Technical Report: Distributed Parallel Computing Using Windows Desktop Systems

David Hart, Douglas Grover, Matt Liggett, Richard Repasky,
Corey Shields, Stephen Simms, Adam Sweeny, Peng Wang
University Information Technology Services
Indiana University
Bloomington IN 47408

2003

Please cite as: Hart, D., D. Grover, M. Liggett, R. Repasky, C. Shields, S. Simms, A. Sweeny and P. Wang. *Technical Report: Distributed Parallel Computing Using Windows Desktop Systems*. Indiana University, Bloomington, IN. 2003. [PDF] Available from: http://hdl.handle.net/2022/13612

1. Introduction

Like many large institutions, Indiana University has thousands of desktop computers devoted primarily to running office productivity applications on the Windows operating system, tasks which are necessary but that do not use the computers' full capacity. This is a resource worth pursuing. However, the individual desktop systems do not offer enough processing power for a long enough period of time to complete large scientific computing applications. Some form of distributed, parallel programming is required, to make them worth the chase. They must be instantly available to their primary users, so they are available only intermittently. This has been a serious stumbling block: currently available communications libraries for distributed computing do not support such a dynamic communications world well. This paper introduces Simple Message Broker Library (SMBL), which provides the flexibility needed to take advantage of such ephemeral resources.

Condor [1] offers an approach to managing jobs on scattered computing resources that is well suited to this situation; there is a Windows version of Condor, although it does not at the time of this writing provide support for parallel computing. There are other systems for managing jobs in a distributed environment, such as Globus [2]. SMBL addresses a different problem: performing extended computations using a continually changing collection of small computers. We could not find a sufficiently fault-tolerant and well-behaved PVM [3] implementation for Windows. MPI [4] implementations expect the same machines at the end of a job as at the beginning. This is only reasonable, since these libraries are generally used on dedicated systems. DOGMA [5] supports the desired type of computing, but only for applications written in Java. SETI@Home [6] does not provide a general-purpose framework.

SMBL enables parallel computing on sporadically-available desktop systems by introducing a server to keep track of the processing nodes and route messages between them. The SMBL server acts as a communications broker for processes associated with a particular parallel job running on many different processors. SMBL is designed to work with heterogeneous systems. It is not a part of Condor, but they work well together. In conjunction, they can be used to run parallel jobs on Windows computers in an opportunistic fashion, without interfering with the computers' primary users. Available as open source, SMBL is scalable, flexible and robust enough for a highly constrained and highly dynamic distributed computing environment, using ephemeral resources for massive computations.

2. Overview

SMBL is unique among parallel programming libraries in its flexibility with handling dynamic processes and heterogeneous platforms. The

SMBL client library is written in portable ANSI C. The syntax and logic are similar to commonly used MPI calls. SMBL provides entry points where the application process makes MPI-like calls to inform the SMBL server of its intent to pass messages to other processes, while the SMBL server process platform heterogeneity. The SMBL server and the SMBL library are implemented on top of this socket-abstraction layer, so SMBL processes on different platforms can communicate with each other.

SMBL is only a component of the solution to this problem, and so of course all of the components of the proposed solution must be tested and shown to work well together. The components of the trial implementation are:

- An Apache-based portal to provide authentication (via Kerberos) and user services, such as creating Condor submissions;
- A Condor server to manage the task and worker queues;
- The Condor service running on desktop computers, to start and stop the worker processes on the desktop computers;
- The Worker processes running on the desktop computers;
- A Process and Port Manager server (PPM) to create a SMBL server and assign a port for each parallel session;
- The SMBL servers to handle the communication between foreman and worker processes for each particular parallel session.

In our trial implementation, the workers all run on Windows systems and the servers all run on a single Linux system. The exchange of messages between the several components of this system is illustrated in Figure 1.

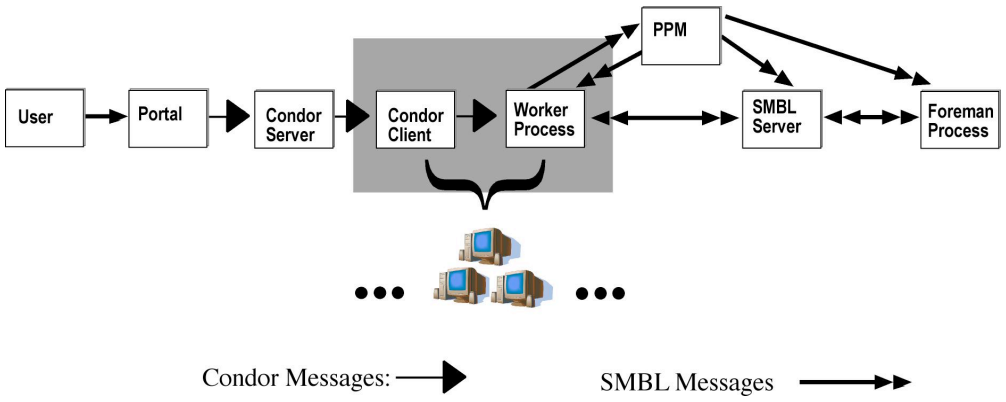

Figure 1. The shaded box indicates components hosted on multiple desktop computers.

This system relies on Condor to initiate job migration, and the application program to tolerate unannounced disappearance of worker nodes (for example, with its own task and worker tables). SMBL's role is as a messaging library that functions with a fluctuating communications world.

A note about usage: we will refer to a parallel session consisting of processes or workers possibly running on many computers, since each of those processes is referred to as a job in the Condor documentation. We also discuss processes or servers associated to a parallel session, running on still more computers. All SMBL processes associated to a particular parallel session share one SMBL server, but SMBL does not impose a static communications world on them. This is essential to achieve the goal of utilizing a pool of occasionally idle desktop computers, accommodating such

contingencies as a user touching the keyboard (and thereby taking it away) or rebooting the system.

3. The Process and Port Manager

When Condor starts a worker process, it creates a temporary directory and temporary account on a Windows NT/2000 machine. The worker contacts the Process and Port Manager, PPM, with its job identification. PPM has the following responsibilities:

- If the worker is part of a new job, PPM then:
  - checks that the total number of processes created on the server node is below a set limit;
  - picks a computer and a port number where the SMBL server listens;
  - creates the SMBL server and foreman processes on that computer.
- If a SMBL server exists for a job, PPM directs the worker to the appropriate SMBL server by sending the SMBL server process′s IP node address and port number.
- When the job is done, the SMBL server notifies PPM. Any additional workers made available by Condor for that job are then immediately terminated. The SMBL server associated with that job then terminates.

The SMBL server for a job starts only after a worker is actually started by Condor, in order to limit the workload on the server.

4. The SMBL Server

During the execution of a parallel session, multiple SMBL processes communicate via one SMBL server. The SMBL server uses select(2) [8] to provide a non-forking, non-threaded framework that allows the user process to instruct the kernel to wait for any one of multiple events to occur, and to wake up the process only when one of these events occurs. This design choice is based on efficiency and portability considerations. It relies on neither fast, semantically compatible fork() support nor compatible thread library support across platforms. Furthermore, without the need to perform expensive memory copying (in fork(2)) and process/thread creation, both processor overhead and SMBL server memory footprint are reduced.

This framework is coded in C++ on top of the socket abstraction library, with the following algorithm (a connection means to one of the SMBL application processes):

```
while(there are ready read/write/exception
connections in the
    range of connections specified)   {
    accept new connections
    read ready-to-read connections
    write ready-to-write connections
    update the range of connections
specified
  }
```

When a process (new worker) joins in, the SMBL server adds one more socket/connection to the list of sockets/connections it selects from; when a process quits, the socket/connection this process is associated with becomes inactive.

The semantics of message passing calls are incorporated into the SMBL server on top of this framework. Specifically, an object of class ConnectionHandler is defined to handle the messages from a new TCP connection; depending on the intended action, the object actually belongs to a subclass of ConnectionHandler. For example, SMBLSendHandler inherits from ConnectionHandler. The advantage of this approach is that the logic of SMBL message passing calls can be implemented in subclasses of ConnectionHandler and the server code is much easier to understand and maintain.

Each subclass of ConnectionHandler has access to its own TCP connection, the global message queue that stores the actual messages, and the global buffer that stores the global SMBL node list and the meta-messages (such as a message that informs the SMBL server of a send operation’s destination and size). When a SMBL application process makes a SMBL library call (e.g., SMBL_send), the SMBL server passes that process's TCP connection to an instance of a specific ConnectionHandler subclass (e.g., SMBLSendHandler). This subclass interprets the data coming over the socket, takes the appropriate actions (e.g., queues a message to deliver to another node), and then appends a response to the connection's output buffer. It is the main server loop's responsibility to write this data back to the application process.

5. The SMBL Library

The SMBL library implements SMBL message passing calls between various processes by sending requests to and receiving responses from the SMBL server using the abstract socket library. Both requests and responses are essentially freeform. In the current implementation, for example, requests for SMBL_send take the following form: | session id | 'S' | destination | type | size | message data |

Here session id, destination, type, and size are all 4-byte integers in network byte order. Session id is the unique id of an application process given by the SMBL server when this process connects to it, essentially the socket descriptor id on the SMBL server host; 'destination' is some other process's session id; type is an arbitrary, application-dependent message type; 'size' is the length of 'message data' in bytes; and 'message data' is the string of raw bytes. For the given call, a single byte response is expected — the character 'K' means the call succeeded (mnemonic for "Okay"), and any other character indicates a failure.

In the SMBL server process, an object of type SMBLSendHandler will take the request, construct a message object (using the socket descriptor field as the source of the message), and put it into the global message queue, writing character 'K' to the output buffer of the connection to be sent to the corresponding application process.

At present, the following message passing calls are implemented in the SMBL prototype: process initialization, message probes (blocking and non-blocking), message receives, and message sends.

Like MPI, SMBL has blocking and non-blocking calls (SMBL_probe and SMBL_iprobe). For example, SMBL_probe will return only when there is a message found that matches the pattern specified by the arguments, while SMBL_iprobe will return immediately, with appropriate flag set to 1 if there is a matching message. These semantics are implemented in the appropriate subclasses of ConnectionHandler in the SMBL server.

6. Initial Implementation and Testing

We have tested a trial implementation with fastDNAml, a program for inferring evolutionary relationships from DNA sequence data [8]. fastDNAml is a foreman-worker program which scales linearly to hundreds of processors [9]. A large fastDNAml session will generate millions of messages while consuming hundreds of CPU-hours. It is common for a researcher to simultaneously submit hundreds of variations of each job. During the 1999-2000 academic year, students in Indiana University's biology department were the largest users of parallel computing cycles, accounting for 40% of the usage of IU's IBM SP -- over 70,000 CPU-hours; one student running fastDNAml over the course of five years consumed nearly a million CPU-hours.

The test implementation has taken place in Indiana University's Student Technology Centers (STCs) during classes, while they were in normal use by students, and is currently running on 1000 desktop systems. Over the course of a week, up to 814 systems have been in use by the Condor pool, with an average over 400, making available more than 60,000 CPU-hours/week for parallel computing. Parallel sessions with up to 125 worker processes have been included in the mix. The systems associated with a parallel system vary continually: for example, a 61-processor session had 122 job migrations. Each Windows desktop is rebooted automatically each morning; some jobs have taken days to complete. The system accepts simultaneous submission of thousands of multiprocessor jobs. No untoward interactions with the 200+ application programs and plugins installed on the STC computers have been discovered. The transition from Condor mode to Interactive mode on the desktop systems occurs in less than a second – quickly enough that no students have complained.

An Apache-based Web portal provides user services and authentication, authorization, and access (in coordination with a Kerberos server); Condor manages the job and worker queues; and PPM and SMBL handle the interaction of various processes in this volatile environment. A single Linux system hosts the Web server, Condor, the Process and Port Manager, and the SMBL server and two fastDNAml foreman processes for each parallel job. This is a recognized limit to scalability: since each parallel session creates three processes, this is restricted to 48 sessions. However, SMBL is designed to support the use of additional servers, permitting many more simultaneous parallel sessions.

Figure 2 shows the performance on a moderately large job of both the IBM SP and the Student Technology Centers. "CPU-seconds/CPU" is a measure of parallel efficiency, and on a dedicated and carefully scheduled system like the IBM SP corresponds closely to wallclock time after a job begins to run — not generally the case

for the opportunistically scheduled and oft-interrupted Condor+SMBL system.

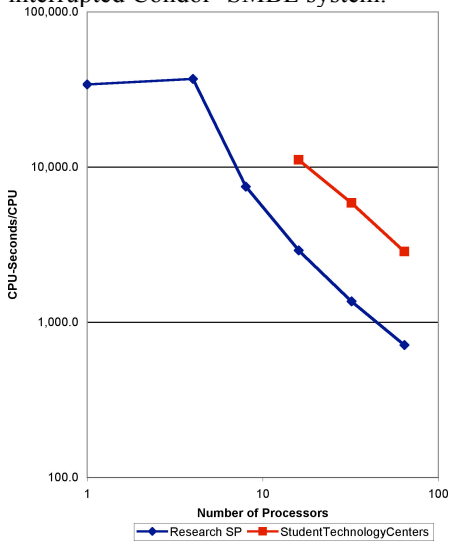

Figure 2. Scaling behavior of fastDNAml on desktop systems and supercomputer.

The IBM SP and the STC/SMBL/Condor systems each have three processes devoted to parallelization, so for example "16 processors" in fact means just 13 workers. The desktop systems in the STCs deliver ~25% of the performance of the SP on this particular application; the scaling behavior is quite similar.

7. Conclusions and Future Plans

SMBL provides an infrastructure for information exchange in connection with parallel computations using ephemeral resources, something that other parallel computing messaging libraries do not do. In conjunction with Condor, it allows the scavenging of cycles from idle Windows-based desktop computers, something that could not otherwise be done. SMBL is useful for parallel programs with a single thread of control, whenever the ratio of computation to communication is large enough and both can be subdivided finely enough —for example in design studies, Monte Carlo methods, and foreman-worker programs. It has not been difficult to make such programs fault-tolerant with regard to worker processes. Conversion from MPI to SMBL is quite straightforward; migrating from Unix to Windows has been more of a challenge.

SMBL offers the possibility for Indiana University to migrate applications that have consumed a significant fraction of IU's supercomputer resources to a pool of Windows-based desktop computers. While these jobs run more slowly than on IU's IBM SP, these applications are running on a system that is essentially without additional cost, scavenging cycles from PCs purchased for use by students as personal productivity workstations. This has been done on a trial basis with no adverse effects on the primary users' computing experience. We are currently adding additional SMBL analogs of MPI calls, and expanding the number of applications available from our portal.

SMBL is open-source and available at *http://www.indiana.edu/~rac/hpc/SMBL/*

References

[1] Livny, M., J. Basney, R. Raman, and T. Tannenbaum, "Mechanisms for High-Throughput Computing," *SPEEDUP* 11, 1997.

[2] Globus Project, http://www.globus.org/, accessed 10 February 2003.

[3] Geist A., A. Beguelin, J. Dongarra, W. Jiang, M. Manchek and V. Sunderam, *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, 1994.

[4] Gropp, William, M. Snir, W. Nitzberg, and E. Lusk. *MPI: The Complete Reference*, MIT Press, 1998.

[5] G. Judd, M. Clement, and Q. Snell, "DOGMA: Distributed Object Group Management Architecture", in *Proc. of the Workshop on Java for High Performance Network Computing*, Stanford University, Palo Alto, CA, USA, 1998.

[6] SETI Institute, http://www.seti.org/, accessed 10 February 2003.

[7] Olsen, G. J., H. Matsuda, R. Hagstrom, and R. Overbeek, "fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood," *Comput. Appl. Biosci*. 10: 41-48, 1994.

[8] Stevens, W. R., *UNIX Network Programming*, Prentice Hall, 1990.

[9] Stewart, C.A., D. Hart, D. K. Berry, G. J. Olsen, E. Wernert, and W. Fischer, "Parallel implementation and performance of fastDNAml - a program for maximum likelihood phylogenetic inference," *Proceedings of SC2001*, 2001.