The Armada framework for parallel I/O on computational grids

Ron Oldfield and David Kotz
Department of Computer Science
Dartmouth College
{raoldfi,dfk}@cs.dartmouth.edu

An exciting trend in high-performance distributed computing is the development of widely-distributed networks of heterogeneous systems and devices, known as *computational grids*. Grid applications use high-speed networks to logically assemble collections of resources such as scientific instruments, supercomputers, databases, and so forth. One important challenge facing grid computing is efficient parallel I/O for data-intensive grid applications. Data-intensive grid applications are particularly challenging because they require access to large (terabyte-petabyte) remote data sets and often have computational requirements that can only be met by high-performance supercomputers. In addition, data is often stored in "raw" formats and requires significant preprocessing or filtering before the computation can take place. Such applications exist in seismic processing, climate modeling, physics, astronomy, biology, chemistry, and visualization.

In this report, we present the Armada framework [OK01] for building I/O-access paths for data-intensive grid applications. We designed Armada to allow grid applications to efficiently access data sets distributed across a computational grid, and in particular to allow the application programmer and the dataset provider to design and deploy a flexible network of application-specific and dataset-specific functionality across the grid.

Using the Armada framework, grid applications access remote data sets by sending data requests through a graph of distributed application objects. The graph is called an "armada" and the objects are called "ships". Figure 1 shows a simple armada for an application accessing applying a preprocessing operator to a distributed data set. We expect most applications to access data through existing armadas constructed by a data set provider; however, it is also possible for the application to extend existing armadas with application-specific functionality or to construct entire armadas from scratch. The armada encodes the programmer's interface, data layout, caching and prefetching policies, interfaces to heterogeneous data servers, and most other functionality provided by an I/O system. The application sees an armada as an object providing access to a specific type of data through a high-level interface. One use of Armada, for example, is to construct complicated data sets on top of legacy files and databases.

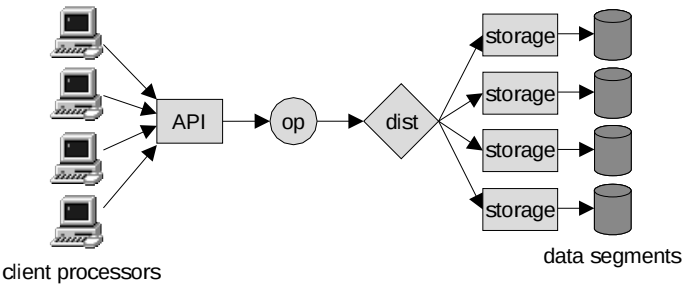


Figure 1: Armada for an application accessing a distributed data set.

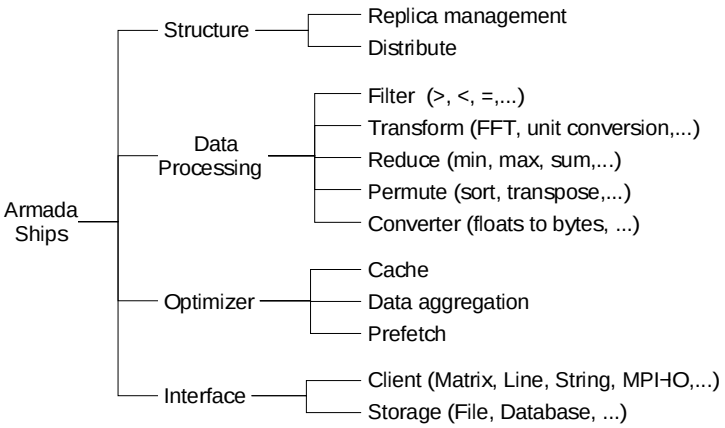
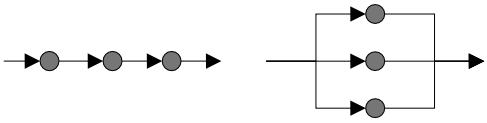


Figure 2: Hierarchy of Armada ships.

Our framework includes a rich set of ship definitions (shown in Figure 2) implemented as Java class objects. Among these are structural ships to manage distributed data sets and replicas; data-processing ships that manipulate data as it passes through the network; optimization ships that use latency-reducing techniques like caching, prefetching, and data aggregation to improve performance; ships that provide high-level interfaces to applications; and ships that interface with local file systems or databases. For more complex applications, the basic ships can be extended to provide application-specific functionality.

The metadata used to describe the interconnection of the



(a) Sequential subgraph. (b) Parallel subgraph.

Figure 3: Subgraphs in a series-parallel tree.

ships and the information about each ship (location of its implementation, hints about placement of the ship, and so forth) is called a "blueprint". We use a series-parallel tree (SP tree) to describe the layout of the ships. In such a graph, a node represents a ship (base case) or a subgraph. Subgraphs (illustrated in Figure 3) take two forms: a sequential subgraph that represents a series of connected nodes and a parallel subgraph that represents a set of nodes connected in parallel. An SP tree is syntactically easy to describe (we use XML) and easy to manipulate internally.

The Armada system attempts to improve performance in three ways: selecting an effective placement of ships in the network, optimizing the data-flow within the armada, and restructuring the graph (which may include replicating ships). Placement of ships will likely have the largest effect on performance. In a typical situation, there may be a slow or congested network connecting sites hosting the application and sites hosting the data segments. When reading a data set, moving data-reduction operators close (in terms of network connectivity) to the data can dramatically reduce the amount of data that travels through the "slow" portion of the network. We can also reduce the number of data transfers by bypassing portions of the armada that do not process the data. For example, many of the structural ships simply route data to a particular location. Finally, in some cases, restructuring the graph may improve performance by distributing compute and network load and possibly moving data-reduction operators closer to the data source. Consider, for example, a parallel application that filters data from a federated data set distributed to two administrative domains. We illustrate this example in Figure 4. The application first constructs a blueprint describing the compute node layout, an interface to the data, and the preprocessing required. It then attaches the application blueprint to a blueprint from the data provider that describes the data layout. Armada restructures the graph, where possible, to remove network bottlenecks and push data reduction filters closer to the data servers. It then assigns ships to hosts within the appropriate administrative domain and initiates the data transfer. In our example, Armada replicates the API ship to place one on each compute node, it converts the filter "Op" to "combine" ships and other "Op" ships that are placed in the local domain of the data servers.

Although several existing software systems support processing of data near the data source, these systems typically restrict the application to a specific access interface. Mocha [RMR00] from the University of Maryland, and dQUOB [PS00] from the Georgia Institute of Technology, require an SQL-like interface. DataCutter [BFK+00], from the University of Maryland, is specially designed to select data subsets through multi-dimensional range queries. In Armada, the application interface will usually be defined or selected by the data provider or the application. In many cases, they may choose a standard interface (e.g., POSIX or SQL); however, some applications may prefer a more specialized interface, for example a collective interface for a parallel application.

The Coign [HS99] distributed partitioning system, from Microsoft, and Abacus [APGG00], from Carnegie Mellon University, are both systems that attempt to improve network performance of distributed applications by selecting an appropriate placement for application components to minimize communication. While our current version places objects manually, our goal is to extend the ideas from Coign and Abacus to place objects based on communication, CPU, and memory requirements of the various components.

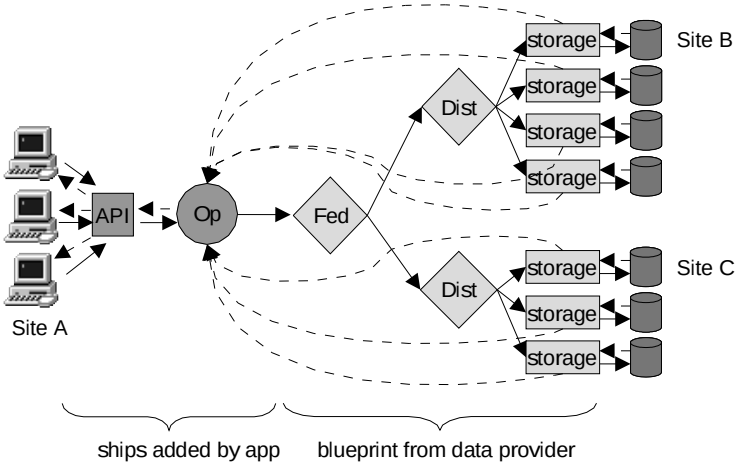
In summary, the goal of our work is to improve performance of data-intensive grid applications by reducing the amount of data transferred through the network and providing a mechanism for arranging access to distributed and replicated datasets. If we can show that our approach successfully meets this goal, our work will impact and influence the way large distributed datasets are managed and accessed across computational grids.

References

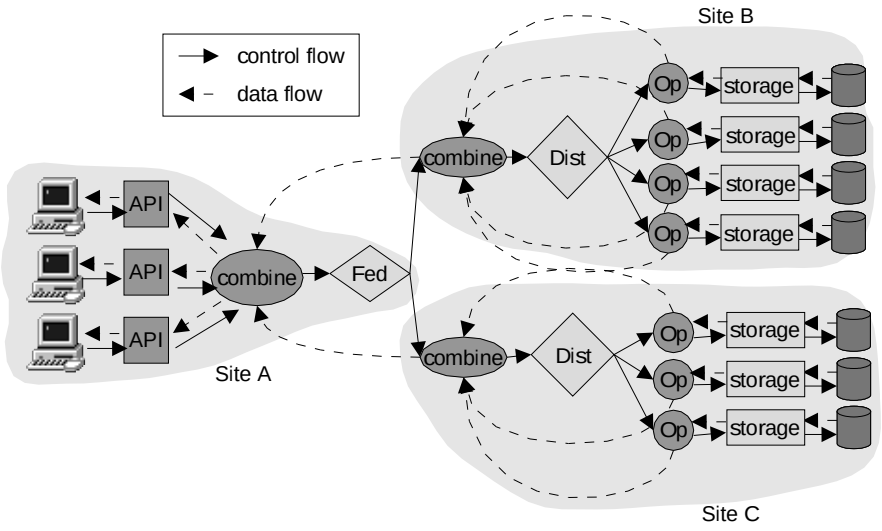
[APGG00] Khalil Amiri, David Petrou, Gregory R. Ganger, and Garth A. Gibson. Dynamic function placement for data-intensive cluster computing. In *Proceedings of the 2000 Annual USENIX Technical Conference*, pages 307–322. USENIX Association, 2000.

[BFK+00] Michael D. Beynon, Renato Ferreira, Tahsin Kurc, Alan Sussman, and Joel Saltz. DataCutter: Middleware for filtering very large scientific datasets on archival storage systems. In *Proceedings of the 2000 Mass Storage Systems Conference*, pages 119–133, College Park, MD, March 2000. IEEE Computer Society Press.

[HS99] Galen Hunt and Michael Scott. The Coign automatic distributed partitioning system. In *Proceedings of the 1999 Symposium on Operating*



(a) Original armada.



(b) Final armada.

Figure 4: Original and optimized armadas for an application spanning three administrative domains.

Systems Design and Implementation, pages 45–56, San Diego, CA, February 1999. USENIX Association.

[OK01] Ron Oldfield and David Kotz. Armada: A parallel file system for computational grids. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 194–201, Brisbane, Australia, May 2001.

[PS00] Beth Plale and Karsten Schwan. dQUOB: Managing large data flows by dynamic embedded queries. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, August 2000.

[RMR00] Manuel Rodríguez-Martínez and Nick Roussopoulos. MOCHA: A self-extensible database middleware system for distributed data sources. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Dallas, TX, May 2000.