Net-dbx-G: A Web-Based Debugger of MPI Programs Over Grid Environments

Panayiotis Neophytou[1] Neophytos Neophytou[2] Paraskevas Evripidou[1]

[1]*Department of Computer Science*
University of Cyprus
P.O. Box 537
CY-1678 Nicosia, Cyprus
{cs99pn1,skevos}@ucy.ac.cy

[2]*Computer Science Department*
Stony Brook University
Stony Brook, NY 11794-4400
nneophyt@cs.sunysb.edu

Abstract

Net-dbx-G is a tool that utilizes Java and other World Wide Web tools as an interface to Grid services to help Grid application developers debug their MPI programs from anywhere in the Internet. Net-dbx-G is a source level debugger with the full power of gdb (the GNU Debugger), providing the user with full Grid functionality. The portability of the tool is of great importance as well because it enables the tool to be used on heterogeneous nodes that participate in an MPI enabled Grid environment. The users of our system simply point their browser to the Net-dbx-G webpage and authenticates to the Virtual Organization that they are part of. Having at their disposal the shared resources participating to that VO, they start debugging by interacting with the provided GUI environment of the tool. The users can dynamically select which MPI processes to view/debug.

1. Introduction

This paper presents Net-dbx-G, a tool that uses World Wide Web (WWW) capabilities, in general, and Java applets, in particular, for portable parallel and distributed debugging across an MPI enabled Grid environment. It takes advantage of Java's portability, to be used virtually from any Java-enabled web browser. Net-dbx-G is now a working prototype of a fully-fledged development and debugging tool for the Grid. This new tool is based on the debugging functionality of Net-dbx [12, 13].

Net-dbx's approach to achieving distributed debugging is based on individually attaching every process to a local debugger at the lowest level and then integrating the individual debuggers into a device-independent, interactive, user-friendly environment [13]. For each process to be monitored, the integration environment interacts with the local debugger. As the user defines global and individual operations to be applied to all or some of the processes, these are translated by the integration tool into interaction with each of the local debuggers individually. To attach all the required processes to the local debuggers, an initialization scheme has been implemented [12]. The overall architecture of Net-dbx is based on a layered design: the lower layer, which resides on the MPI-Nodes, the communications layer, which is implemented on the client side, and the integration layer, which coordinates the communication objects and provides the graphical environment to the user. Many changes have been made to the initialization scheme, the communications layer as well as to the user interface.

Grid environments, being developed today, enable the sharing of resources as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. Resource provider and resource consumers define clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what is called a *virtual organization* (VO) [4]. To be part of a VO and be able to use resources shared in that VO you have to obey the rules set by the VO [8]. This is a requirement of our tool since it enables users to use resources shared at any VO. To satisfy this requirement we had to extend the security layer of Net-dbx, in order to provide the user with the correct

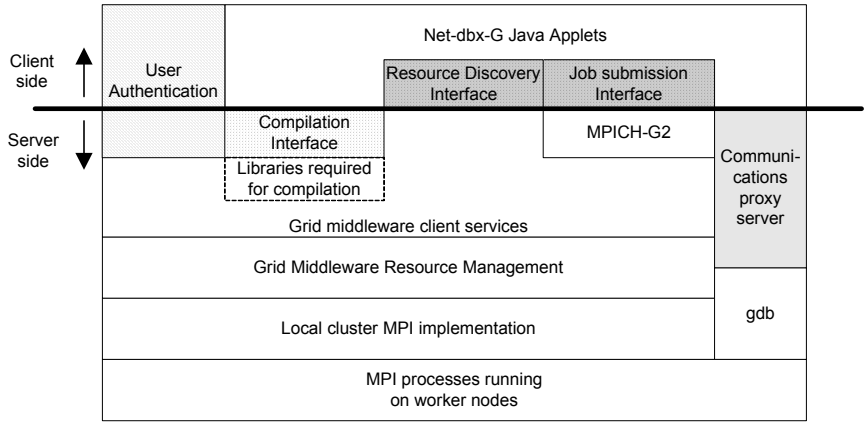

Figure 1. The layered architecture of Net-dbx-G. Striped is the security layer. In gray are the Globus interfacing layers and in light gray is the communications layer.

credentials that will allow him to use the resources he is entitled to use.

The prototype described in this paper is a runtime; source-level debugging tool for MPI enabled Globus Grid environments. It combines the capabilities of the gdb debugger [15], the functionality offered by the available vendor-dependent MPI implementation environments, and the resource sharing, monitoring, and management services provided by the Globus Toolkit [4]. It acts as a debugging user portal to Globus Toolkit enabled Grids. Using our proposed modular architecture, it is easy to provide support for additional Grid middlewares by adapting specific modules of our tool that reside in the abstraction layers. This is accomplished by implementing Java Interfaces that are specified in our design. New Java classes have to be created to implement the methods defined the Java Interfaces we have designed. These Java Interfaces provide the necessary abstractions needed between the User Interface and the Grid services and tools.

In the rest of the paper we will describe how this tool works and how it is designed to provide an interface to Globus Toolkit services. In Section 2 we briefly describe the supporting tools used to base our implementation. In Section 3 we provide a detailed description of our architecture by giving the outlines of the architecture's layers. In Section 4 we give the implementation details of our prototype. Following, is a brief retrospection of current tools that are currently available for similar tasks and compare them to Net-dbx-G.

2. Supporting Tools

2.1 Globus Toolkit 2.4

The Globus Toolkit is a collection of software components designed to support the development of applications for high-performance distributed computing environments, or "Grids" [6, 5]. Core components typically define a protocol for interacting with a remote resource, plus an application program interface (API) used to invoke that protocol. Higher-level libraries, services, tools, and applications use core services to implement more complex global functionality. We use Globus enabled resources as MPI enabled clusters, where one node of it is the head and the rest are the worker nodes. The head is the resource manager that controls access and usage upon the worker nodes. These clusters must be MPI enabled with any vendor MPI implementation available. The various Globus Toolkit components are reviewed in [4] and described in detail in online documentation and technical papers.

2.2 MPICH-G2

MPICH-G2 [11] is a complete implementation of the MPI-1 standard that uses Globus Toolkit services to support efficient and transparent execution in heterogeneous Grid environments, while also allowing for application management of heterogeneity.

We use MPICH-G2 libraries to compile and run the MPI applications being debugged. MPICH-G2 enables us to utilize resources spanning multiple sites to run a user's mpi application.

2.3 Gdb

Gdb [15] is a free GNU debugging tool that appears to be implemented on most of the major UNIX platforms. The functionality of gdb includes attaching to processes at runtime, changing variable values, and

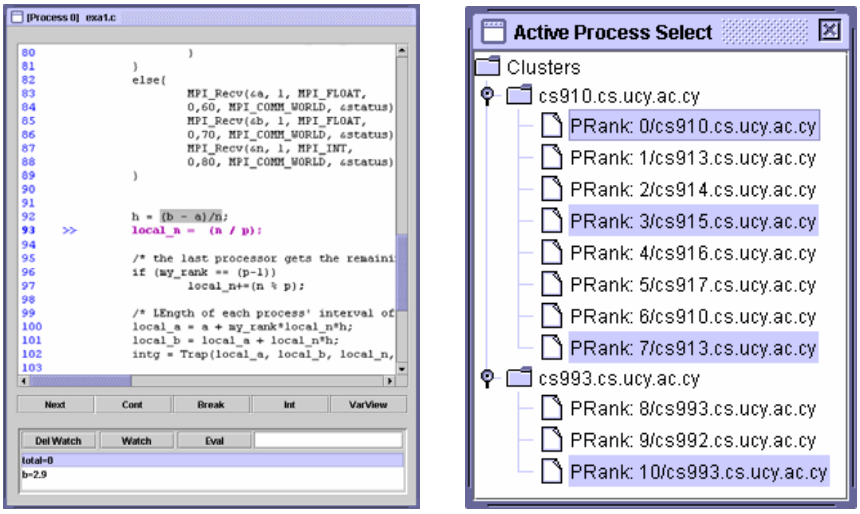

Figure 2. (a) A screenshot of the Debugging Session Window where the user can set breakpoints, add variable watches and control individual processes. (b) The Active Process Select Window where the user can choose processes to open and control through the Debug Session Window.

setting conditional breakpoints based on expressions.

3. Architecture

Net-dbx-G is based on the same principles of layered design as its predecessor Net-dbx. In order to make this new tool portable, the previous design has been enriched with basic components and methods that are required for compatibility with the supporting tools. An illustration of the layered architecture is shown in Figure 1.

At the lowest level, our implementation relies on the vendor-dependent MPI implementation and local debugging tools on each node. These will then communicate with higher communication layers, and at the topmost part everything is integrated into the debugging GUI applet, which is hosted at the remote user's web browser.

An initialisation scheme is required in order to locate and attach all the participating processes to the local debuggers, and establish a communication link between those processes and the debugging applet.

At the top layer, which resides at the user's web browser, we have the debugging user interface and all the functionality that integrates all the components from the lower layers of the system. Two of these components, Debugging Window and Active Process Select are shown in Figure 2.

A user-friendly wizard (shown in Figure 3) is available upon initialization to help the user go through the process of setting up his/hers Grid debugging environment. With the help of the wizard the user can choose the resources that he needs and the executable of his application. Then after confirming the input data the wizard submits the job to Globus and then the debugging process is ready to get started.

The interactive remote control of process in a Globus toolkit enabled environment presents quite a few challenges. An inherent limitation of Grid architectures is that the remote user cannot directly connect to the grid nodes, but rather submit jobs that are going to be executed asynchronously. In order to perform real-time debugging on these parallel processes, one needs to have direct access to the participating nodes, so a proxy workaround had to be provided by our architecture. In the following sections we will describe the new architecture of Net-dbx-G, which was designed to overcome these problems. Particular focus will be given to the communications and the Globus-enabled modules of the system.

3.1 Lower Layer – Providing access to remote processes

The lower layer consists of the server-side services of the Globus Toolkit 2.4, the Grid enabled MPI implementation (MPICH-G2), the vendor-dependent MPI implementation on each cluster and debugging tools (currently supported is gdb). All of these tools rely on each node of the clusters participating in a Virtual Organization.

In order to use the underlying local tools to start and maintain a debugging session, an *Initialization Scheme* is needed. To be able to attach a process for

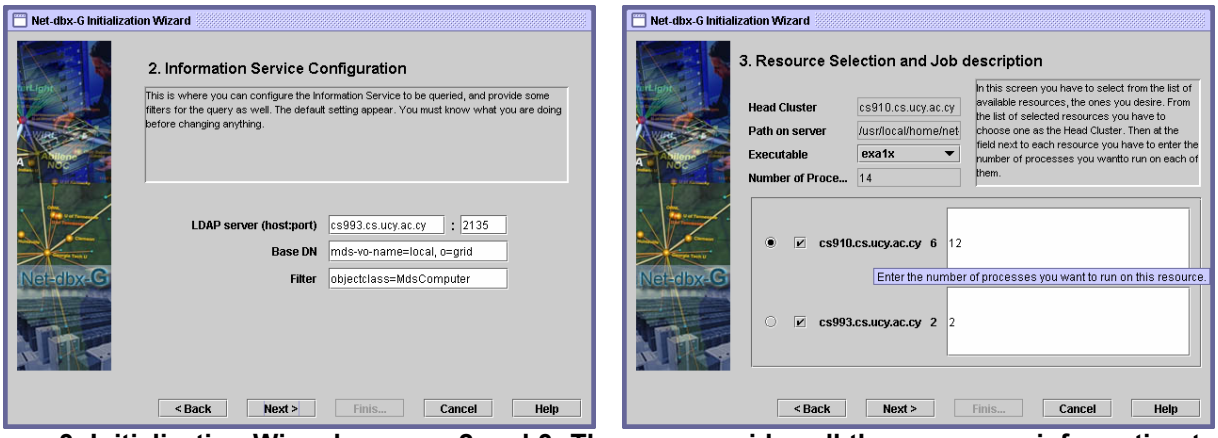

Figure 3. Initialization Wizard screens 2 and 3: The user provides all the necessary information to the wizard in order to submit the job and start the debugging process. (a) Information Service Configuration, (b) Resource Selection and Job description.

debugging, the system requires the PID and the node where it is running. This data can be acquired from the process only at runtime. In order to be able to attach the processes to the debugger before they terminate, the system must stall them until they are all "captured". A small piece of code has to report each process' PID and IP address and stall them until the "interesting" ones (the ones that the user wants to debug) are attached to the debugger. This is done by intercepting the call to MPI_Init and linking to the Net-dbx-G instrumentation library during compilation. This is done transparently to the programmer when they use the Net-dbx-G aware scripts. The above scheme works uniformly on FORTRAN, C, and C++ programs using any vendor-dependent MPI implementation.

3.2 Communications Layer

The communication between the nodes on the server side and the *integration tool* on the client side is managed by the DebugSession Java Objects. These objects have integrated basic communication functionality for doing SSH connections and are also enriched with debug functionality, inherited from the predecessor of our tool, Net-dbx. These objects provide the necessary abstractions needed to individually debug their corresponding processes. Methods for synchronizing with the other DebugSessions are also provided in order to initiate the debugging process or perform global operations. The debugging functionality and commands of the predecessor Net-dbx are used in Net-dbx-G.

In Net-dbx, the commands were propagated to the local debuggers through simple telnet/SSH sessions. However, the Globus Toolkit does not allow external users to login directly to the resources, but rather submit computational jobs asynchronously. We found this issue particularly challenging in the process of developing our tool, because of the high interactivity needs of the debugger. Globus' build-in I/O handling uses buffers. This makes interactive communications difficult to handle. To overcome this problem we have developed a proxy server that runs on the Net-dbx-G hosting web-server. This proxy server accepts connections from MPIHost (nodes that run processes) and from Net-dbx-G client applets. MPIHost objects send a label to the proxy server in the format "MPIHOST hostname rank" and client objects send "CLIENT hostname rank" labels to indicate the process they want to connect to. The proxy server matches these labels and binds the sockets to each other by forwarding the input and the output of these sockets. The main reason that we are using the proxy server is because one of the major security constraints posed in Java is the rule that applets can have an Internet connection (of any kind – telnet, FTP, HTTP, TCP, etc) only with their HTTP server host [9].

3.3 Security Layer

There are mainly two security issues raised by Net-dbx-G's architecture. The first is user authentication. A user of Net-dbx-G is automatically considered a user of the whole Virtual Organization because he is going to submit jobs on the resources. As such he must be authenticated with the correct credentials required by the VO [8]. So far Net-dbx authenticates the user using a username and password given at login time and by comparing that with a list of usernames and passwords in a database on the server. The previous authentication mechanisms were revised and a security layer has been added to the architecture (Figure 1:

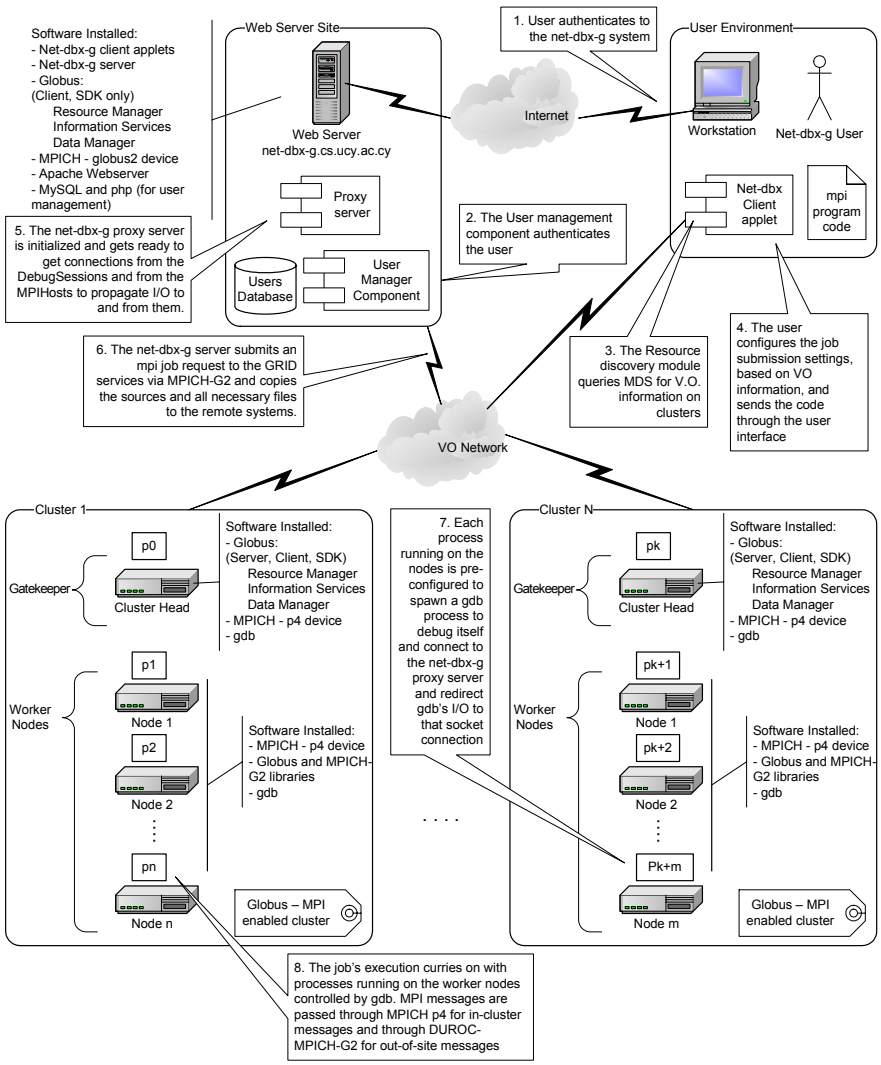

Figure 4. Initialization and Running Scheme of Net-dbx-G.

shown in stripes is the Security Layer (User Authentication)) so the implementers of the architecture can adapt the debugger's methods to the ones required by the VO. The Globus Toolkit uses certificates to authenticate the users. A certificates proxy provides the user with the necessary credentials for single sign-on. The user can then use them to run jobs on shared resources. Our tool provides this interface to user in order to be able to run jobs on desired resources.

The second aspect of security that was considered is the communication between the nodes, the client and the proxy server. The use of secure sockets [14] is required to prevent interception of the messages by malicious intruders who could manipulate the command stream to the debugger and compromise the target machine thereafter.

3.4 Configurations Layer

The configurations layer manages the parsing of the user options into the system. The main module of the configuration layers (Figure 1, Layers in dark gray) is the Initialization Wizard. The wizard component is the link between the user and the abstraction layers. Some Java Interfaces were also designed to be used in the implementation of the Wizard's screens, in order to provide the correct interface to the Grid Services. These interfaces act as the necessary bridge to the Globus Toolkit services. With the help of the Wizard and the implemented classes the user first defines the Information service to be queried (Figure 3 (a)). In our case the information service is MDS-2 [2]. MDS-2 uses GRIS services to store information about local resources and GIIS service to store aggregate information on a set of VO resources. The user can

either query GRIS or GIIS services. The results from these LDAP servers are parsed by the Resource Discovery module and are then displayed to the user. The resources are displayed as illustrated in Figure 3(b). Each resource manager with the number of nodes that comprise it and a text field next to it so the user can enter the number of processes to run on each cluster is displayed in a table. The user may also choose on which cluster to run the first process. This is the Head Cluster. On the same screen the user also pick the executable file located on the server.

After the data entered is verified, they are passed to the Job Handler module and an RSL script is generated and displayed to the user. The RSL [1] script contains the default options for running the job. The user is able to make his own changes to the RSL script and save them. The changes are then parsed to validate the script and with the user's confirmation the job is passed to the Integration layer for submission. After the job submission the user is then prompted to choose the processes he wishes to attach to the debugger. Although the presence of queues on various resources makes the application halt until the job is initiated and started, the debugger's instrumentation library notifies the proxy server upon initialization of all the processes. Then the proxy server in turn notifies the user that his job was submitted and that he/she is able to debug the application.

At this point, the system is ready to proceed to the initialization procedure that will attach all the selected processes and prepare the session for debugging.

3.5 Initialization Scheme

The initialization scheme used in Net-dbx is also preserved in Net-dbx-G. The participating processes' PIDs are used to attach the debugger. The synchronization scheme ensures that all the processes of interest to the user are attached to the local debuggers right after the MPI initialization calls. Net-dbx-G extends this scheme to meet the requirements of our grid-enabled architecture that was designed to overcome the heterogeneity-related difficulties encountered in a grid environment. Figure 4 shows the exact order of the initialization tasks starting from the point where the user chooses runtime parameters for his application, to the point that all the processes are fully attached and ready to be debugged using the user interface. The initialization procedure starts from the point where the user chooses runtime parameters for his application, through the initialization wizard, to the point where all processes are fully attached and ready to be debugged using the user interface.

At first, the system opens an ssh session with the debugger server (which is the web server) and authenticates as the net-dbx user. The ssh session provides a command line interface towards the client services of the grid. It also provides a way to handle the application's standard input and output after the job submission. Using the ssh session, the net-dbx client checks to ensure the availability of the proxy server. Once the proxy is available, the user preferences are passed to the Job Submission module. Then this module can submit the job directly to the grid services or indirectly by using the ssh session and command line tools. After job submission the processes begin execution on the nodes. The first call the processes make goes to the net-dbx module added at compilation time. Process 0 gathers all the information regarding the PIDs, the hostnames and ranks of all the processes and sends them to the proxy server, which in turn prints them to the ssh session for the client to collect. Then, Process 0, which is responsible for handling program's I/O, connects to the proxy server and redirects the program's standard input and output to the secure socket connection. The proxy server prints the program's output to the ssh session and gets the program's input from the ssh session, which is now available to the user. After that, all processes call fork() and clone themselves. The clone connects to the proxy server and immediately replaces itself with a gdb process. Then gdb's I/O is redirected to the connection. In the meanwhile, on the client side, one object is created for each process to control it (based on the information about the execution environment which includes PIDs, ranks and hostnames). Each of these objects connects to the proxy server and requests a connection with the appropriate gdb socket. If the proxy server has the requested socket it binds them together, if not it waits until the appropriate gdb connection occurs. After binding all the "interesting" (user selected) processes the program is synchronized at the point where all the processes left the initialization call, and it is ready for debugging.

4. Implementation Status

All the functionality described above is already implemented and tested in a prototype version of Net-dbx-G which is hosted at the computing facilities at the University of Cyprus. It has been tested by local users from project groups working on Grid application development. The system is highly interactive, and the client applet has been tested to work in all available

browsers at our facilities, as well as remotely from testers overseas.

The lower-layer, server components are currently running on our local test-bed that consists of 2 clusters. One with 6 dual processor Fujitsu Super Servers and one with 2 single processor IBM eServers. The clusters are running on Globus Toolkit 2.4 and MPICH-G2. For in-cluster message passing, MPICH with p4 device is used.

A web application hosting environment was implemented using PHP server side scripting language. It provides users with an interface to upload their files to Net-dbx-G's alpha local system and then to compile, view, delete etc. To be able to do this, we grant users with a user password on the Net-dbx-G system which, with the help of the Web Server, controls access to the corresponding PHP scripts. Every user of the PHP script system is entitled to a local directory on which his/her files will be uploaded and manipulated. When the user first accesses one of the Net-dbx-G utility scripts, he is asked for his/her user ID and password.

When the user invokes the Net-dbx-G applet, the system knows which compiled files are to be made available to the user to run using the debugger.

5. Related Work

There are two commercially available distributed debuggers of note. TotalView [3], from Etnus, is a third party debugger that runs on a number of high performance computing platforms. It is currently not capable of debugging in heterogeneous environments. Prism [16], from Sun Microsystems is derived from the Thinking Machines product of the same name. It is not portable to systems other than Sun. While its user interface led the way in scalability, it too, can benefit from more abstraction.

A project used to debug Grid applications running on the Globus Toolkit, is p2d2 debugger [10] developed at the NASA Ames Research Center. It is a debugger for heterogeneous, distributed programs. It relies on the existence of a debugging server on each of the processing nodes. The debugging server provides abstractions according to a predefined framework and it can be attached to the graphical environment at runtime. Following this architecture, the tool achieves portability and heterogeneity on the client side, but relies on the implementation of the server on the server side. Net-dbx-G achieves the same results as p2d2 using a much simpler approach. The server side of Net-dbx-G is considered to be just the set of Grid services, MPI tools, and the gdb debugger on the processing nodes. Providing support for additional platforms does not require the implementation of a new debugging server.

6. Concluding Remarks and Future Work

We have developed, Net-dbx-G a Grid enabled, portable, web-based, source-level MPI debugger. The tool can be used as a web debugging portal for any VO running Globus and MPICH-G2. Our architecture enables full interactivity with the application at runtime. It enables the user to debug his application from anywhere. The Configurations Layer enables the user to choose the resources he wants to utilize and automatically produce the right RSL scripts without knowledge of the specifics of the Globus RSL language. The graphical interface also provides an image of the running environment by grouping the running processes in clusters, so that the user can have some topology awareness. The architecture can scale to match the size of the grid used. The user has the ability to select which process to view/debug and thus can keep them to a manageable number. Of course at any time he can select any process to view/debug.

Our future plans include providing more runtime measurements of the performance of the running application. This will help the user make his applications topology aware. We are also working on extending our architecture to support the new Open Grid Services Architecture [7] and Globus Toolkit 3 as soon as MPI is supported on this platform.

7. References

[1] Karl Czajkowski, Ian T. Foster, Nicholas T. Karonis, Carl Kesselman, Stuart Martin, Warren Smith, Steven Tuecke: A Resource Management Architecture for Metacomputing Systems. JSSPP 1998: 62-82.

[2] Karl Czajkowski, Carl Kesselman, Steven Fitzgerald, Ian T. Foster: Grid Information Services for Distributed Resource Sharing. HPDC 2001: 181-194.

[3] Etnus, Inc. The TotalView Multiprocess Debugger. http://www.etnus.com/products/totalview/

[4] Ian T. Foster: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Euro-Par 2001: 1-4.

[5] Ian T. Foster, Carl Kesselman, editors. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1999.

[6] Ian T. Foster, Carl Kesselman: The Globus Project: A Status Report. Heterogeneous Computing Workshop 1998: 4-18.

[7] Ian T. Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke: Grid Services for Distributed System Integration. IEEE Computer 35(6): 37-46 (2002)

[8] Ian T. Foster, Carl Kesselman, Gene Tsudik, Steven Tuecke: A Security Architecture for Computational Grids. ACM Conference on Computer and Communications Security 1998: 83-92.

[9] J. Steven Fritzinger, Marianne Mueller: Java Security Whitepaper, Sun Microsystems Inc., 1996.

[10] Robert Hood, Gabriele Jost: A Debugger for Computational Grid Applications. Heterogeneous Computing Workshop 2000: 262-270.

[11] Nicholas T. Karonis, Brian R. Toonen, Ian T. Foster: MPICH-G2: A Grid-enabled implementation of the Message Passing Interface. Journal of Parallel and Distributed Computing 63(5): 551-563 (2003).

[12] Neophytos Neophytou, Paraskevas Evripidou: Net-dbx: A Web-Based Debugger of MPI Programs Over Low-Bandwidth Lines. IEEE Trans. Parallel Distrib. Syst. 12(9): 986-995 (2001).

[13] Neophytos Neophytou, Paraskevas Evripidou: Net-dbx: A Java Powered Tool for Interactive Debugging of MPI Programs Across the Internet. Euro-Par 1998: 181-189.

[14] "Secure Sockets Layer", http://wp.netscape.com/security/techbriefs/ssl.html, 2003.

[15] Richard M. Stallman, Roland Pesch, Stan Shebs, et al., Debugging with GDB:The GNU Source-Level Debugger, Ninth Edition, for GDB version 5.1.1, Free Software Foundation.

[16] Thinking Machines Corporation. Prism User's Guide. Thinking Machines Corporation, Cambridge, MA, Dec. 1991; also: http://www.sun.com/servers/hpc/software/configuration.html#prism