [6] proposes a scheme for application-specific remote file access. A Smart File Object (SFO) is user-level code intended to mitigate network performance problems through application-specific prefetching that can adapt based on application and network information. SFOs can be naturally implemented as streamlets in our active I/O streams.

As a final example of the use of activity, consider file declustering in a heterogeneous environment. The appropriate distribution of file data across storage nodes can be improved by considering the heterogeneity of such nodes and of network links connecting them, as well as the dynamic variation in resource availability across the data path. Activity could be used for declustering of data based on runtime-determined weights.

Abstractions. As depicted in Figure 1, Adios models an application's I/O as a directed network comprised of *high-level streams* originating at *sources*, arriving at *sinks*, and routed through a number of *intermediate* vertices. Each stream is a sequence of self-describing application-specific data units, such as complex data structures containing chemical concentration levels in an atmospheric model. Source and sink vertices may be application programs, or they may be devices such as disks, cameras, or satellites. Streamlets can be assigned for execution to sources, sinks, and/or intermediate vertices. By acting on the data units composing the stream, streamlets may modifying the stream's characteristics.

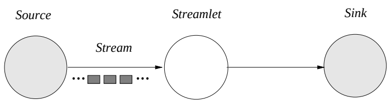

Figure 1. Streams and streamlets

Streamlets are registered with the I/O system at runtime and are subsequently attached to sets of one or more streams. The attachment of streamlets to I/O streams implicitly creates one or more additional streams (the streamlets' "output") which can then be accessed via read/write operations.

Adaptive I/O Streams. By managing I/O streams at runtime, it is also possible to cope with dynamic changes in resource availability and user needs[7]. The stream adaptations considered in our work exploit (1) the dynamic parameterization and/or specialization of streamlets that reduce/increase their execution times, while degrading/improving the fidelity or resolution of the results they produce, and (2) the migration of streamlets over the datapath in order to reduce a stream's composite bandwidth requirements or to adjust the computational/communication loads imposed by streams on the underlying hardware to match the current resource availability.

The metrics used to evaluate and drive runtime stream adaptation in real-time or media applications typically concern total throughput or end-to-end latency[7]. Our work on I/O utilizes, among other metrics, a derived characteristic of a streamlet that we have found useful for both the initial allocation and the dynamic placement of streamlets. This metric, called *sprox, benefit from proximity to source*, indicates the potential benefit of having the streamlet placed closer to the source of data. Intuitively, there is a clear benefit in placing a streamlet that filters out the incoming messages as close to the source(s) as possible (and one that expands them as close to to the sink(s) as possible) in order to improve bandwidth utilization and/or end-to-end latency. Sprox tries to quantify this benefit as a weight function of the incoming and outgoing streams' data rates.

Adios Architecture and Implementation. Active I/O is realized in the Adios system. Adios high level design is depicted in Figure 2. Adios logically consists of a Directory Service, Client-Side Servers, Storage Servers, and Intermediate Servers.

The Directory Service acts as the contact point for all Adios components and the manager of files and stream metadata for the applications. Client-Side, Storage, and Intermediate Servers bind streamlets to their corresponding streams and perform resource monitoring and management functions.

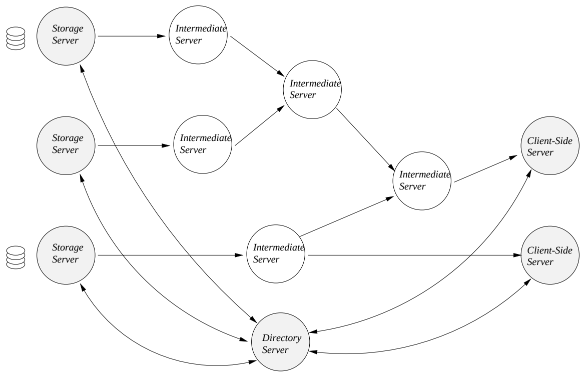

Figure 2. Adios architecture

Applications are linked with the Adios runtime libraries that translate their requests into lower level requests handled by Adios components.

Table 1. Response time to a client application's request

Streamlet location	Response time (sec.)
Part of application	62.8136
Streamlet at client	67.4798
Streamlet at server	49.4747

3 Experimental Results

The Importance of Activity in I/O. The following experiment validates the performance benefits derived from the association of activity with I/O streams and the potential advantages of adaptability. We measure the response time experienced by a visualization client requiring only a subset of the application's total data. An Adios operator performing the necessary data filtering is compared to a situation in which Adios and its filtering ability are not present.

In the experiment, a single server is connected to a single client via a 2-hop 10Mbps Ethernet link, and the client visualizes approximately 50% of the total (41.06 MB) data. The resulting performance differences are shown in Table 1.

Notice that, once the streamlet has been attached to the stream, the application can proceed as normal while the streamlet does the filtering on its behalf. This programmability improvement comes at a reduced cost, as can be seen by contrasting the first and second rows of Table 1.

Since activity is not associated with any particular device but with streams, we are able to dynamically change the placement of the filter streamlet. This application-specific filtering does not reduced disk I/O overhead but may result in substantially less network traffic. The resulting performance benefits are significant as can be seen by comparing the response times for both cases, when the streamlet is placed at the client or at server side (second and third row in Table 1). Since Adios currently uses runtime linking to migrate streamlets, however, adaptations will experience certain delays before they become effective. This delay was 20 milliseconds for the 10.7 KB streamlet used in this experiment.

Active I/O Streams for weight-based declustering. One way in which parallel I/O libraries address the I/O bottleneck is to aggregate multiple nodes with attached storage devices into one logical storage device. In our environment, the appropriate distribution of file data across such nodes must consider the heterogeneity of storage nodes and network links as well as the dynamic variation in resource availability for nodes and links. Consequently, the Adios library uses runtime-determined weights to decluster files across

Table 2. End-to-end bandwidths (in MBps) to I/O nodes

I/O Node	End-to-End		Disk		Network	
	Read	Write	Read	Write	Read	Write
lanai	0.155	0.156	1.65	3.85	0.164	0.164
micronesia	0.156	0.156	2.96	13.65	0.164	0.164
bimini	0.039	0.111	2.41	3.06	0.041	0.164
etna	0.156	0.156	1.43	2.93	0.164	0.164

Table 3. Effect of Weight-based declustering based on end-to-end bandwidths

Weight				Bandwidth (MBps)	
lanai	micronesia	bimini	etna	Read	Write
1	1	1	1	1.607	1.081
3	3	2	3	1.574	1.164
4	4	1	4	2.247	1.007

nodes; what we have termed *weight-based declustering*.

The effective bandwidths of heterogeneous storage nodes available in our lab are shown in Table 2. These bandwidths were determined by performing remote blocking reads and writes on a file in blocks of 8 KB.

From these measurements, it is clear that data must be declustered across storage engines based on the effective storage bandwidth. This is an example of useful parameterization of streamlets that route data. Comparisons of the base case of 'equal treatment' of storage engines to their unequal treatment with respect to bandwidth are depicted in Table 3.

From these experiments, it is clear that declustering files over a heterogeneous set of I/O nodes based on the nodes' performance characteristics can result in significant performance benefits. Several open issues remain concerning the appropriate assignment of weights to file components, including the fact that clients located on different nodes may experience different network connectivities to the same storage nodes. This implies that different clients sharing a single file may wish to use different weights for the file. However, the dynamic adaptability of other associated streamlets could cope with many of these situations.

4 Related Work

The potential performance benefits of moving computation across the I/O bottleneck and closer to the data has being recognized in a number of different areas including active disks, active networks, and file systems.

Active networks [8] provide a mechanism for running application code at the network routers and switches. Similarly, active disks [9,10,11] make possible to assign such functionality to empowered disk drives. In our work, activities are associated with applications' I/O data streams instead of a particular device. Once attached to their streams, the assignment of activities to specific "hosts" can be changed dynamically, adapting to the dynamic variation of resources demands and availabilities.

The Bridge file system [12] implements parallel interleaved files on the BBN Butterfly shared memory machine. One of the most innovative aspects of Bridge is its 'tool' interface, which allows application to create tool processes on the storage nodes on which the segments of a parallel file have been placed. Both Bridge and Adios allow the user to move functionality across the I/O bottleneck and towards the data. However, Adios provides an interface at a higher-level of abstraction than Bridge's tools, deals with heterogeneity, and makes use of dynamic adaptation to cope with the dynamically changing environments to which is targeted.

The dynamic placement and relocation of streamlets is a restricted type of code mobility. Code mobility raises a number of security issues and our work relies on solutions to some of those issues as proposed by projects like [13,14].

5 Conclusions and Future Work

In this paper we have introduced our ideas on *active I/O streams* and their runtime adaptation to deal with dynamically changing resource availability and with the distribution and heterogeneity of resources. Activity is supported through the association of application-specific/system-level operators with parallel/distributed I/O streams at runtime. Adaptation includes the dynamic variation of I/O streams' activities in terms of their assignments to execution sites and the precise actions performed by those computations.

We have presented our proposed programming model for active I/O streams, examples of useful activities associated with streams, and the results of our initial experimentations that validate the performance and programmability benefits of such ideas.

References

1. David Kotz. *Parallel I/O Archive*. Dartmouth College, ninth edition, February 1997. http://www.cs.dartmouth.edu/pario.
2. Beth Plale, Volker Elling, Greg Eisenhauer, Karsten Schwan, Davis King, and Vernard Martin. Realizing distributed computational laboratories.

to appear in The International Journal of Parallel and Distributed Systems and Networks.
3. Kenneth Salem and Hector Garcia-Molina. Disk striping. In *Proceedings of the IEEE 1986 Conference on Data Engineering*, pages 336–342, 1986.
4. John F. Karpovich, James C. French, and Andrew S. Grimshaw. High performance access to radion astronomy data; a case study. In *Proceedings of teh 7th International Working Conference on Scientific and Statistical Database Management*, September 1994.
5. Thomas P. Kindler, Karsten Schwan, Dilma Silva, Mary Trauner, and Fred Alyea. A parallel spectral model for atmospheric transport processes. *Concurrency: Practice and Experience*, 8(9):639–666, November 1996.
6. Jon B. Weissman. Smart file objects: A remote file access paradigm. In *IOPADS'99*, pages 89–97, Atlanta, GA, May 1999.
7. Daniela Rosu, Karsten Schwan, and Sudhakar Yalamanchili. Fara - a framework for adaptive resource allocation in complex real-time systems. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium (RTAS)*, Denver, CO, June 1998.
8. David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.
9. Anurag Acharya, Mustafa Uysal, and Joel Saltz. Active disks: Programming model, algorithms and evaluation. In *Eight International Conference on Architectural Support fo Programming Languages and Operating Systems*, pages 81–91, San Jose, CA, October 1998.
10. Erik Riedel, Garth Gibson, and Christos Faloutsos. Active storage for large-scale data mining and multimedia. In *Proceedings of the 24th VLDBL Conference*, August 1998.
11. Kimberly Keeton, David A. Patterson, and Joseph M. Hellerstein. A case for intelligent disk (IDISKs). 27(3), 1998.
12. Peter Dibble, Michael Scott, and Carla Ellis. Bridge: A high-performance file system for parallel processors. In *Proceedings of the Eighth International Conference on Distributed Computer Systems*, pages 154–161, June 1988.
13. George C. Necula and Peter Lee. Safe kernel extensions without run-time checking. In *Proceedings of the Second Symposium on Operating System Design and Implementation*, Seattle, WA, October 1996.
14. Rober Wahbe, Steven Lucco, and Thomas Anderson. Efficient software-based fault isolation. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 203–216, December 1993.