1996

Bond- A Parallel Virtual Environment

Mihai G. Sirbu

Dan C. Marinescu

Report Number:
96-010

Sirbu, Mihai G. and Marinescu, Dan C., "Bond- A Parallel Virtual Environment" (1996). *Department of Computer Science Technical Reports*. Paper 1266.
<https://docs.lib.purdue.edu/cstech/1266>

BOND - A PARALLEL VIRTUAL ENVIRONMENT

**Mihai G. Sirbu
Dan C. Marinescu**

**Department of Computer Sciences
Purdue University
West Lafayette, IN 47907**

**CSD TR-96-010
February 1996**

Bond - A Parallel Virtual Environment

Mihai G. Sirbu¹ and Dan C. Marinescu¹

{sirbu,dcm}@cs.purdue.edu

¹ Computer Sciences Department, Purdue University, West Lafayette, IN 47907, USA

Abstract. The Bond environment, currently under development at Purdue, allows execution of parallel programs on sequential machines, clusters of sequential systems, and massively parallel systems. Bond allows a group of users to share programs and data as well as knowledge about past experiences in using parallel programs. The system uses several resource and knowledge bases to validate user's requests and to suggest alternative ways to carry out a remote computation. In this paper we report on the Bond Shell, an intelligent shell which builds on the familiar concept of a search path in a UNIX environment and allows a user to locate programs and data on remote hosts and initiate remote execution of parallel or sequential programs.

1 Introduction

The Bond environment currently under development at Purdue University is designed to support concurrent execution of parallel and/or sequential programs on computing platforms with different architecture and system software, interconnected by a high speed network. We consider a model of parallel and distributed computing which allows an individual working in a group to provide a high level description of the problem to be solved and let an intelligent environment determine a sequence of actions optimal in some sense leading to the desired result. To accomplish this goal the environment has several inference engines and maintains a set of resource databases containing the description of the computing platforms and networks, information about the programs, the services, and the data available to the group, and to each individual within the group.

Bond extends the ideas behind PVM [1] [2], MPI [4] [6], and other parallel environments developed at Argonne National Laboratory, Oak Ridge National Laboratory, NASA Ames and other research communities. It allows a user to locate programs on remote hosts by means of program databases. Bond allows a group of users to share programs and data as well as knowledge about past experiences in using parallel programs. The user is provided with information about past usage of the program, and predictions of the execution time on different platforms.

Bond is a general purpose parallel virtual environment and the immediate goal of its designers is to respond to the needs of a structural biology group. This research group uses parallel and distributed computing to solve the structure of viruses using X-ray crystallography and electron microscopy.

Fig. 1. Bond Architecture

2 Architecture and Design Philosophy

Bond is a groupware system which supports batch as well as interactive execution. It is designed to run on top of different operating systems, makes no assumptions concerning the communication libraries used by the parallel programs, and supports the management of hardware and software objects.

The Bond system shown in Figure 1 consists of a kernel, resource databases, remote services including Expert Advisors, EAs. The user interface provides access to a set of computing engines interconnected by a high speed network. The environment allows a user to provide a high level description of the problem to be solved, including execution and data dependencies. A Scheduling Advisor converts this description into a set of complex tasks and returns a task schedule to the kernel. The Bond kernel uses other agents e.g. Program and Data Replication Advisors, the Mapping Advisor, etc. to execute simple tasks. Each simple task implies running a program with a particular data set on a target system under the supervision of a Bond process. This supervisory process informs the environment about the outcome of the execution and allows the Scheduling Advisor to proceed with the scheduling of the next task or to attempt an error recovery procedure.

When activated, Bond creates a user environment, reflecting information from shared and private resource databases. The services and the Expert Advisors invoked in behalf of a user share the same view of the environment. The set of services and Expert Advisors are distributed and they can be accessed via an *oracle*. The system is open-ended, as new services are added they are registered with the oracle. Some of the services are replicated and the oracle directs a request for service to the server capable of providing the service in an optimal way.

Other Expert Advisors use facts stored in shared knowledge bases to determine if

similar task have been carried out previously, and based upon the size of the current problem suggest alternative ways to carry out the computations, provide estimates of the execution time on different configurations. The Data Replication Advisor determines if the data needed for the computation is available at the execution site and performs a variety of operations related to data staging. For example it determines if enough storage space is available at the execution site, then establishes if data conversion is necessary, if so decides where it should take place, compresses and eventually encrypts the data and finally makes a copy of the data at the execution site. The Program Movement Advisor provides similar functionality for program staging. When the remote execution completes, the EA extracts the relevant facts and stores them into shared knowledge bases.

3 The Bond Shell

As a first step in building the Bond Environment we implemented an intelligent shell which allows a user to create a *Remote Execution Environment* (REE), and to reuse it. An REE contains information about the desired host, the program to be executed, the execution mode, etc.

Bond integrates a variety of system programs like telnet, tar, uname, compress, uuencode, ftp, etc. or their functionally equivalent counterparts with its own utilities into an environment for distributed and parallel computing, using heterogeneous computing engines. The system is designed to facilitate the use of existing parallel applications rather than the development of new applications.

Most of the code of the Bond shell is written in EXPECT [5] and Tk [7], both extensions of Tcl [7]. Thus, Bond can be ported with relative ease to different platforms.

3.1 The User Interface

The user interacts with the Bond Environment by means of a user shell. The Bond SHell (BSH) recognizes a set of commands to: define/view environment(s) for remote execution, start and control remote execution, view and modify resource databases, provide help, interact with the Expert Advisors, and provide utility functions (e.g. copy files), as shown in Figure 2.

Any command unrecognized by BSH is passed to the native shell and thus the user is able to interact directly with the underlaying system. The Bond shell uses a graphics user interface (GUI) to provide help, to display and modify resource databases, and to interact with the EAs. The interaction with Expert Advisors can be initiated by the user or by the EAs in response to a user action. For example a user may request the best alternatives to execute a program. The user interface can be integrated with applications so that the application initiates the local Bond, which in turn starts the application on the remote system and then controls the remote execution.

Bond Shell Commands

There are manual pages available for the following BSH commands:

- `ch` = change host
- `env` = program environment manager
- `exec` = remote execution of a command
- `exit` = leave the Bond shell
- `help` = access the hypertext help system
- `info` = information about the resource databases
- `options` = select special execution options
- `reload` = reload the resource databases
- `run` = run a database program
- `view` = view database content

[Return to BSH Help Page, Main Page.](#)

This page was last modified by Mihai G. Sirbu on 17 Nov 1995.

Fig. 2. A Snapshot of the BSH Command Help Page as of November 1995

3.2 The Resource Databases

The Bond system uses system resource information. Part of it is related to the hardware environment (host names, platform information, clusters, etc.), and part of it is related to the application software (system, group and user programs). For access convenience these resources are stored into ASCII files called *Resource Databases*.

There are two types of resource databases: *shared* and *private*. The shared resource databases store information relevant to all users of the group, such as host and group program entries. The private databases contain information specific to individual users, such as host nicknames, clusters, user programs information. The cluster resource database groups hosts into clusters, which are used as targets for remote execution. The clusters are defined dynamically by the user.

As a general rule, shared resource databases can only be updated by a system administrator, while each user has full control upon its private resource databases. The private host database provides only the uid (user id) of the user on each of the hosts the user has access to while the shared host database contains additional information as the architecture of the system, the number of processors, the size of main memory, the I/O characteristics, and so on for all the hosts used by the entire group.

Information about the group applications are stored in the shared program database. Extending the UNIX path concept, each program has entries showing its nickname, the paths to executable files on different platforms, the paths to source and help directories. In addition, each user can define its own programs in the private program resource database.

3.3 The Knowledge Processing

Knowledge processing, currently in the development stage, is used for resource management, remote execution control and error recovery. It consists of independent Expert Advisors (EAs) which work in client-server mode. Each Expert Advisor is a rule-based expert system with an independent set of facts and rules, and has access to the user environment and shared knowledge bases. The EA can work in active and verify modes. In active mode, BSH requests a service and provides an access point to the environment of the current user. In turn, the active EA can request execution of particular operation from BSH, and is informed of the result. In verify mode, the EA receives a copy of the operation requested by the user, and does a performance analysis on it. In case of high system load, the EA will inform the user of the predicted outcome and request confirmation (for example if a requested task has a running time of many hours on a slow machine).

Resource management decisions involve the selection of the remote execution site and the execution conditions. During the post-processing phase of a program execution, environment and execution time information is stored in the knowledge base of the Mapping Advisor. Estimates for the running time are generated for requested execution environments.

Expert Advisor are also involved in the execution phase. Given knowledge base information, current system status, and input from the Mapping Advisor, the Data Replication Advisor decides where to carry out the data conversions, what type of data transport mechanism to use, etc. The Program Movement Advisor locates the appropriate executable for the target platform and moves it to the target. If no executable can be found, and if the source code is available, it will attempt to generate an executable by compiling the source code on the target system. Another Advisor determines the exact structure of the command line based on the remote system platform and the REE.

If an error is encountered, the Error Recovery Advisor is activated. A recovery may consist of several procedures tried sequentially until the operation which generated the error succeeds or no more alternatives are available and an error condition is reported. For example, if the execution of the default help viewer fails, the advisor checks the current system and returns some of the following suggestions: update the windows environment (if the access permissions are incorrect), start "this executable" (if a different copy is located by the EA), execute "this WWW program" (if the EA knows about an alternate viewer and can locate an executable for it), remotely execute "this viewer on that remote system" where a copy is available.

3.4 Other Features

BSH has several additional features as handling of hosts with special requirements, hypertext help, and automatic error reporting. Execution on massively parallel systems often require additional information, such as the number of processors, the partition name, etc. This information is usually program independent. When such a host is selected for a remote execution, a *Selector* module prompts the user for the additional information. On the remote system, a *Wrapper* module handles the additional information, together with the normal execution command. Special host characteristics are thus hidden from the general Bond processing.

The BSH help is based on hypertext documents. A World Wide Web viewer is started, with a link to the Bond help structure. The help files can be viewed without starting BSH first, and contain a general description of the shell, a list of commands, usage examples and an installation guide.

An extra layer of error recovery is available in BSH. A fatal error can be caught by a Tcl program. If a BSH component aborts, it will first generate an error report and e-mail it to the Bond system administrator and to authors of the program. A number of programming bugs were thus uncovered.

4 Future Work and Conclusions

The diversity of the architecture and system software and limitations due to latency and bandwidth of the computer networks have made heterogeneous distributed computing all but impractical in the past. While the performance of the computer networks is steadily improving, the heterogeneity and diversity of computer platforms is likely to be a factor affecting parallel and distributed computing for the foreseeable future. One possible solution to accommodate heterogeneity is to design new languages and interpreters for them, able to run on all the platforms, as in case of Java [3]. But this elegant solution is clearly unacceptable for high performance computing. In this case there are many legacy codes consisting of tens or possibly hundreds of thousand lines of code, carefully tuned for specific architectures. A solution is to have an intelligent environment where a set of Expert Advisors can make decisions based upon facts pertinent to each function to be performed.

Future developments of Bond include superconcurrency control, the definition of a High Level Problem Specification language capable to support superconcurrency, and the design of a set of servers including Expert Advisors. At the present time the user interface is able to cooperate with Expert Advisors to create scripts and pass them to the Bond Shell for execution. Clearly the environment needs to asses dynamically what is the next action to be performed and the framework for this dynamic flow control is currently under development. In this new framework Expert Advisors are used for on-line monitoring of the program execution, and for error recovery in case of system failure.

The current users find the Bond Shell effective in hiding the details of remote execution of parallel programs. Members of the Computational Biology Group at Purdue are actively using it to execute parallel application programs on a Paragon system and

on clusters of Sun, SGI, and IBM workstations.

An X-Windows front end based on Bond Shell was developed for a performance monitoring tool [8]. A simple interface allows the user to select the remote host, the remote execution directory, and to start the application. The front end starts Bond, sets up the remote execution environment, and starts the remote program. The front end development has taken half a day, including Bond installation for the user and entering the monitoring tool in the private programs database.

The code for BSH is available from the authors.

5 Acknowledgments

This research is partially supported by the NSF grants BIR-9301210 and MCB-9527131, by a grant from the Intel corporation, and by SIO, Scalable I/O Initiative. The authors express their thanks to Calin Costian, Ioana Martin, Zhongyun Zhang, and Kuei Yu Wang for providing ideas during the design phase and for their patience during testing and debugging.

References

1. Adam Beguelin, Jack J. Dongarra, Al Geist, Robert Mancheck and Vaidy Sunderam. PVM and HcNCE: Tools for Heterogeneous Network Computing, in Jack J. Dongarra, Bernard Tourancheau, *Environments and Tools for Parallel Scientific Computing*, Elsevier Science Publishers B. V., pp. 139-153, 1993.
2. Jack J. Dongarra, G. A. Geist, Robert Mancheck and V. S. Sundaram. Integrated PVM Framework Supports Heterogeneous Network Computing. *Computers in Physics*, vol. 7, no. 2, pp. 166-175, 1993.
3. James Gosling, Henry McGilton. The Java(tm) Language Environment: A White Paper. "<http://java.sun.com/whitePaper/java-whitepaper-1.html>".
4. William Gropp, Ewing Lusk, and Anthony Skjellum. Using MPI: portable parallel programming with the message-passing interface. The MIT Press, 1994
5. Don Libes. Exploring Expect: A Tcl-Based Toolkit for Automating Interactive Programs. O'Reilly & Associates, Inc. 1995.
6. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. 1994.
7. John K. Ousterhout. Tcl and the Tk Toolkit. Addison-Wesley Publishing Company, 1994.
8. Kuei Yu Wang and Dan C. Marinescu. A Performance Monitoring Environment and its Use for the Study of Paging and I/O Activity of Parallel Programs, 1996 (submitted).