

Debugging MPI Grid applications using Net-dbx

Panayiotis Neophytou¹, Neophytos Neophytou², and Paraskevas Evripidou¹

¹Department of Computer Science
University of Cyprus
P.O. Box 537
CY-1678 Nicosia, Cyprus
{cs99pn1,skevos}@ucy.ac.cy
²Computer Science Department
Stony Brook University
Stony Brook, NY 11794-4400
nneophyt@cs.sunysb.edu

Abstract. Problem solving using grid computing environments has become very popular amongst research groups in computation-demanding fields. This is due to the ability of Grid technologies and middleware to enable large-scale resource sharing. Application-development in such environments is a challenging process, thus the need for grid enabled development tools is also one that has to be fulfilled. In our work we describe the development of a Grid Interface for the Net-dbx parallel debugger, that can be used to debug MPI grid applications. Net-dbx is a web-based debugger enabling users to use it for debugging from anywhere in the Internet. The proposed debugging architecture is platform independent, because it uses Java, and it is accessible from anywhere, anytime because it is web based. Our architecture provides an abstraction layer between the debugger and the grid middleware and MPI implementation used. This makes the debugger easily adaptable to different middlewares. The grid-enabled architecture of our debugger carries the portability and usability advantages of Net-dbx on which we have based our design. A prototype has been developed and tested.

1 Introduction

The rapidly growing demand on computational and storage resources has made the development of Grids of various kinds and sizes, of different scopes and objectives, a common yet difficult task. A lot of different factors need to be considered in the development and deployment of a Grid, the most important being the application domain on which it is going to be used. The requirements of a Grid are provided by the set of people that are resource providers and resource consumers that share common interests. These sets of people are what are called Virtual Organizations (VO) [3]. So far a lot of Grids have been developed with different uses to serve a wide variety of researchers as well as business corporations. In this process, developers of such distributed systems for large-scale research have formed a community that has developed good quality middleware (Globus, EDG, etc.) that

allows resource providers to interconnect their resources and efficiently and securely share them with other users, to solve common problems.

The next task is the development of the Grid applications that lie atop of the middleware's infrastructure. These applications make the Grid accessible and of service to its users. While the analysis and design of such applications is quite an important part of the process, their implementation is the most challenging part of this development cycle. These large-scale, resource demanding applications may have their data and processing widely distributed across a Grid system. Powerful tools and IDEs (Integrated Development Environments), specifically debugging tools are becoming even more necessary. In this work we have developed a grid-aware debugger targeted to the Message Passing Interface (MPI) [12], a well-known low-level parallel programming paradigm that is now being used as an additional alternative model for grid programming.

In this paper we describe our architecture for debugging Grid applications. Our architecture relies on an MPI enabled grid development and runtime environment, a Grid enabled MPI implementation, and an existing debugging tool (Net-dbx) that enables debugging of MPI applications across the internet. We have developed an architecture that addresses the heterogeneity associated with Grid environments, with the help of a communications proxy server and the use of abstraction layers. We call this architecture Net-dbx Grid Interface (Net-dbx GI). We have tested this architecture on our local test bed using the Globus Toolkit 2.4, MPICH-G2 and Net-dbx. Support for other Grid enabled MPI implementations, such as PACX-MPI and LAM-MPI will be tested in the near future.

Net-dbx [14] utilizes WWW capabilities in general and Java applets in particular for portable, parallel and distributed runtime source-level debugging across the Internet. It can be used with MPI programs written in C, C++ and FORTRAN. Its design is based on a framework that utilizes gdb [16] to attach the application processes on the individual nodes, and the additional facilities provided by the underlying MPI implementation to provide a complete picture of the topology and communication aspects of the debugged application. Net-dbx GI extends this existing architecture and lays the necessary interfaces for proper initialization, communication and collaboration of the resources needed to participate in the debugging process for applications deployed on the Grid. Other similar architectures to Net-dbx are TotalView [2], p2d2 [9, 1], Panorama [11] and others.

In the rest of this paper, we will show the improvements and changes made to the existing Net-dbx architecture. In Section 2 we present the Net-dbx debugging tool. In section 3 of the paper we describe the layered architecture and the interfaces which are needed to adapt Net-dbx GI architecture to any Grid MPI environment. In section 4 we describe the testing procedure of the architecture using the Globus Project [4] middleware and MPICH-G2 [10] MPI platform.

2 Net-dbx

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 as described in [14]. The overall architecture of Net-dbx is based on a three-layer 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.

The communications and integration layers, which rely on the client side, are implemented in Java. As a Java applet, the system is capable of running in the user's Internet Browser display. Having the requirement for a Java-enabled browser as its only prerequisite, the Net-dbx applet can be operated uniformly from anywhere on the Internet using virtually any client console.

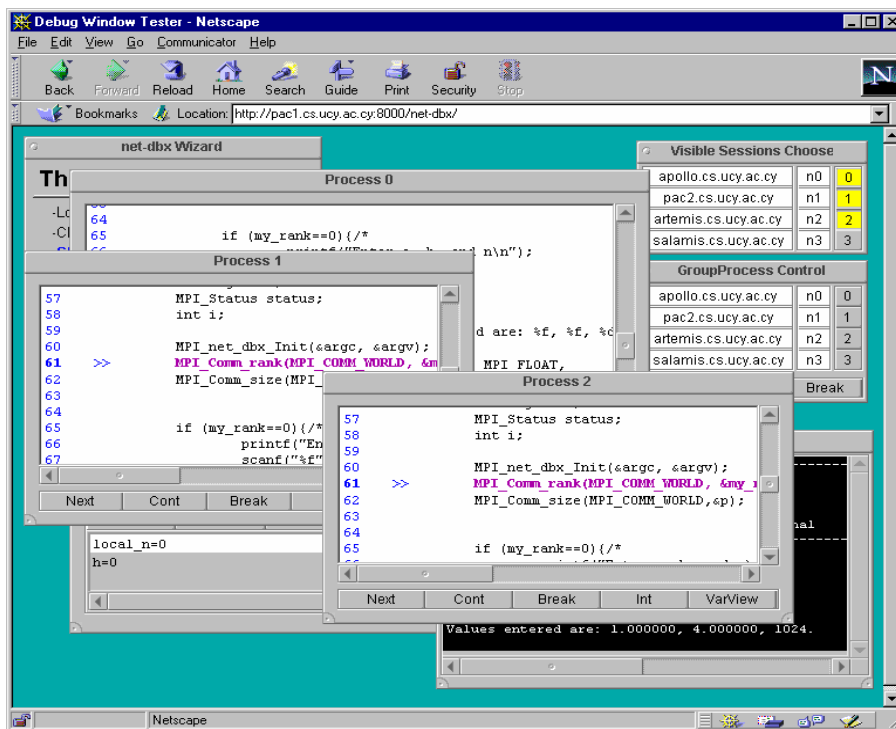


Fig. 1. Net-dbx User Interface at debugging time.

Net-dbx gives you the ability to select which processes to debug. In the example shown in Figure 1 we have a multicomputer that consists of 4 nodes, with a process running on each one of them. The visual components available to the user at debugging time are also shown in Figure 1. All the components are controlled by a

process coordinator. When the user chooses a process from the Visible Sessions Choose window the coordinator opens the corresponding process window visualizing that particular process. The user can open as many process windows as the number of processes running. Through these windows the user can set breakpoints, individually control the progress of a process etc. The user also has the choice of controlling groups of processes from the Group Process control. The complete functionality of Net-dbx is described in detail in [14].

We have chosen Net-dbx over any other of the tools mentioned in the Introduction because of some important advantages it has for usage on Grid environments. Firstly, it is web based, which makes it portable and available from anywhere. Secondly, it is supported by low-bandwidth connections (as low as 33Kbps) which makes it ideal for distant debugging through anywhere in the internet in contrast with the other debuggers that are designed mostly for local use.

3 Architecture of Net-dbx Grid Interface

The Net-dbx GI architecture is based on the same principles of layered design as Net-dbx. In order to make the new architecture portable, the previous design has been enriched with basic components and methods that are required to support most Grid implementations. An initialization scheme is required in order to be able to locate and attach all the participating processes to local debuggers, and establish a communication link between those processes and the debugging applet. This is particularly challenging in Grid implementations, as the inherent architecture of the grid does not allow external users to login directly into the Grid nodes, but rather submit computational jobs asynchronously. The initialization scheme and communications layers have been tailored to address these constraints, and establish direct links to all participating nodes, using an intermediate proxy server.

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 abstraction layers, and at the topmost part everything is integrated into the debugging GUI applet, which is hosted at the remote user's web browser. In the following section we describe Net-dbx GI architecture, with particular focus on the communications and middleware interface layers that enable this tool to operate on Grid implementations.

3.1 Layered Architecture

Net-dbx GI architecture is depicted in Figure 2. Following the Net-dbx architecture, the lower layers are defined as the set of resources that all together with the help of Grid services collaborate and constitute the Virtual Organization's grid network, which with the help of the local vendor-MPI implementation and a grid enabled MPI implementation work as a large MPI multi-computer. It consists of simple tools that are required on each Node and can be used for individually debugging the processes running on that Node. These tools include the local vendor-MPI runtime environment. A grid enabled MPI implementation will help in the communication of the

participating Nodes that are geographically distributed. Also included on each node are the Grid middleware tools, and a source level runtime debugger (we currently use gdb).

On the higher layers there is a client tool that integrates debugging of each Node and provides the user with a wider view of the MPI program, which runs on the selected resources on the Grid environment. The program residing in the client side is the Net-dbx Java applet enriched with a set of interfaces to Grid services. It is used to integrate the capabilities of the tools, which rely on the MPI-Nodes. Net-dbx GI architecture also includes three interfaces to the Grid services and a communications proxy server to help with the quick and easy propagation of messages. We will explain the usage of the interfaces in section 3.2 and the usage of the proxy server in section 3.3.

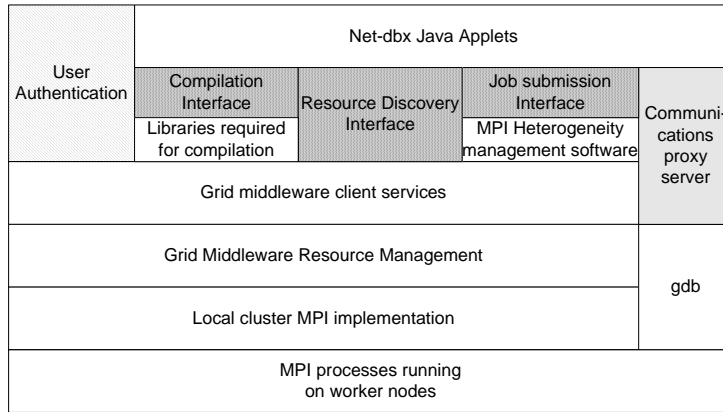


Fig. 2. Net-dbx Grid Interface Architecture

3.2 Abstraction Layers

Our architecture provides abstraction layers as interfaces that can be used with any standard Grid middleware and grid enabled MPI implementation. To achieve this we had to change the way Net-dbx worked to add more abstraction in the initial setup and initialization scheme as well as in the way Net-dbx communicated with the individual nodes.

The first of the three interfaces shown in Figure 2 is the Compilation Interface. This component is responsible for the correct compilation of the user’s source code so that the executable can run on the desired environment. The operation of this component is illustrated in more detail in Figure 3. This module adds a call to our instrumentation library in the user program. This call enables the synchronization of all participating processes and makes them available to be attached for debugging. This initialization process will be described later in more detail. The instrumentation and all the functionality related to Net-dbx is only invoked when the program is run from the debugger, with a special command line argument.

The second interface is the Resource Discovery Interface (Figure 4). This interface consists of two main components. The Information Services Client module is responsible for connecting with the Grid Middleware Information Services component and retrieves information about available resources, and cluster configuration. The output of this component is an XML file containing cluster information about the available resources, depending on the queries made by this component. This component must be implemented in order to enable the retrieval of such information as available resources on which someone can run MPI applications and be able to debug them using gdb. The second component is the Resource configuration module that interprets the XML file and presents the results to the user so that he can make his own choices about the environment on which his application will run.

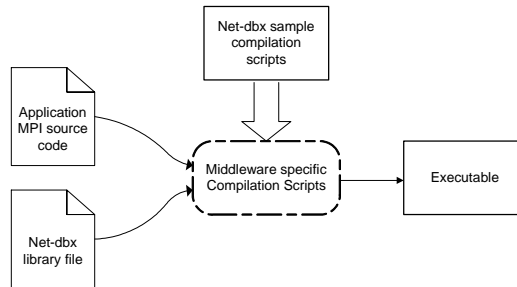


Fig. 3. Compilation Interface

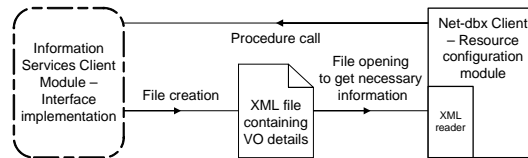


Fig. 4. Resource Discovery Interface

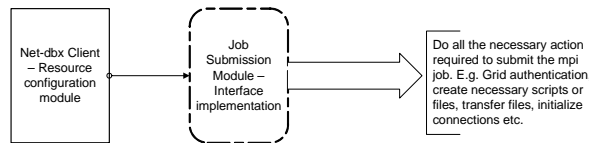


Fig. 5. Job Submission Interface

The third interface is the Job Submission Interface (Figure 5). This component takes its input from the User Interface containing all of the user's options and it is responsible to do all the necessary actions to fully submit the job. This component is provided with a ssh interface and it's able to send and receive data from a ssh connection with Net-dbx server. If the middleware in use doesn't provide a Java API then all the middleware's client services can be invoked using this ssh connection.

3.3 Communications Proxy Layer

Most of the changes to Net-dbx's architecture were made to the communications scheme, in order to provide full platform independence. Net-dbx uses telnet sessions to accomplish communication between the client and the gdb sessions attached to the individual processes. This is not feasible in a grid environment because a local user may have different mappings on the remote resources. Grid user has no real access to the resource itself; rather the grid middleware provides an interface to the resources and single sign-on to the whole system using different authentication methods. So to avoid having to integrate the authentication methods of every grid middleware currently available or under development, we have implemented a new method that is access independent.

Secure sockets [15] are used¹ for communication between the gdb debugging session, and the client. We also used sockets for the communication between the user and the application's I/O. Previously in Net-dbx, gdb was instantiated using a telnet session and received commands through telnet. Currently, in Net-dbx GI, gdb is spawned by the process itself using code from the instrumentation library integrated at compilation time, and its I/O is redirected to a secure socket.

The socket-handling is made by the proxy server running on Net-dbx server. This proxy server accepts connections from MPIHost (nodes that run processes) and from Net-dbx 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. It is only required for the ip address of the proxy server to be public. Since the MPI hosts are the ones who initialize the connections to the proxy server they may also be within private networks with non-public ip addresses.

The proxy server also help us overcome 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 [6].

3.4 Security

There are mainly two security issues raised by Net-dbx GI architecture. The first is user authentication. A user of an implementation of the Net-dbx GI debugging architecture 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 [5]. 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 authentication mechanisms for Net-dbx had to be revised and a security layer has been added to the architecture (Figure 1: shown in stripes) so the implementers of the architecture can adapt the debugger's methods to the ones required by the VO.

¹ In Grid environments where security is not an issue then secure sockets can be used.

The second aspect of security that needs to be considered is the communication between the nodes, the client and the proxy server. The use of secure sockets [15] 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.

3.5 Initialization and Running Scheme

The initialization scheme used in Net-dbx is also preserved in Net-dbx GI. The participating processes' PIDs are used to attach the debugger. The synchronization scheme [14] ensures that all the processes of interest to the user are attached in the local debuggers right after the MPI initialization calls. Net-dbx GI extends this scheme to meet the requirements of our architecture and to overcome the heterogeneity-related difficulties encountered in a grid environment. Figure 6 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 three main components are shown in the Figure 6. The Net-dbx applet which is the component running on the user's browser. The MPI processes that run on the MPI enabled nodes that are shared within a GRID. Finally, in between is the proxy server which is hosted on the Net-dbx web server.

At first, the system initiates an ssh session² with the debugger server (which is the web server) and authenticates as the Net-dbx user. The ssh session provides a command line interface to initiate and control the client services of the grid. It also handles 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 (this depends on the implementation of the module). After job submission the processes begin execution on the nodes.

The first call the processes make, after `MPI_Init()`, 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 (using standard MPI commands) and sends them to the proxy server, which in turn outputs 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 handles the program's input and output through the ssh session, which is now available to the user through a window in the client's applet. 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

² Ssh is used in Grid environments where security is not an issue.

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.

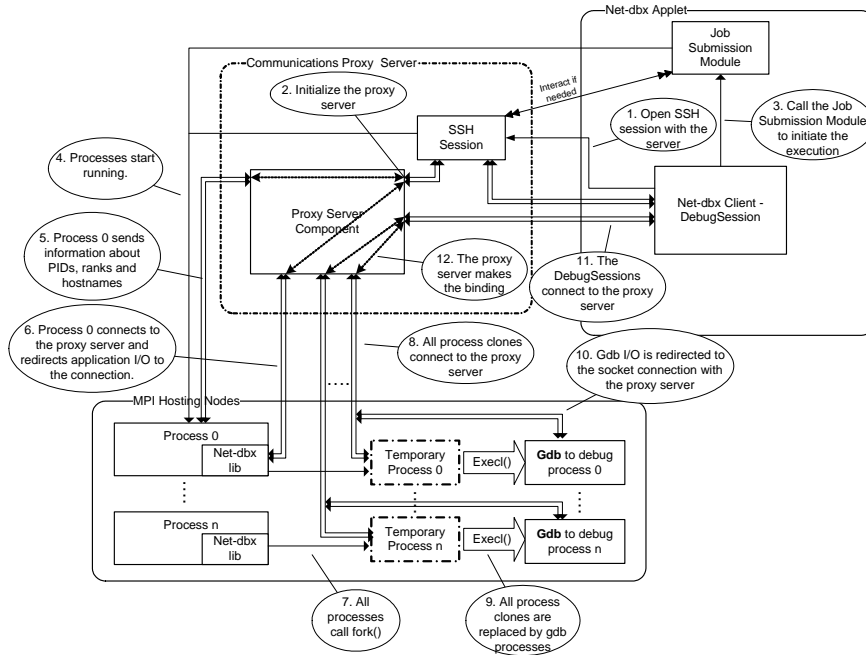


Fig. 6. Initialization and Running Scheme. This figure shows the main components of the architecture and how these interoperate with each other.

4. Prototype Implementation

A working prototype of the proposed architecture has been developed and tested. The goal of this prototype was to achieve the same behavior for the debugger on both our disparate Grid clusters, as it worked on a local network.

We have tested the Grid Interface of Net-dbx using 2 clusters. The first one consists of six dual processor SuperServer machines and the second one consists of six single processor IBM PCs. The operating system used is RedHat Linux 8. The grid middleware we have chosen is the Globus Toolkit v.2.4, which supports MPI, using MPICH-G2 as the grid enabled implementation of MPI. The underlying clusters’ MPI implementation we used is the MPICH p4 device [8], although any other MPI implementations may also be used as well. The services provided by the Globus Toolkit fulfill all the needs of our architecture. That is the resource management component, which is responsible for the job submission mechanism and

the information services component responsible for providing information on the grid environment.

The implementation of the communications layer included the full development of the proxy server in Java and the extension of the Net-dbx instrumentation library. The proxy server was fully implemented in Java, as a portable platform independent module, and it will be used in all the future implementations of the architecture. The Net-dbx library was extended in order to enable the spawning of gdb and the I/O redirections, as described in Section 3.5. We have implemented the compilation interface by using the existing Net-dbx compilation scripts and extended them for use with MPICH-G2 and Globus. The current Resource Discovery interface just hard-coded the resources that we had available in our test-bed, as full implementation of the architecture was not our goal in this working prototype. Finally the Job Submission module was also hard-coded because it depends on the Resource Discovery interface. The hard-coded layers are currently under further development, along with some additions to the user interface.

5. Conclusions and Future work

In this paper we have presented Net-dbx GI, our architecture for building a grid-enabled debugger based on the Net-dbx tool for remote debugging of local network based MPI applications. We have tested our architecture on our local Globus MPI enabled test-bed and we have shown that our architecture can be used to debug parallel grid applications on real grid environments. We are currently working on an implementation of our architecture that will work as a debugging portal on the web. Developers will be able to use the portal to debug their parallel grid applications from anywhere with the use of a common web browser. Further additions to the user interface will provide useful runtime environment information to the user, such as message propagation delays between clusters and other grid-specific and topology aware information. We are also working on a complete security infrastructure to ensure proper access to the VO's resources.

References

1. Doreen Cheng, Robert Hood: A portable debugger for parallel and distributed programs. SC 1994: 723-732
2. "Etnus, Online Documentation for the TotalView Debugger", <http://www.etnus.com/Support/docs/index.html>, 2003
3. Ian T. Foster: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Euro-Par 2001: 1-4
4. Ian T. Foster, Carl Kesselman: The Globus Project: A Status Report. Heterogeneous Computing Workshop 1998: 4-18
5. 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

6. J. Steven Fritzinger, Marinanne Mueller: Java Security white Paper, Sun Microsystems Inc., 1996
7. Andrew S. Grimshaw, William A. Wulf: The Legion Vision of a Worldwide Computer. CACM 40(1): 39-45 (1997)
8. William Gropp, Ewing L. Lusk, Nathan Doss, Anthony Skjellum: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Parallel Computing 22(6): 789-828 (1996)
9. Robert Hood, Gabriele Jost: A Debugger for Computational Grid Applications. Heterogeneous Computing Workshop 2000: 262-270
10. 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)
11. John May, Francine Berman: Retargetability and Extensibility in a Parallel Debugger. Journal of Parallel and Distributed Computing 35(2): 142-155 (1996)
12. Message Passing Interface Forum. MPI: A message-passing interface standard. International Journal of Supercomputer Applications, 8(3/4):165-414, 1994.
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. Neophytos Neophytou, Paraskevas Evripidou: Net-dbx: A Web-Based Debugger of MPI Programs Over Low-Bandwidth Lines. IEEE Transactions on Parallel and Distributed Systems 12(9): 986-995 (2001)
15. "Secure Sockets Layer", <http://wp.netscape.com/security/techbriefs/ssl.html>, 2003
16. 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.