From Proceedings of the
IASTED International Conference on Parallel and Distributed Computing Systems
PDCS'97 – Washington, D.C. – October 1997

DATA EXCHANGE: HIGH PERFORMANCE COMMUNICATIONS IN DISTRIBUTED LABORATORIES

GREG EISENHAUER BETH SCHROEDER KARSTEN SCHWAN
VERNARD MARTIN JEFF VETTER

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332

Abstract

Current communications tools and libraries for high performance computing are designed for platforms and applications that exhibit relatively stable computational and communication characteristics. In contrast, the demands of (1) mixed environments in which high performance applications interact with multiple end users, visualizations, storage engines, and I/O engines – termed 'distributed laboratories' in our research – and (2) high performance collaborative computing applications in general, exhibit additional complexities in terms of dynamic behaviors. This paper explores the communication requirements of distributed laboratories, and it describes the DataExchange communication infrastructure supporting high performance interactive and collaborative applications.

Keywords: parallel, distributed, communication, dynamic, interactive

1 COMMUNICATION SUBSTRATES

Communication tools and libraries for high performance distributed computing have evolved substantially during the last few years, from systems enabling distributed parallel computing such as PVM[3], to industrial standards being actively implemented and improved such as MPI[2], to recent proposals for infrastructures that can facilitate the construction of nationwide, networked supercomputers. Our work contributes to such research by addressing several specific issues that arise whenever diverse networked machines are used in research and development settings or even in production environments, by single or multiple end users. The target applications we consider are those in which end users interact with powerful computational tools and with each other across heterogeneous networked machines, termed *distributed laboratories.* The communication demands of such dynamic computational environments share some commonalities with traditional distributed systems as well as with the highly dynamic environment of today's web browsers and CORBA-based distributed object systems, but unfortunately these similar environments do not adequately address the high performance needs of some applications. Moreover, these environments' dynamic communication and computational behaviors are not addressed by standards for distributed high performance computing such as MPI.

Our work assumes that end users should be able to exploit the potentially high performance of vendor-provided implementations of standards like MPI when appropriate for their applications. In addition, when application requirements can be stated statically, users should be able to exploit the automatic support for parallelization and for efficient inter-task communications offered by compilers, their runtime support, and specialized communication libraries. Given these assumptions, the Distributed Laboratories Project[5] at the Georgia Institute of Technology is developing communication support that addresses the interactive and dynamic nature of high performance applications, and enables their use in scientific investigations or in production environments by single and by multiple end users. Specifically, the *DataExchange* communication library utilized in the implementation of several distributed high performance applications offers the following functionality:

- a publish-subscribe communications model,
- fast heterogeneous data transfer, and
- support for application-level diversity.

In the remainder of this paper, we characterize the communication characteristics of high performance applications when they are used in distributed laboratory settings. Next, we identify the requirements these characteristics impose on the underlying communication infrastructure. Using examples from the high performance interactive applications being developed at Georgia Tech, we also describe the DataExchange communication library.

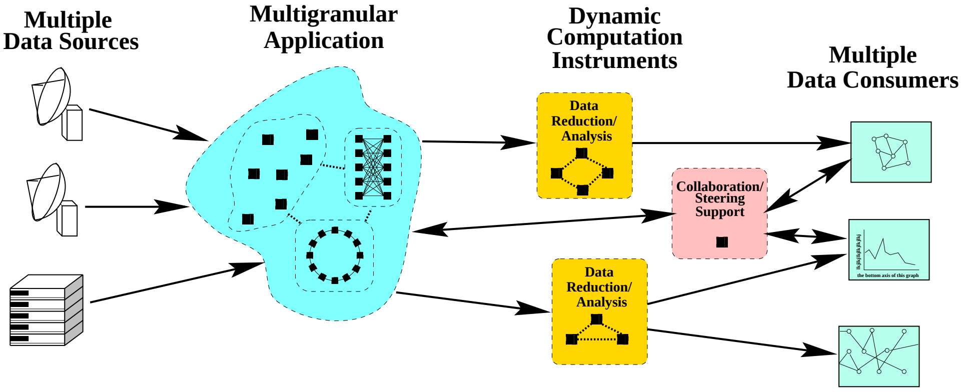

Figure 1: A SAMPLE DISTRIBUTED LABORATORIES APPLICATION CONFIGURATION.

2 DISTRIBUTED LABORATORY ENVIRONMENT

2.1 Characterization

In order to illustrate the challenges posed by distributed laboratory environments, we first consider a specific application. In particular, consider a large scientific simulation running on a set of computational resources, such as a global climate model being developed for networked parallel machines[4]. This global model might interact with more detailed local climate, atmospheric or pollution models in order to enhance the accuracy of both the local and global models. These models may run concurrently on a variety of parallel and distributed computing resources, or, ideally, any one of the global or local models should be able to be replaced at runtime with a historical database containing information from previous model runs. In addition, model outputs may be processed using a variety of instruments, including specialized visualization interfaces or computational instruments performing calculations which derive from and expand upon the basic model results. Similarly, model inputs may be provided via other instruments that utilize live satellite feeds or access stored satellite data on remote machines.

Consider the introduction of observers into this application scenario. Given the scale of such an application and the speed and complexity of its execution platform, it is easy to imagine that the simulation's progress and results are monitored by multiple scientists in distributed locations. The scientists' interests may vary from wishing to see the "big picture," to investigating in detail subsets of the simulation's output, to collaborating with each other via the computational instruments these models implement.

The distributed application has additional components that assemble the information needed to drive various interactive displays, by gathering data from the distributed simulation and performing the analyses and reductions required for these displays. Some of these components may themselves have substantial computational or storage needs and require dedicated additional resources of their own. They may also require access to additional information, as is the case for the atmospheric model's display with which end users compare observational (satellite) data with model outputs in order to assess model validity or fidelity. Moreover, since end users control the set of computational instruments, input and output components, and the displays, the current set of interacting computational instruments will change dynamically, driven by end users' needs or by the current needs of the running simulation. Figure 1 depicts a sample application configuration.

Taken as a whole, this application scenario has requirements for data management and exchange that go well beyond those of a traditional parallel or distributed application. In particular, it presents a need for efficient high-performance data exchange coupled with data location and analysis in a highly dynamic heterogeneous environment. These communication needs are discussed in more detail in the next section.

2.2 Communication Requirements

2.2.1 Flexible High-bandwidth Data Transfer

The need for high bandwidth and low latency in communications is common to most high performance applications, and those running in the distributed laboratory are no exception. However, distributed laboratory applications have several additional communication demands:

1. *Data-driven, time-constrained computations* – the communication infrastructure must provide for suitable latencies of the data transfers from input engines to computational instruments, from instruments to observers' displays, and between multiple, collaborative displays.

2. *Scalable communications* – because any number of different data generators and consumers may co-

exist in a large-scale laboratory, it is not sufficient simply to transfer data, but the infrastructure must also permit suitable transformations on the data as it is being transferred, and it must provide mechanisms for efficient data distribution and collection among large numbers of clients.

As a result of these additional two demands, the distributed laboratory's communication infrastructure cannot rely on inefficient techniques like ASCII transmission of data to handle such issues as heterogeneity in the execution environment. The infrastructure must also facilitate the association of computations with data transfers, as explained in Section 2.2.4 below.

2.2.2 Operation in a Diverse Environment

The application environment described above is clearly heterogeneous, but it can perhaps be more strongly described as *diverse*. This means that, in addition to operating in an environment where machine-level representations of elementary types may differ, the various components of the application may be developed in a much less tightly coupled manner than a traditional parallel or distributed program. That is, they may not all be produced by a single group or organization and they may include "generic" components designed to operate in many situations.

One implication of this application diversity is that it is not generally practical for all of the components operating in the distributed laboratory to agree at compile-time on the formats of all data records to be exchanged during execution. Even if agreeing on a common and consistent format for the data were possible, systems depending upon such agreement are very fragile and unsuitable for a dynamic environment. This is a problem that has not been addressed in the past by high-performance communication systems. The most obvious approach to operating in an environment this diverse is to transmit data in some easily-interpreted free-form manner, such as encoded in ASCII, and to parse it upon receipt. This approach achieves the required flexibility, but at the cost of seriously compromising the efficiency of the communication system, and so it is not an option for a distributed laboratory.

2.2.3 Dynamic Connection of Clients and Data Flow Management

In the distributed laboratory environment, computational agents may be very long-running, and physical instruments may always have data available. In this environment it is natural to expect displays, their associated data processing agents and other components of the execution environment to come and go as computations proceed. To accomplish this, external clients must be able to locate and connect to data sources dynamically.

The potential for dynamic connection of clients also means that the flow of data and the processing needs experienced by these flows in the distributed laboratory are themselves very dynamic. In particular, it implies that senders may not have *a priori* knowledge of the identity of all possible receivers. This is a problem for communications systems which require destinations to be explicitly specified by the sender, such as traditional send/receive and RPC-style communication mechanisms. One can, of course, program the application such that it responds to the dynamic connection of clients and automatically routes information flows to interested parties. However, such implementations tend to make inefficient use of network resources[7] and to require the application programmer to create some complex code on top of the basic one-to-one communication mechanisms. Using a generative, or publish-subscribe, communications model instead hides the complexity inside the communications layer and gives that layer more flexibility to exploit the capabilities of the underlying network.

2.2.4 Information Filtering and Analysis

As already stated above, a communication infrastructure for distributed laboratories must manage both data flows and the necessary computations controlling those flows and their content. The resulting richness and dynamic nature of the environment and its traffic create additional opportunities for optimization and improvement. For example, a capable communications system may take advantage of external knowledge of the application and reduce overall communication requirements through the use of application-specific filtering and analysis. Because of the application-specific nature of the computation, the main contribution a communication infrastructure can make is in easing the construction of filters and computational agents and providing simple mechanisms through which application-level routines can be activated.

3 CONTRIBUTIONS

The DataExchange library addresses the requirements described in Section 2. First, DataExchange is based on PBIO[1], a binary I/O package which provides a variety of mechanisms for handling operation in a diverse environment. Second, DataExchange provides facilities for naming and locating data sources, for dynamically connecting to those data sources and for automatic and configurable redirection of data flow. Third, to support flexible data processing, analysis and reduction, DataExchange also offers an active-message-style processing message processing facility. This processing facility can also serve to further refine and extend the data flow management functionality by making content-based message forwarding decisions. The remainder of this section will briefly examine these features of DataExchange.

3.1 Binary I/O Support

In trying to create reusable distributed laboratory tools, one would like to have displays and filter programs that could be applied to a variety of programs and situations in the distributed lab without requiring *a priori* knowledge of the data formats involved. However, message passing systems which rely on implicit receiver/sender knowledge of the data formats being exchanged do not generally transmit sufficient information for an external party to meaningfully interpret a message in an unknown format.

DataExchange relies on PBIO to support its data transfer operations. In addition to other features, such as named data types and the ability perform type conversion between matching fields of different basic types, PBIO supports more elaborate and flexible type matching than either PVM or MPI. In particular, PBIO supports type matching based on *field names* rather than just elementary types. PBIO's type matching is also more flexible in that receivers are not required to specify every field in the types to be exchanged, but need only specify the names and types of the fields that they are interested in. Receivers will automatically extract the named fields of interest and ignore unspecified fields.

Efficient transmission using name-based type matching is possible because PBIO transmits meta-level **format descriptions** before transferring any data records. A format description consists of the name of the record (message) format, and the names, sizes, offsets and data types of the each of the fields in the record. A previously registered format may also serve as the basic type of a field, allowing the creation of complex nested record types. PBIO also supports fields which are statically sized one or two dimensional arrays or dynamically sized one dimensional arrays of simple or derived datatypes. Record meta-information is transmitted only once. Thereafter, transmission occurs in the writer's native format and the PBIO library on the receiver transparently handles discrepancies between the writer's format and the format required by the reader. Transmitting in the writer's native format allows complex messages to be sent over the wire without any copy or "buffer building" stage. In the case of transfers between homogeneous machines, the only additional overhead imposed by PBIO is the transmission of a 4-byte format ID.

Because PBIO record meta-information is transmitted on the wire rather than relying on implicit knowledge in the sender and receiver, the task of creating generic tools and displays that can be "plugged in" to applications in the distributed laboratory is eased. PBIO also provides a variety of facilities for manipulating format meta-information and extracting individual fields. These functions support the special needs of programs which are required to process a portion of the data in a record and forward the remainder untouched.

3.2 Location, Connection and Flow

The dynamic nature of the distributed laboratory environment creates the requirement that a new display be able to locate and connect to information sources and that communication topology be dynamic at run-time.

Dynamic location of communicators is seldom directly addressed by communication packages. Basic socket implementations require the use of a port number that is known to all parties, and there are no common mechanisms for publishing a dynamically created name/port number association. PVM supports communication with a dynamically forked remote process, but lacks the ability to initiate communication between two unrelated processes. MPI contains no support for initiating communication between unrelated processes, although MPI-2 will provide some primitive communicator location facilities. CORBA[6] recognizes the problem of dynamic location and includes a specification for the CORBA Name Service (which is getting widespread implementation). However, the CORBA name service is not useful outside of the CORBA domain. While a naming service to support communication in the distributed laboratory need not be overly complex, it must be available.

The ability for a client to selectively receive only a portion of the data available from a particular source is also a capability that is seldom supported by communication libraries. However, the distributed laboratory environment has the potential for large flows of scientific and monitoring data. While clients always have the option of receiving everything and discarding that data which is not of interest to them, adding the ability for them to specify what is interesting so that the uninteresting is not transmitted (and so does not take bandwidth) may be key to reducing overall network resource demands.

The facilities for locating, connecting to, and controlling flow between data sources are explained in greater detail in Section 4.

3.3 Information Filtering

In a distributed laboratory environment, it is useful to be able to create generic filter and display programs that can be assembled in a plug-and-play manner. Both filter and display-type programs are often constructed in an event-driven programming style. Like a variety of other packages, DataExchange provides the ability to bind application level functions to communication system events. DataExchange events include the connection of new clients, shutdown of client connections, and message arrivals. DataExchange also allows different functions to be bound to each type of message. With this level of support, simple filter programs can often be reduced to a very few lines of code.

This basic support for binding application functions to message arrivals also provides for the creation of data-dependent routing and forwarding of messages. Data-dependent routing could be used to supplement the record-type-based forwarding in DataExchange.

4 DATAEXCHANGE LIBRARY

DataExchange facilitates communication between processes in a distributed laboratories environment through a set of services that provide the behavior needed by applications. For example, a visualization client receiving large amounts of data from a global atmospheric model may at some point no longer need messages containing updated species concentration values. Instead of being forced to discard the species concentration messages upon arrival, DataExchange provides a means for the client to issue a command to the server to stop sending messages of that kind, thus freeing bandwidth that would otherwise be used. DataExchange currently executes on Sun Solaris, SGI, Linux, RS6000, and Windows NT platforms.

4.1 Concepts

Before discussing the services provided by DataExchange, we introduce several general concepts, an understanding of which is useful for understanding those services. One enables Dataexchange communication by creating a *DataExchange context*. The context, specific to a single process, supports a group of related connections. The separate communication context insulates communication internal to the library from external communication and provides a means of associating related communication channels. A DataExchange context is created with a call to `DExchange_create()`, which creates the context by initializing a `DExchange` data structure. The returned pointer to the data structure is attached to subsequent calls to the library.

An individual connection has associated with it a *DEPort structure*. The DEPort structure maintains information about the connection, such as its hostname and port id. A DEPort structure is created automatically when a connection is established.

4.2 DataExchange Services

4.2.1 Dynamically Locating and Connecting to Data Sources

In a distributed laboratories environment it is natural to expect displays, their associated data processing agents, and other components of the execution environment to come and go over the course of time. A library that supports dynamic locating and connecting to data sources must provide several services: dynamic location of communicators, dynamic connection to the communicator, and the ability to direct events to the new client without disrupting the data source.

Dynamic location of communicators is provided through a *group server*. The group server is a simple name service that relieves an application of the responsibility of knowing the specific location <hostname, port> of a connection point. The application need only know an agreed upon name to locate and connect to a connection point. The group server implements its service as a repository of name, connection pairs. The name, a <user name, application name, group type> triple, uniquely identifies a connection point. *User name* is usually the userid associated with the process establishing the group. If not given, it is derived from the execution environment. The *application name* is an agreed upon name identifying the set of communicating processes. *Group type* is provided to further delimit the application name. Group type is used in cases where an application supports several connection points for different purposes. For example, one of the processes participating in the global atmospheric simulation may establish a connection point for all distributed components of the simulation. That same process may need to establish a communication point to enable communication between the model and the filters and visualization/steering clients. The two communication 'groups' would be established with the same application name but differentiated by group type.

The group server accepts queries based on any field of the <user name, application name, group type> triple, and supports both exact matching and regular expressions. In response to a query, the group server returns a list of connection points from which the process originating the query can select.

Dynamic connection to a data source requires the application to respond to arriving events, including connection request events. Event-based processing is supported by the `DExchange_poll_and_handle()` routine, the basic event handling call in DataExchange. The poll and handle function is analogous to the `XtAppNextEvent()/XtAppDispatchEvent()` pair in X windows toolkit programming where a DataExchange event is a basic communication occurrence, such as a connection request or message arrival. The function queries all communication channels associated with a DExchange context and processes events from each connection having input ready. The poll routine has a `block` parameter which specifies whether or not DataExchange should block (suspend program execution) waiting on an event if none is ready.

In addition to services for dynamically locating data sources and accepting connection requests dynamically, a communication library meeting the dynamic needs of distributed laboratories must also provide a means for a newly connected client to automatically begin receiving data for which it has registered an interest. DataExchange provides such a service through default forwarding behavior. A DataExchange server will automatically forward all messages it receives to all connections that have registered an interest in the messages. This default behavior can be modified through control functions described in Section 4.2.3 that allow the selective forwarding and blocking of messages based on format description.

4.2.2 Analyzing and Filtering Data

Many event-based applications must handle arriving messages of multiple data formats where often the action to be taken is specific to the data format. DataExchange provides a means for associating an action to a data format through **action routines**. Action routines are routines associated with a particular data format and invoked when a message of that format arrives. Similarly, applications may register routines to be called when a connection is established or shutdown. Action routines may be registered and unregistered with DataExchange at any time.

DataExchange provides two types of action routines for handling incoming data, functions and filters. **Functions** are called *after* any implicit data forwarding whereas **filters** are called *before* data forwarding and can return a value which preempts further processing. Functions are useful for gathering statistical or debugging information on the event stream. Filters, on the other hand, are useful for filtering events, reordering an event stream, or consuming events and generating derived events. Functions and filters are associated with specific data formats though the same subroutine can be registered as a handler for multiple data formats.

4.2.3 Controlling Flow of Data

As mentioned previously, the default DataExchange behavior is to resend all data received from one connection to all of its other connections (except the sender). This mechanism is useful for constructing interesting servers and managing data flow in networks of communicating DataExchange programs. The DataExchange library allows control of this resending in ways which can be loosely divided into *record forwarding* and *record blocking* controls. Record forwarding and blocking controls both affect the implicit resending of received data, but in different ways. Forwarding controls are available to the DataExchange program performing the resending operation and allow that resending to be controlled on a per-format basis. Forwarding controls are local controls in the sense that a DataExchange program calling those routines affects its own resending. The complementary record blocking controls are instead used by the DataExchange programs *to whom data is being resent* to control the type of data that is resent to them. In particular, by using the record blocking controls a DataExchange program can tell another communicating program *not* to send it records of a particular format.

5 CONCLUSION

The current trends in high-performance computing applications are towards large loosely-coupled distributed computation and more dynamic models for both the computation itself and for run-time interaction with the computation. Unfortunately, these trends are not well supported by traditional message passing environments. This paper describes a sample application from an application class which we are examining as part of the Georgia Tech Distributed Laboratories Project and discusses its communication infrastructure requirements. These requirements are used to drive the discussion of DataExchange, a communication infrastructure we have developed to support the creation of high performance and collaborative applications in the Distributed Labs environment.

References

[1] Greg Eisenhauer. Portable self-describing binary data streams. Technical Report GIT-CC-94-45, Georgia Institute of Technology, 1994. *(anon. ftp from ftp.cc.gatech.edu)*.

[2] MPI Forum. MPI: A Message-Passing Interface. Technical Report CS/E 94-013, Oregon Graduate Institute, March 94.

[3] Al et.al. Geist. *PVM 3 Users Guide and Reference manual*.

[4] Thomas 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.

[5] Beth Schroeder, Greg Eisenhauer, Karsten Schwan, Jeremy Heiner, Vernard Martin, and Jeffrey Vetter. From interactive applications to distributed laboratories. *To appear in IEEE Concurrency*, 1997.

[6] Jon Siegel. *CORBA Fundamentals and Programming*. John Wiley & Sons, Inc., 1996.

[7] Mark D. Wood and Bradford B. Glade. Information servers: A scaleable communication paradigm for wide area networks and the information superhighway. In *Proceedings of the 7th SIGOPS European Workshop*. ACM, 1996.