A High-Performance Active Digital Library

Roy Williams
Center for Advanced Computing Research, Caltech

and

Bruce Sears
Space Radiation Laboratory, Caltech

Abstract

We describe Javaflow and Paraflow, the client and server parts of a digital library, providing high-performance data-retrieval and data-mining services, with emphasis on user interface as well as computing efficiency. Paraflow is a component model for high-performance computing, implemented as a thin layer on MPI (Message Passing Interface); it controls a heterogeneous metacomputer, allowing groups of processes (services) to work collectively and communicate with each other by parallel messaging links (channels). Javaflow is a straightforward, intuitive, loosely-coupled component model for Java applets, allowing them to be combined into a GUI that can run on a thin client from a standing start. Javaflow controls Paraflow through a well-exposed text interface which is carried by authenticated telnet, while the output of the computation can be web-pages, files, or high-speed graphics. As an example, we discuss the SARA remote-sensing library, which provides public retrieval of multichannel images of the Earth, as well as private, authenticated access to a variety of image-processing services running on parallel supercomputers.

Introduction

Parallel supercomputers and high-bandwidth communication technology is capable of providing enormous processing speeds and high-speed transfer between processing engines, while RAID disk and tape-robots can store the vast quantities of data that drive these systems. But using these systems entails learning where to find the compiler, TCP/IP port numbers, byte-ordering conventions for different systems, and other tedious details. Collaborating on a software project often involves inventing a communication protocol at the byte level, then writing raw socket code. Furthermore, using the machines themselves often involves a morass of scheduling interface, allowing the computer to choose when to run a job rather than the human user doing so.

But to many people, computing means something quite different: a web-client machine at home which can browse remote databases, carry on a chat-group, or download movie previews. Under the surface, complex events occur that may involve several machines, but these are hidden from the user, enabling him to think about an objective rather than its implementation. In fact, web-servers already provide much of the software infrastructure that makes distributed computing easier: the MIME mechanism to simplify the processing of heterogeneous data types, HTML forms for data input, multithreading, and connectivity to databases. But perhaps the most important advantage of a web interface is the recognition of a web-browser by a neophyte user and the consequent feeling of control.

In the old days, a scientific experiment or computation produced a result that could be immediately apprehended by the experimenter. A current trend, however, is that data analysis is a major part of the work involved in the simulation or experiment; examples of this might be recording a terabyte of data in a few days, from a radio telescope[1], gravitational wave detector [2], or electron microscope[3], then spending a year searching for the interesting signals in the data. The same pattern reappears in computer simulations of cosmology[4], neurobiology[5], and other fields, where the real work begins after the supercomputer run.

Scientific collaborations are already distributed across continents, and software to enable these work groups will become increasingly vital. Such software should enable access to the digital data archive that represents the work of the group, using the Internet to retrieve, process and mine the data, as well as allowing the researchers to codify, annotate, and talk about the resulting knowledge. It will become increasingly necessary for heterogeneous digital archives to interoperate in increasingly sophisticated ways [6]; it will be necessary for human interfaces to these archives to gain a huge measure of utility, simplicity, and flexibility; it will be necessary for the scientific establishment to quickly leverage software advances from the commercial sector into their traditionally conservative environment.

In this paper, we describe an architecture for a digital library, called Paraflow/Javaflow [7], and its initial implementation for a library of remote-sensing data [8]. By the term “digital library”, we mean ways to find and present data, perhaps also with simple filtering tools. By “active digital library” we also imply powerful ways to mine the data for knowledge through classification, statistics, pattern recognition and visualization.

Javaflow uses a component model such as JavaBeans to provide the user with guidance around the library, perhaps a ready-made GUI for a specialized application, perhaps a collection of forms to fill in, like Microsoft’s wizards, or at a more abstract level, GUI components can be combined to build something new. Each component of a Javaflow GUI can produce a text string as one of its methods: this text string can be passed to the Paraflow virtual machine as all or part of a command. When Javaflow has provided an interface by which the user can interact with data from the library, Paraflow can execute the commands on high-performance parallel supercomputers.

Paraflow addresses the paradigm of “data computing”, characterized by a low ratio of floating-point operations to bytes transferred — the “flops-per-byte” that characterizes an application. When this ratio is high, we have traditional number-crunching, where disk, tape and network I/O speeds are not important, but the objective is to make arithmetic fast. For meaningful knowledge discovery from remote-sensing data, however, we must emphasize the opposite paradigm, where computing is cheap, but the difficulty is delivering the data to the processors. Data are taken by a high-bandwidth parallel streams to a heterogeneous, distributed metacomputer, after which these results are archived. Later, a client does specialized, on-demand computing on this archive, from which precious morsels of knowledge are extracted. Thus only the products actually requested by the scientist need be generated and they are determined interactively as previous results are evaluated.

Paraflow/Javaflow allows an authorized user to do meaningful work from any Net-connected computer anywhere, to browse and search, to do interactive filtering and processing, to launch a multiple-hour production run on a parallel supercomputer. The web-based architecture is also designed to allow sharing of results through annotation conferencing, reuse of both processing

algorithms and presentation widgets, and a straightforward way to compile and link a new component into the system.

Javaflow is a component-based GUI for text-based control of an application based on the JavaBeans architecture. Paraflow is a component architecture based on MPI which uses text-based commands to control the metacomputer. The two may be used separately or together.

High-Performance Computing Meets the Web

The Internet and Web have and will continue to revolutionize scientific collaboration and research, and a library as described here would lose a great deal of its impact if it were not accessible at many different levels from the Web. In addition to the Web (http) protocol, we use an implementation of the Message Passing Interface (MPI) [9] that can support heterogeneous computing and I/O, and also a secure telnet protocol as discussed above. In this section we discuss how the http protocol will be used, and below we discuss the Paraflow library that uses MPI communication.

The flexibility and universal acceptance of the http protocol is unprecedented: heterogeneous data, and programs too, can be passed to a client on a different continent, geographically and culturally, with confident expectation that she can make use of the data. The huge industrial investment in multithreaded software and powerful multiheaded servers may be another opportunity for HPCC projects to use low-cost, mass-produced commercial products that are also of high quality. Given servers with large TCP windows and low latency networks, there is no reason why we cannot have both flexibility and speed at low cost.

Web technology also acts as a bridge from conventional HPCC with its files and directories, to a richer set of names, criteria, and questions that concern the data, not the computer. A relational or object database management system (DBMS) allows complex queries: by spatial position, by name, by spectral class, by date, or correlations of these. Rather than working directly with the DBMS, we intend to use web servers as request brokers, hiding the implementation of the query engine from the rest of the system. In this way, a DBMS can be swapped out and replaced with another without changing the facade it presents to the world. For example, a query such as “Datatype=Landsat,Longitude=120,Latitude=34” might be sent to a metadata web server, which responds with a web page containing hyperlinks to datasets that cover this point on the surface of the Earth.

In the following, we will discuss how clients can retrieve data from the public SARA remote-sensing library, then the interface to the active version of the library: an image-processing system that can do supervised and unsupervised data-mining algorithms.

The SARA Digital Library

Retrieving Data: The Digital Library

SARA [8] is a digital library of multispectral remote-sensing imagery of the Earth, 40 Gbyte in total, acquired by the space shuttle in 1994/95. The data is partially replicated on disks and tape robots at Caltech, the San Diego Supercomputer Center, and the University of Lecce, Italy. While the data is kept in different kinds of file systems (Sun NFS, IBM/Livermore HPSS), it is delivered by web servers: these are acting as *request brokers*, insulating the client from the different server

implementations. A data request is a URL that contains the ID-number of the requested dataset, the desired data format, the requested image channels, and the pixel coordinates of a sub-image of the full dataset.

By *data* we mean large data objects obtained by simple requests, and by *metadata* we mean small quantities of data accessed by complex requests. A data request could be a file name and a simple filter to be applied to that file, whereas a metadata request might be a SQL query to a database, whose output is a list of files which satisfy the query. Metadata is available from web-enabled database systems, converting longitude and latitude to dataset ID's, converting these ID's to filenames, and so on; metadata queries are also insulated from implementation by web servers.

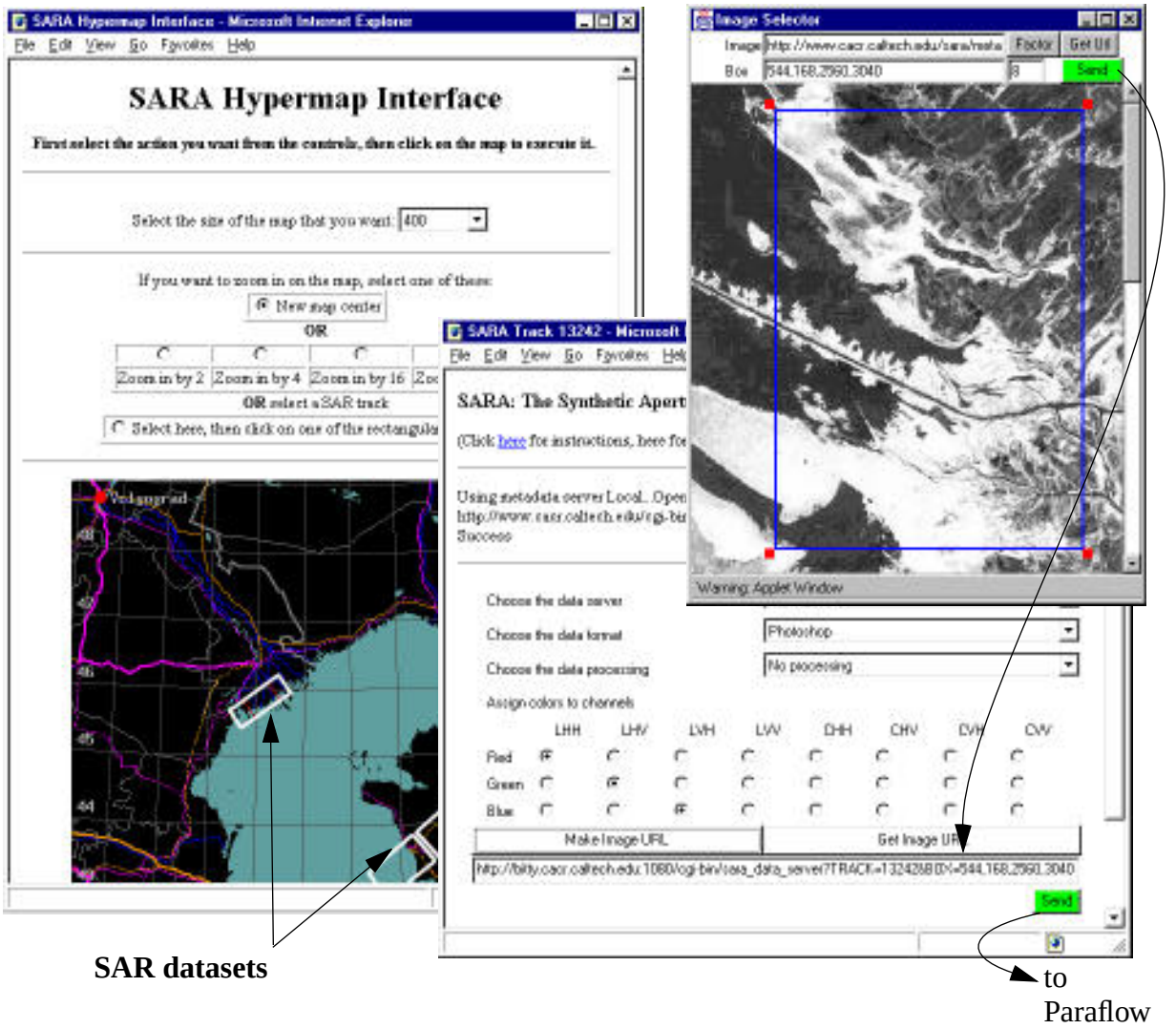

Figure 1: User interface to the public part of the SARA digital library. A multichannel dataset is chosen from an interactive world atlas (left), then a subset of the data is chosen from a single-channel, one-eighth scale thumbnail. Output from this component goes to the main panel, where it is combined to produce a URL, that can be viewed directly or used as input to the private, active part of the library.

Data and metadata are distributed and replicated over geographically-separated file systems, providing fault-tolerance. When a metadata server does not respond in some way — connection refused, time-out, file not found, etc. — then SARA simply tries another server from a list. Only if all the metadata servers fail does it give up.

For the SARA project, data is stored at Caltech, SDSC, and Lecce, Italy, linked by OC-12 Hippi, the vBNS and trans-Atlantic Internet. Caltech and SDSC each have 40 Tbyte IBM tape robots, which can operate at transfer speeds of up to 40 Mbytes/s. Each of these systems is already utilizing the High Performance Storage System (HPSS) in production mode. The data consists of sets of Synthetic Aperture Radar (SAR) channels, with each channel being a file between 5 and 100 Mbytes. Channels are monochrome images of the surface of the Earth: in general several co-registered channels have been taken together to form a track. The SARA interface delivers and analyses these rectangular subsets of tracks.

Data Computing: The Active Digital Library

“Computing” in the context of the proposed digital library may mean several things: a scientist may be interactively analyzing images with code she has written, there may be a scheduled run through the entire database to gather information, or it may mean a regular production run to ingest new data into the system, simultaneously skimming off metadata. In the following, we describe the Paraflow library: this allows such tasks to be decomposed with two distinct paradigms of parallelism: SPMD data parallelism and dataflow functional parallelism.

SPMD (Single Program Multiple Data) parallelism is, for example, splitting a large image across several processors so that processing speed is increased and large datasets can fit in memory. As the name implies, each process of the group is running the same program. By contrast, in dataflow functional parallelism, different processors have different programs which are connected in a directed acyclic graph, each process waiting to receive data from its inputs before sending results on its outputs. Paraflow combines these paradigms by considering a dataflow graph of ‘services’, where each service is an SPMD collection of processes. Paraflow provides an MPI communicator to each service and a local process rank, so that each service can send messages within its group of processes, so that it can do collective operations such as reduce, barrier, or make parallel I/O calls.

Connecting the services are ‘channels’: since the services are parallel, this implies that the channels are also parallel. Paraflow provides different kinds of channels, for different purposes, and we have implemented three kinds so far: Broadcast, Round-robin, and Incoherent. When one of the processes from the sending service sends a data packet into the channel, one or more of the processes of the receiving end of the channel will get the packet, as determined by the behavior of the particular type of channel.

Access to a Paraflow virtual machine is through a gateway process which provides a command interface that allows the user to associate text commands with functions to be executed on the services of the metacomputer. Paraflow also provides a mechanism allowing data access via web servers.

In the case of the SARA library, we have implemented reading and writing images as Photoshop and Jpeg formats, with other methods for changing image size, principle component analysis of a multichannel image, and a supervised classification with training areas selected from a thumbnail

image, then a simple “closest-to-mean” classification. We set up services for reading and writing, with just one process each, and a third service for the computation with many processes.

Operating the SARA library

The steps involved in operating the SARA active library are as follows. From the home page, use the map of the Earth and the zoom tools to select an area where data is available, known as a track, if desired, choose a subset of this track for analysis. The finding tool now provides a URL whose content is the multichannel dataset to be analyzed by the active system; the content may be hundreds of megabytes, so it may not be a good idea to attempt a direct download of this to the client. Instead, this data will be loaded into a parallel computer which is presumably close to the data source. Now the user can securely log in to her account at the computer center, using the telnet client and one-time-password generator applet. The user can select which computing resources to use, then create a virtual machine from these resources — a heterogeneous parallel computer — and start the SARA image processing executive. The dataset selected above can be read into the virtual machine, and derived images created by a number of classification methods. The results of computation, images or text, can be written to web pages and examined with a browser; further data can be imported and processed. Scripts for production runs that run for hours or days can be created, scheduled and run from the executive.

The Client

The Thin Client for Portability

Javaflow provides a user interface to the library that is based on the “thin client” model, which assumes as little as possible of the client’s terminal: specifically, we shall assume that the thin client has a connection to the Internet and a java-enabled web browser, and no other software or hardware. Thus we hope to encourage collaborative access to this library — from the scientist at a conference, in a cybercafé, in a colleague’s office. Furthermore, the thin client model enables a quick start at the digital library, without reading manuals or meticulous software installation.

The client is connected to a metacomputer (a combination of computing and data services) through typing text commands, or using the Javaflow interface (see below) to generate the text commands from a menu. The metacomputer is running the Paraflow software (see below), a thin layer over MPI, and the results of a computation may be written conventionally, as files in the data archive; or as web pages which are accessible to the user that created them. One advantage of using web pages as direct output from a computation is multimedia: images can be of many different formats, compressed, multichannel, hyperspectral, or whatever. Another advantage is that the pages can be organized and classified by linking them from HTML header pages, using output from metadata servers.

As the client becomes richer, it is not difficult to integrate non-Java software into the thin client. Since the model is based on a web browser, we can attach browser plug-ins and helper applications. For example, the server could make web pages for the client consumption that consist of Adobe Photoshop or VRML files, or serve GIS data to an ArcView or MapInfo plug-in. Other kinds of data might also be created, such as Excel spreadsheets, matrices in Matlab format, and so on. These files can be connected and catalogued by collecting them into HTML documents and database tables. Client graphics that uses scripting languages can also be accommodated from

the server side, such as Tcl/Tk, xmgr, or PostScript.

Supergraphics for the Rich Client

The thin client model enables a fast learning curve, flexibility, portability, and encourages collaboration; unfortunately, however, it does not allow the kind of high-speed access to data that we have come to expect from HPCC systems.

The rich client model is not a change of user interface from the thin client model, but an extension of it. The rich client will allow the system to provide parallel high-bandwidth streams of images, video, and virtual reality, with floating, embedded Java applets to control the computation. In this fashion, we hope to utilize human intelligence for knowledge discovery to the greatest possible extent, by maximizing the rate at which the senses can communicate to the brain. Once a secure control connection is established between client and a Paraflow gateway, as described below, a group of servers may establish a (perhaps parallel) connection to a supergraphics machine, at OC3 bandwidth and above, which is separate from the original connection. Data can flow either by a TCP/IP socket or by a raw, custom protocol, perhaps in many striped streams; in either case, the objective is to ensure that no protocol processing gets in the way.

Security and Authentication

The difference between public and private libraries is in accounting and authentication. The professional running a terabyte of data through a parallel supercomputer should have authority to use these resources, whereas the browsing schoolchild should be able to take a small amount of data with very little bureaucracy. While the latter accesses public web pages and Java applets, the former must be authenticated.

The authentication system assumes the thin client model, as above, and that the client machine can be anywhere on the Internet —meaning that it is unknown to the server. This implies that we need a user-based authentication control rather than host-based. Thus client is assumed to have a an account with a username/password that is valid for a number of resources at the computer center, including the supercomputers and data resources that the client wishes to use; we also assume that the client has a home directory where files can be written, and from which a web-server can deliver data.

In the thin client model, the client gets Java applets from a webserver that is at the computer center, and one of these is a telnet client applet such as the one from NACSE [10]. When the client chooses a machine and tries to connect, the telnet applet attempts to make a socket to the telnet demon on that machine; because of the Java security model, there is only one machine for which this can succeed, which is the machine which provided the telnet applet. When this login has occurred, the client is authenticated: she can read, write and delete in her part of the file system and start jobs on supercomputers. All the resources used by the client are correctly accounted by the computer center because the client is logged in under her own username. The client can send and receive text data via this telnet session, and/or the text can be synthesized and interpreted by the Javaflow system to provide a wider spectrum of multimedia interaction.

The simple scheme using telnet has the disadvantage that a password is being sent across the open Internet in unencrypted plain text — a serious security problem given the potential power available to a successfully authenticated client. We have chosen to solve this with the popular

“Skey” one-time password systems: on the server machine (web server and telnet demon), the standard telnet is replaced with a version using Skey authentication, so that when a user connects to the telnet port, she is asked for a one-time password identified by number, for example “give me password number 96”. This password can be easily computed from the private password known only to the user, but it is only useful for one login session because next time the user will be asked for password 95, then 94, and so on. If a hacker intercepts one of these one-time passwords, it is useless for a subsequent login.

Users may use the one-time password security system with a Java applet [11] that computes one-time passwords that is part of the Javaflow component collection.

Once a user is authenticated on one machine at the computer center, we would like to launch jobs on other machines. If we do not encrypt the session, then we may either use more one-time passwords for these subsequent logins, or we may assume that the computer center has a host-based trust fabric such as that provided by the popular ssh package.

Paraflow: Heterogeneous Parallel Supercomputing

For scalability and portability, we have chosen to implement Paraflow over MPI (Message Passing Interface). An MPI metacomputer consists of a set of Unix processes running on a set of processors: each process is numbered sequentially from zero and it can exchange messages with other processes in the metacomputer. This SPMD message-passing achieves maximum scalability and efficiency for data- and numerical-intensive computations, on homogeneous parallel supercomputers, besides providing reuse of existing SPMD parallel software.

Programs and data can be brought together in new and powerful ways through a point-and-click Java GUI, called Javaflow, as described below. The output of the GUI is a stream of human-readable, text commands. Because the GUI produces a text stream, it means that Javaflow and the GUI is an option, not a necessity.

Computational speed and efficiency is achieved because the real work is done with the proven model of SPMD parallelism, where a service may have a thousand nodes of a parallel machine working together, using optimized message- passing for inter-node communication. Each node program in a service may be multi-threaded, allowing compilers to get the greatest possible speed from the compute node. It may also be that the “node” is in fact a shared- memory multiprocessor (SMP), in which multithreading is the preferred route to parallelism in place of the SPMD model. This multithreaded model is expected to fit especially well with the 256-cpu, shared-memory HP/Convex machine at Caltech, and also to fit well with COTS server solutions.

The data channels between services are explicitly parallel, so that multiple high-bandwidth physical links can be used as a (conceptually) single channel. Different types of channels can be implemented in different ways, depending on circumstances: for example if a service of N processes talks to another service with N processes, we could implement it with N independent connections (for a Round-robin channel), or with a router process and virtual circuits for an incoherent channel.

In Paraflow, services are connected together by channels. Services are computing units, connected by the dataflow paradigm to form a directed acyclic graph. The data that flows between services is assumed to move as packets, or messages, from service to service. In an application, a packet may

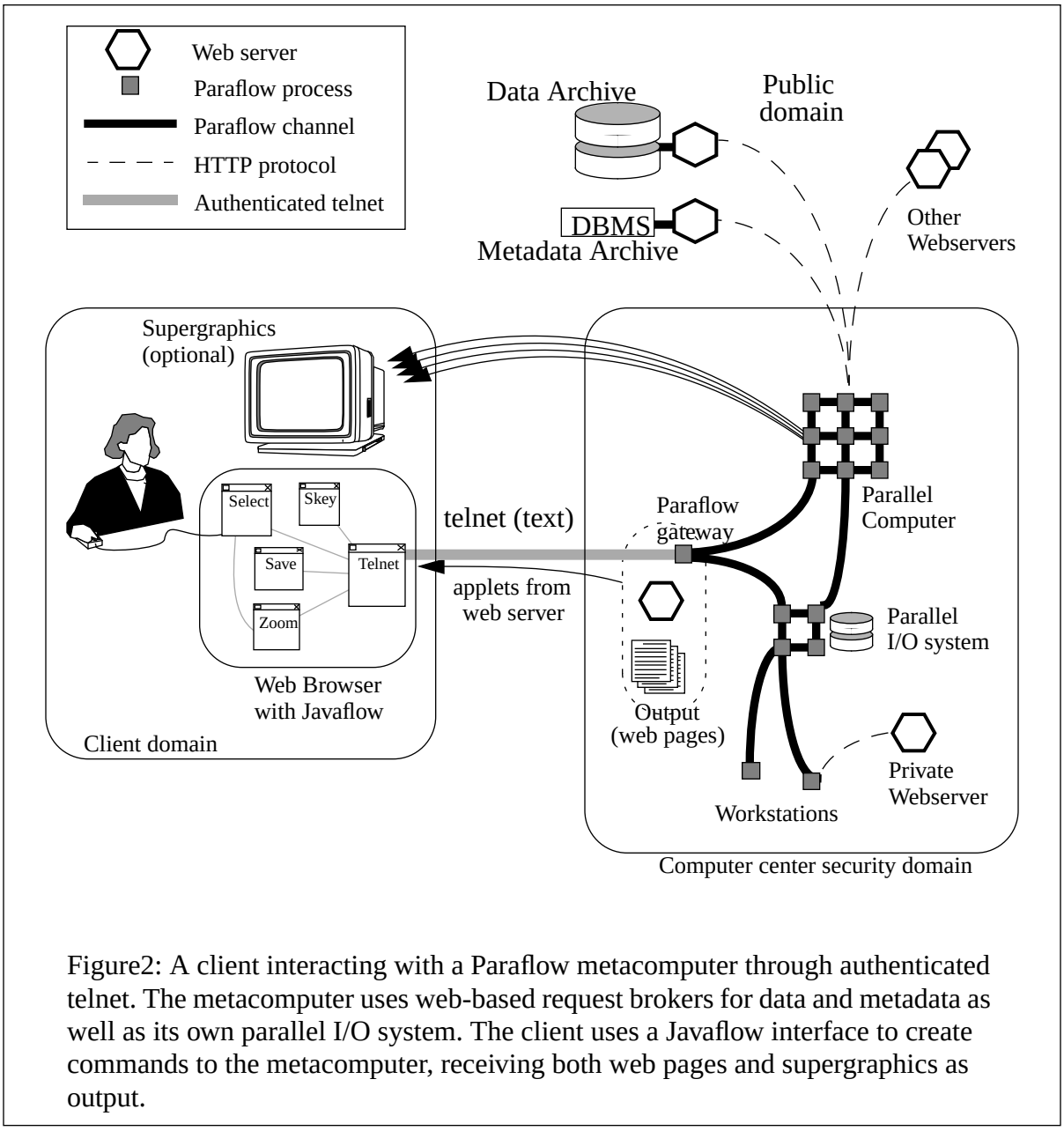

Figure2: A client interacting with a Paraflow metacomputer through authenticated telnet. The metacomputer uses web-based request brokers for data and metadata as well as its own parallel I/O system. The client uses a Javaflow interface to create commands to the metacomputer, receiving both web pages and supergraphics as output.

be a large binary object, or it may be metadata: small forms that are dictionaries of keyword-value pairs. In either case, the packet has a header with information such as packet size and what type the data is.

Services

A service is a barrier-synchronous group of processes communicating with other members of the group by MPI. When a service is initialized, it gives its name, and in return is given an MPI communicator object, enabling message-passing, broadcast, and collective I/O. Other processes that have received this communicator are those with the same service name.

A service can be created by a user by writing C or Fortran or other programming languages, then linking it with a library. This library provides the initiation of the program (main program), then passes control to the user-code; on exit, the user code should pass control back to the library.

Services communicate with other services through the named channels attached to them: a channel is semantically similar to a file-pointer or file descriptor to C programs, and as a logical unit number to Fortran. Blocking, data flow, and progress of the computation are described in the next section. Each service also has a diagnostic output available, which is assumed to contain only small quantities of text, frequently flushed, to communicate with the user of the system: every effort is made to make sure that these diagnostics from the services are interleaved and appear in front of the user, or a log file, with maximum reliability and minimum latency.

Channels

Services can connect to channels by giving the correct name of that channel. Channels may be thought of as carrying data in one direction only, at least for the purposes of creating a dataflow graph of services connected by channels. In fact, however, there is generally a “back-channel” in the opposite direction for small data such as acknowledgments or (in the case of a server channel) the request to the server.

When the sender and receiver processes of a channel are parallel, there are several ways in which packets can be directed from the processes of the sender to those of the receiver, and different strategies for synchronization and blocking. We have chosen to implement the following useful cases: Broadcast, where each packet emitted by sender goes to all receivers, Round-robin, where packets are distributed from parallel service to parallel service deterministically, and Incoherent, where packets are delivered non-deterministically, such as in the Manager-Worker paradigm.

In addition to communicating within a service by MPI messages, and services may communicate with each other by sending Paraflow packets to other services, or they can open URLs on remote web servers to request data and metadata. Results of computation may be written to a file or passed back through the Paraflow gateway as text, and thence to the Javaflow GUI. Files written by the Paraflow services may be accessible to the client via a web browser.

The Paraflow Gateway

The gateway is a unique, serial process. It interacts with the user, or with a conference of users, and may display the results of a computation. The gateway process passes results and diagnostics to the human user, it orchestrates the persistent network, including setup and tear-down, and it can replicate results to many clients to enable Paraflow to be used as a conferencing engine.

If the Paraflow metacomputer is set up from a Java applet client, then the gateway process will reside at the same IP address as a web-server that served the applets.

Implementation

Cluster computing and parallel computing are coming closer than ever before. Let us define a cluster as a heterogeneous collection of processors where each has an IP address, and the data routing is the responsibility of the system administrator who programs the routers. On the other hand, let us define a parallel computer by the presence of a highly-tuned communication backplane connecting a homogeneous collection of nodes. The cluster uses Unix or NT sockets, it

would have fewer nodes than the parallel computer, and would be comfortable with a web-server on each node; in contrast the parallel computer has one or two IP addresses even for a thousand nodes.

A communications fabric for Paraflow must unify these paradigms. One possibility is the MPI standard, which runs on either a cluster (MPICH or LAM-MPI) or on a parallel computer (usually vendor implementation of the MPI standard). One way to do this would be to use IP sockets and MPI, with custom gateway nodes that translate between the two standards. In this project, however, we will leverage from the Nexus [12] protocol, from Caltech and Argonne, that provides a heterogeneous MPI layer, support for threaded processes, and resource discovery and security services from its companion software, Globus [13].

We have implemented the SARA Active Library on various MPI platforms at the Center for Advanced Computing Research (CACR) [14] at the California Institute of Technology, including an IPM SP/2, a HP/Convex Exemplar, a Beowulf “pile of PC’s” machine [15], and a cluster of SunOS workstations. Data is served from the HPSS-powered tape robots at CACR and at the San Diego Supercomputing Center, and from other servers at the University of Lecce, Italy. Metadata is served from other machines at these sites.

Javaflow: Making a GUI from a Text Dialogue

Paraflow and Javaflow are separate software systems which communicate by text-based commands. If desired, human may type into Paraflow or they can use Javaflow to create and send the commands, as well as render the results; on the other hand, Javaflow could be used to control other text-based programs besides Paraflow.

Figure 3 shows some typical applets from a SARA image-processing session, beginning when a user points a web browser to a URL containing the telnet applet, which allows the user to login to the server that provided this applet. If this server uses Skey authentication, another applet (top left) may be useful for computing the one-time password: clicking the “Send” button on this applet, as with all Javaflow applets, causes the indicated text to be inserted into the telnet stream.

A command such as “File/SaveAs/Jpeg image09.jpg” can be created with the menu applet (top right), where the strings shown on the menu are inserted into the telnet stream; the user can then finish the command with a filename. When the command is executed, the Paraflow shell responds that a web page containing this image is available at some URL, which the client can now fetch with another web browser.

Even when controlling Paraflow by the Java interface, the exchange between client and server is still by text, but the resultant image09 can be displayed to the client rather than just stating a URL. This is because the Paraflow output can contain markup tags, of which the following is an example:

<javaflow event=ImageSelector url=http://naegling/image09.jpg>

The telnet client flags such tags, sending the enclosed string to an event listener called ImageSelector, that responds appropriately. In this case, an image has been written to a URL and is to be used in an ImageSelector applet: the one at the lower left of Figure 3. Using this applet, the user selects a rectangular subset of an image and writes the coordinates of the selection as text, which can in turn be used as part of a command to the Paraflow metacomputer. The applet helps with what would be a most frustrating task, and at the same time the interface between the applets

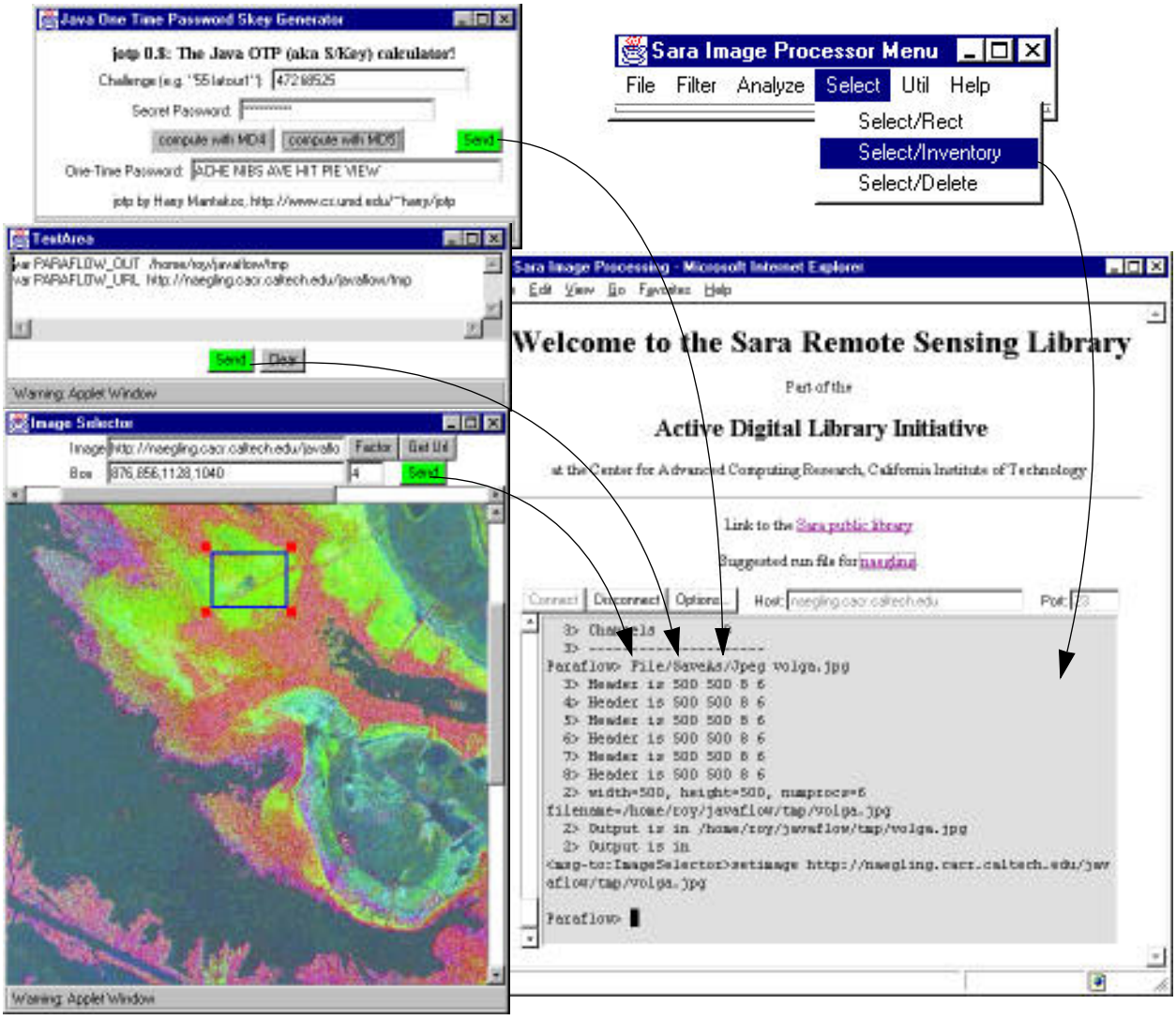

Figure 3: Javaflow applets used in the SARA Active Library. Lower right is the telnet session, where the user may type commands and other input to the computation. Surrounding this applet are applets that may be used to create textual input in an easier way than typing. The image selector (left) provides pixel coordinates for a chosen box.

is so exposed (the text input and output) that users can confidently create their own applets for new or specialized applications.

The SARA Active Library provides facilities to read and write SARA datasets into a parallel computer, and do various processing, such as changing image size, principle component analysis, and supervised classification by selecting known regions. Because of the web focus of the interface, there are pages of help and explanation available whenever a decision is to be made.

Further Work: LIGO

LIGO (Laser Interferometric Gravitational Wave Observatory) is a large project, funded by the

US Government, to detect general-relativistic gravity waves[2]. If successful, LIGO will offer an explicit view of the most cataclysmic events of the Universe, for example a black-hole swallowing a neutron star. The output of the experiment is a multichannel time series, with the principle gravity-wave channel and a number of subsidiary channels such as seismometer, microphone, magnetometer, etc. The computations we have implemented involve optimal Wiener filtering to search for faint signals deep in the noise.

A Javaflow applet with a calendar allows a user to select the time-periods in which data is to be searched; a refinement of this is the ability to specify further criteria on the subsidiary channels to ensure consistent data quality. The template patterns to be found in the data are generated by another Paraflow service, defined by parameters such as the masses of the compact objects (neutron stars or black holes) for which we search; the search is specified by a range of these masses. The main computation is the pattern-matching itself: each section of data is searched for each template, so that parallelization can be done over templates, over data, or both. The efficiency is controlled by the efficiency of the Fourier transform on the computer. Finally, any matches that are found are to be stored in an event database. An attribute of each event is the signal-to-noise ratio (SNR), with a large number of events with small SNR that characterize the noise characteristics of the instrument, and perhaps a few events with large SNR values that represent cosmic collisions.

The SARA and LIGO libraries have some similarity in that they are indexed by real coordinates; in the former, the coordinates are longitude and latitude, and in the latter the single coordinate is time. The value returned at a point in the coordinate space is a collection of named channels. In both cases, the data servers supply sub-blocks of these cartesian arrays of data, and the metadata servers convert coordinates into filenames that the data servers can retrieve. Other examples of the use of multichannel cartesian arrays might be storage of volume images, such as electron-density maps or brain scans.

Conclusion

As supercomputers and data systems gain in speed, complexity, and heterogeneity, we must strive to maintain usability. In some cases the computational part of a scientific endeavor is of such magnitude and importance that people will spend a great deal of effort to learn the details, but many potential benefactors of HPCC technology do not wish to do this. Our objective in this project is to provide the fastest possible route to the data and computing that the client wants, together with the scope to become arbitrarily sophisticated in the use of that data. With the same basic architecture, a schoolchild can get something interesting from a web page and also the professional can do custom processing of a terabyte of data.

References

1. Radio Telescope
http://www.cacr.caltech.edu/SIO/APPL/phy02.html

2. LIGO, Laser Interferometric Gravitational Wave Observatory
http://www.ligo.caltech.edu

3. The National Center for Microscopy and Imaging Research
http://www-ncmir.ucsd.edu/

4. Data mining in cosmology simulations
http://www.cacr.caltech.edu/~johns/pubs

5. Data mining in neurobiology simulations
http://www.bbb.caltech.edu/hbp/database.html

6. Infobus project, federating digital libraries,
http://www-diglib.stanford.edu/

7. Paraflow, Heterogeneous data computing
http://www.cacr.caltech.edu/~roy/paraflow

8. SARA, the Synthetic Aperture Radar Atlas, is a collaboration of Caltech, SDSC, and the University of Lecce, Italy.
http://www.cacr.caltech.edu/sara
http://sara.sdsc.edu
http://sara.unile.it/sara

9. MPI, The Message-Passing Interface
http://www.mcs.anl.gov/mpi

10. Java Telnet client from University of Oregon
http://www.nacse.org/web/webterm/

11. S/key One-time password calculator
http://www.cs.umd.edu/~harry/jotp

12. Nexus, heterogeneous computing fabric
http://www.mcs.anl.gov/nexus/

13. Globus, Computing resource discovery services
http://www.mcs.anl.gov/nexus/

14. Center for Advanced Computing Research, Caltech
http://www.cacr.caltech.edu

15. Beowulf project, cheap supercomputing with clusters of PC’s
http://www.cacr.caltech.edu/research/beowulf