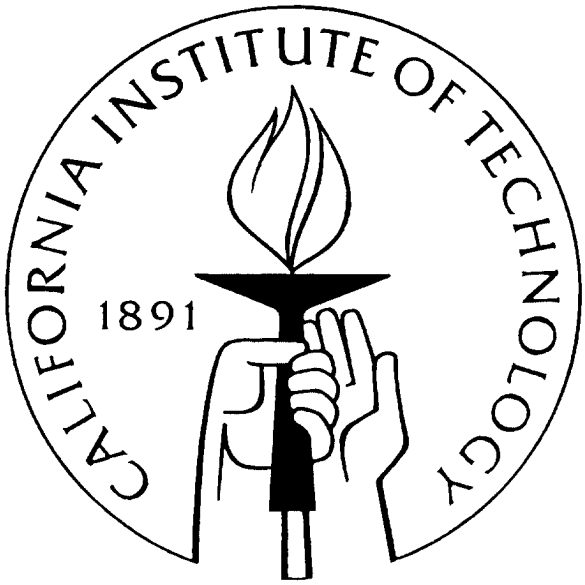

A Framework for Migration of Networked Intercommunication Processes

Daniel Azuma
James Lin
Eugene Chun
Matthew Richardson

Computer Science Department
California Institute of Technology

Caltech-CS-TR-97-21

A framework for migration of networked intercommunicating processes

Daniel Azuma dazuma@kagi.com
James Lin jlin@ugcs.caltech.edu
Eugene Chun genechun@ugcs.caltech.edu
Matthew Richardson mattr@ugcs.caltech.edu

Computer Science Department
California Institute of Technology
Pasadena, CA 91125

As research turns out new algorithms and applications for parallel and distributed systems, the benefits of a framework for mobile processes become clear.

Introduction

The network age—superfast local networks and the all-encompassing Internet—promises great advancements in hardware resource management. Much of the existing computational power sits idle for long periods of time, while a few machines are overloaded. However, emerging technologies offer the ability to reassign jobs to idle hardware. Many types of programs, from parallel computations to distributed applications and mobile agents, would benefit greatly from the availability of the vast resources of idle computing time often available over networks—for these, the potential gains in efficiency for these types of programs are significant.

MINIPROCs (MIgration of Networked Intercommunicating PROCesses) is one of several research efforts currently underway aimed at creating a useable infrastructure for distributing computation jobs over a wide-area network of workstations. Its goal is to enable parallel and distributed algorithms to be run concurrently and transparently on such a network.

Overview of MINIPROCs

MINIPROCs was designed with the following set of goals:

- Cross-platform operation
- High scalability
- Support for transparent process migration in response to changing environment
- Support for interprocess communication and transparent message routing
- Low-overhead software infrastructure
- Easy integration with outside software systems

Cross-platform foundations

Platform independence is one of the keys to creating a flexible network of migratable processes. To help achieve this, Java was chosen as the foundation language for MINIPROCs. Java, with its write-once-run-anywhere nature, immediately brings the capability to span platforms by hiding the underlying operating system and hardware architecture. Both the MINIPROCs infrastructure and MINIPROCs-based programs are written completely in Java. Although there are some drawbacks to using Java, namely speed and uneven implementation of Java virtual machines across platforms, we believe these will become less significant over time.

A scalable distributed framework

MINIPROCs is designed around an "anarchistic" peer-to-peer network topology that can dynamically grow and shrink as individual workstations enter and leave the pool of available hardware resources. In this topology, there is no central server, nor even a well-defined center of networking operations, making the system almost infinitely scalable.

Because of the completely peer-to-peer nature of the MINIPROCs network, every hardware node is equally knowledgeable about the current state of the network, and can therefore direct the allocation of needed resources on its own. Thus, if the network is split, each side will remain capable of operating as an independent entity, and two networks are also capable of joining to form a single larger network. Moreover, to gain access to the entire network, an outside user or software agent need only know the physical location of any one of its nodes.

Communication and migration

To implement communication between different components of a MINIPROCs-based program, the system implements a message-passing library built on top of another infrastructure called the Infospheres Infrastructure[1]. Infospheres supports communication and persistence of objects written in Java. Although MINIPROCs does not use persistent object services, it can be extended to use persistence services provided by Java 1.1 or Infospheres.

Another major feature of MINIPROCs is its ability to transparently migrate processes, from machines that are leaving the network or are no longer in an idle state to alternate nodes. Thus, any user can leave his personal workstation connected to a MINIPROCs network when not in use, and then, when the workstation's resources are needed again, any running process can be kicked off, causing it to migrate to another node. Any communication links the migrating process has established with other processes are automatically rerouted to the new host. In this way, the system is able to take maximum advantage of idle CPU cycles without inconveniencing users. The ability of processes to migrate can also be used to support small programs called agents that perform tasks and migrate from one machine to another.

How it works

The MINIPROCs network in action

Individual workstations in the MINIPROCs system are interconnected in a homogeneous, peer-to-peer network. This network is in the form of an undirected graph, each node maintaining a list of neighbors. Any new workstation that wishes to join the system need only connect to one or several "nearby" workstations already in the network. Similarly, any entity that wishes to communicate with the network need only communicate with one member, which will then route the message to the appropriate region, or propagate it into the rest of the network.

The most common request made of the network is to start a process. When an entity wants to start a process, it sends a message containing all the appropriate information about the process—the URL of the Java classes, a parameter list, a set of security keys, and a return address—to any node in the network. That node will then either run the process on itself or initiate a breadth-first search for a nearby node capable of running the process. A node is capable of running a particular process if it is currently idle and able to accept the security key signature provided by the process. If the network is setup such that the topology reflects a sense of locality, this breadth-first search will ensure that the process actually runs "close" to where it was initially requested. Once a suitable node is found, that node first sends to the initiator's return address an acknowledgment, which is in the form of a process ID that can be used to communicate with the process. It then loads the classes over the network and starts the process within its own thread.

In this scheme, individual nodes need only maintain information local to themselves, thus eliminating the need for a central server.

The MINIPROCs network can also support migration of processes in response to the demands of the host workstation. To this end, a process is expected to periodically checkpoint with the host system, by calling a provided checkpointing method. MINIPROCs will then check with the host to make sure the process may continue to run. If not, the system instructs the process to serialize its current state. The thread is then terminated, and that node initiates another search for a suitable node to continue running the process, as described above. Once a new node is found, both the process and its saved state are transmitted over the network to that new node, the state is restored, and the process is allowed to resume running.

Communication routing

Because interprocess communication is an essential part of most parallel and distributed algorithms, MINIPROCs provides a communication layer that completely hides the underlying network topology with its IP addresses and ports. Designed to match the message-passing interface provided by the Infospheres Infrastructure, the MINIPROCs messaging library provides for the serialization of objects and their transmission between outgoing and incoming mailboxes. The ordering of messages is preserved, and messages destined for migrating processes are automatically routed to the correct physical address.

This implementation is built on top of Infospheres, which already provides object serialization, transmits between mailboxes statically bound to a specific port and mailbox name on a specific host, and preserves message ordering. MINIPROCs extends this capability to support relocatable processes. In addition to being bound to a mailbox to a port and mailbox name on a specific host, a MINIPROCs mailbox can be bound to a process ID and a mailbox name. A process ID, represented by the ProcessID object, uniquely specifies a specific process running anywhere on the network, and contains enough information for the network to find the process.

Inside a ProcessID is the name (IP number and port number) of the node on which the process originally started running, and a string. The string is built from the host's IP address and time stamp, and so is guaranteed to be unique across the entire network, for all time. The node is the process's permanent address. If the process is still running on there, messages addressed to it can be delivered directly to it by that node; otherwise, messages are forwarded to the node to which the process migrated. That node, in turn, then either delivers the message or forwards it on to the next host in the migration path. In this way, each message follows the migration path of the process, to its final destination, thus preserving message ordering.

Shortcutting the message forwarding path

In busy networks, it is possible for this migration path to grow lengthy. Thus, to cut down on the resulting message-passing latency, a shortcutting algorithm is built into MINIPROCs. This algorithm attempts to keep the forwarding chain to a maximum length of two nodes, by bypassing nodes in the middle. It also prevents messages from "cutting in line," by halting traffic during the shortcutting process.

Figure 1 illustrates the shortcutting algorithm. The process begins as soon as the network detects that a process has migrated twice. At this point, three nodes are in the forwarding chain: the initial node A, an intermediate node B, and the final node C. At this point, node B will wish to drop out of the chain, and so it will initiate shortcutting.

In the first step of the algorithm, B sends a Disconnect message to A (figure 1a). This message is a request to begin shortcutting. A responds by suspending the forwarding of all messages related to this process.

Next, A sends AckDisconnect to B (1b). This is an acknowledgment that forwarding has been suspended and shortcutting can proceed. Because Infospheres preserves message ordering, this message will follow all normal messages being forwarded from A to B. Thus, the receipt of an AckDisconnect is a signal to B that it will receive no more forwarded messages.

Next, B sends NewPrev to C (1c). This is a shortcutting notification. Again, this message follows all normal messages being forwarded along the chain from B to C, so once NewPrev has been sent, B is can drop out of the forwarding chain.

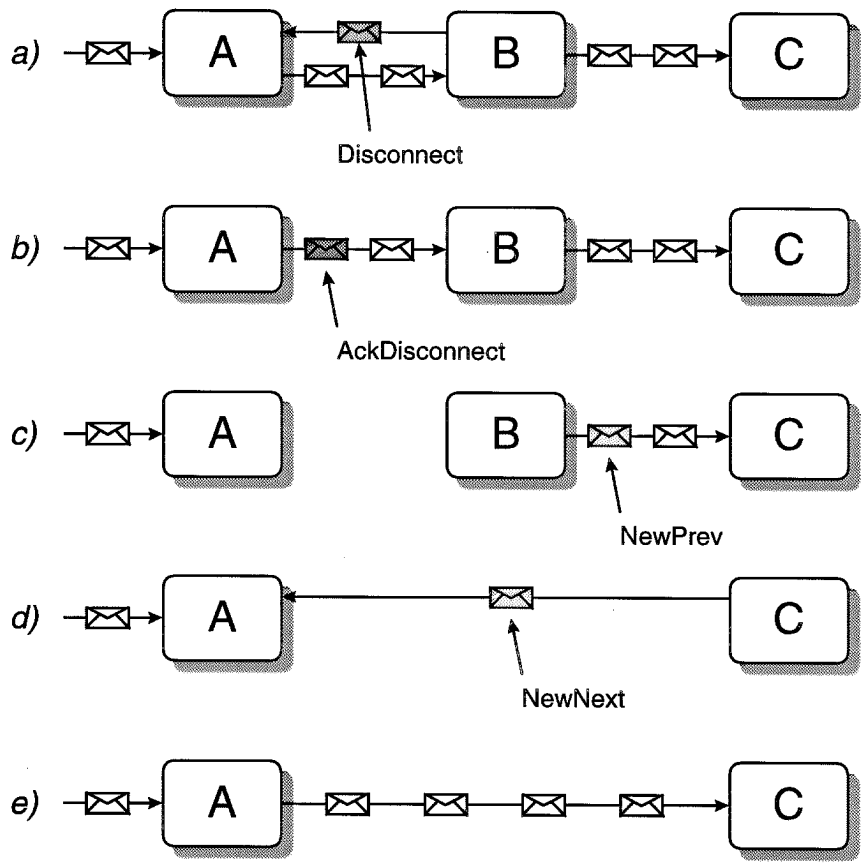

Figure 1. Shortcutting algorithm.

Finally, C sends NewNext back to A (1d). This is a notification that shortcutting has been completed, and A can resume forwarding (1e). Because all messages that were moving through the chain while shortcutting was proceeding have finished propogating through, A can now forward messages to C without worry of any messages "cutting in line." B has been effectively removed from the chain without disturbing the message ordering.

One more case that may arise is a shortcutting collision. This occurs if the process migrates from C to another node, D, while the shortcutting algorithm is running. In this case, C will then attempt to drop out by sending a shortcutting request, Disconnect, to B.

To deal with this case, we modify the algorithm in this way. Suppose node C sends Disconnect to node B (figure 2b) while B is itself in the process of dropping out of the chain (i.e. it has already sent Disconnect to node A, figure 2a). Then B does not respond. Therefore, node C, instead of receiving AckDisonnect (acknowledgment that shortcutting can begin) from B, will instead receive NewPrev, the shortcutting notification (2c). This is a signal that B is no longer a valid previous node. C then sends NewNext to the new previous node, A, to complete the first shortcutting algorithm (2e), and can follow that message immediately with another Disconnect to restart its own request to drop out (2f). This protocol prevents two shortcutting algorithms from colliding.

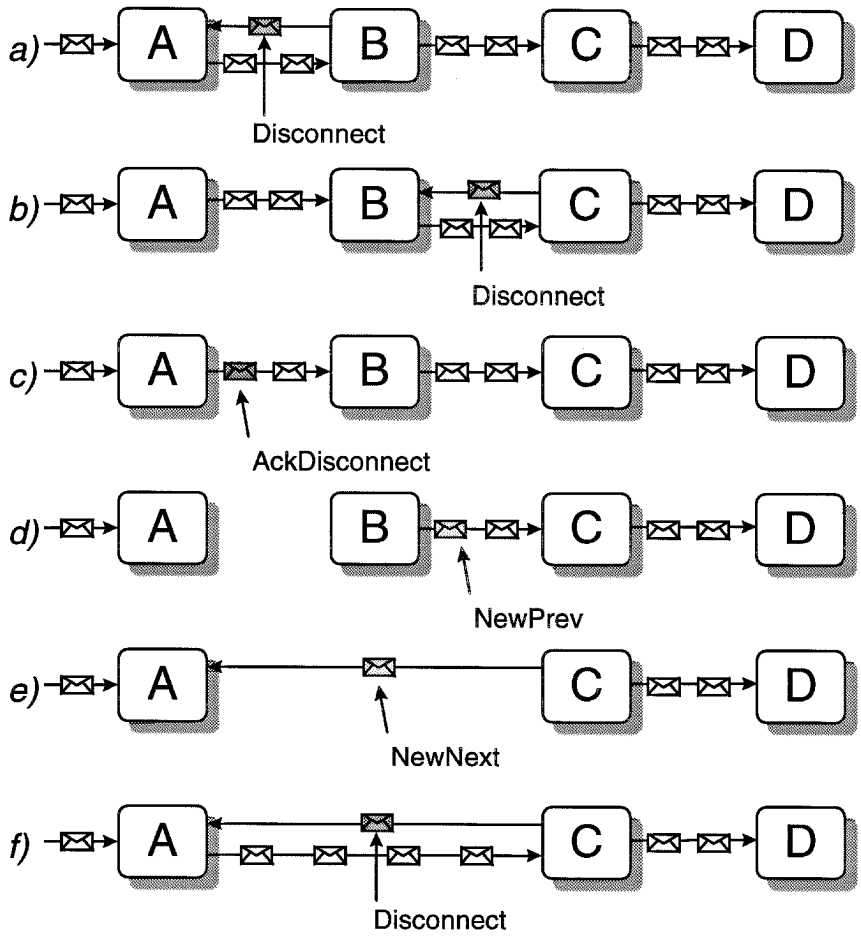

Figure 2. Avoiding shortcutting collisions.

MINIPROCs demonstration

Using the MINIPROCs framework is designed to be simple. Network administrators can set up MINIPROCs daemons on individual machines on a local network, connect them up in any arbitrary topology, and even include links over the Internet to form a global network of potentially available hardware. A user can then write a MINIPROCs-based Java program and start it up on the network, and the framework will handle the rest.

An example Mandelbrot program

Accompanying is an example MINIPROCs program that implements a simple Mandelbrot set viewer using the Java 1.0.2 AWT. Calcuation is done on an arbitrary number of workers using a master-slave topology, with jobs defined as rows in the image. A worker-side job buffer is implemented to combat network latency.

Results

Figure 3 illustrates some sample running times for the Mandelbrot program. To simulate real-world performance, it was run on a 12-node MINIPROCs network running on a network of heavily used SGI Indy workstations under various loads. The image calculated

is a 400×400 pixel area entirely within the set. The iteration value affects calcuation-communication ratios; thus, the 500-iteration runs include a relatively large amount of communication and the 5000-iteration runs include a relatively large amount of calculation. "1 worker" represents a sequential run without the use of MINIPROCs.

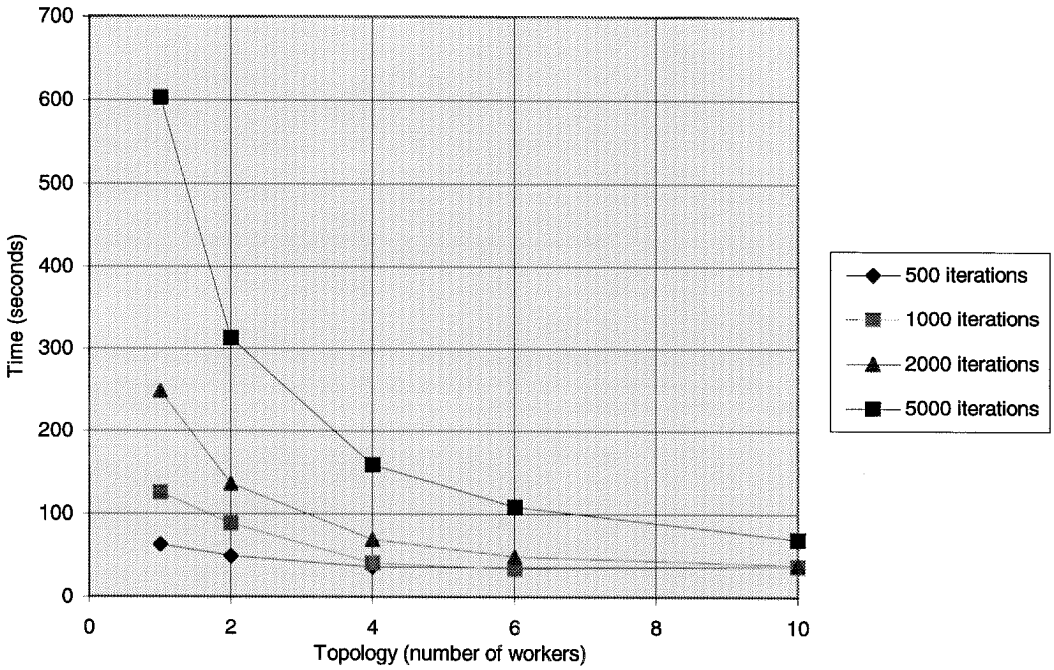

Figure 3. Execution times for Mandelbrot program.

The above results demonstrate that the speed improvement can be significant, particularly for programs with large calculation-communication ratios.

Comparison with other systems

There are numerous other projects underway with similar goals. Several of them aim to create a virtual computer over a network of computers. These include Legion at the University of Virginia[2], Condor at the University of Wisconsin[3], and the Network of Workstations (NOW) at the University of California at Berkeley[4]. Each takes a different approach.

To achieve platform independence, Legion uses a custom language and runtime system. After the Legion project was started, Java was introduced. MINIPROCs takes advantage of Java's runtime library and cross-platform capabilities, making a custom language and runtime system unnecessary.

Condor concentrates on scheduling tasks on workstations with low loads. It relies on a central server to allocate resources and direct traffic. By contrast, MINIPROCs does not rely on a central server and therefore is more scalable. Condor also does not provide a

framework for processes to communicate, especially if the processes migrate. MINIPROCs provides such a framework.

NOW focuses creating a virtual supercomputer by networking a set of dedicated workstations with high-speed connections. Unlike NOW, MINIPROCs can run over an existing network of computers and does not require a bank of workstations dedicated to the virtual computer.

In fact, the scope of MINIPROCs is broader than the systems mentioned above, as it includes mobile agents. The Aglets Workbench[5], developed by IBM Research, is a framework for mobile agents written in Java. Like MINIPROCs processes, the agents, called aglets, can stop execution, migrate to another machine, and resume execution at the same state in which it had stopped. There are several differences in design between MINIPROCs and the Aglets framework.

For each IBM aglet, there is an aglet proxy. Aglets communicate with each other via these proxies. Proxies provide location independence for the aglets; the proxies are always local, but the aglets may be on a remote machine. In MINIPROCs, the processes communicate directly without proxies, since the process ID provides enough information to locate a process on a remote machine.

Since our focus was not originally mobile agents, MINIPROCs does not provide some services which agents may find useful, such as itineraries or agent cloning. These services can be easily added in the future.

Conclusions

MINIPROCs is a highly flexible framework for distributed programs to run over many machines networked together. A program written for MINIPROCs consists of one or more processes. A process can communicate with other processes in the same or different program and migrate to another machine while retaining its state and all of its communication links. The MINIPROCs framework provides location independence; a process never needs to know on what machine a process is located to interact with it. MINIPROCs can be used for a variety of applications, from setting up a virtual parallel supercomputer to supporting mobile agents that can roam over the network. The framework is highly scalable and can run on multiple platforms.

Future directions

MINIPROCs is a work in progress. Although many of the basic technologies are implemented, several additional issues remain to be addressed before this and similar infrastructures can be put into widespread use. Among these are:

- **Fault tolerance** The infrastructure currently makes no provision for catastrophic events, such as daemons unexpectedly going offline. This is an important step towards making the system sufficiently robust to be usable in an uncontrolled environment.

- **Process supervision** The infrastructure should allow more control over rogue processes, including the ability to terminate them remotely. This can probably be accomplished by a high-priority supervisor thread.

- **Distributed termination** A useful feature would be the ability to terminate all processes involved in a deadlocked algorithm, possibly by maintaining a process spawning tree and a deadlock detection mechanism.

- **More flexibility for controlling daemon lifetimes** Currently, the MINIPROCs framework is set up such that certain daemons cannot go offline. For example, if a process was started on a daemon and the process migrated to another daemon, the first daemon must stay online to forward messages to the process's new location. This creates inflexibility in controlling when machines can be taken offline for maintenance or other such work.

Acknowledgments

We would like to thank Professor K. Mani Chandy for giving us the opportunity to research and develop the MINIPROCs framework as part of the Distributed Computation Laboratory class at the California Institute of Technology. This work was funded in part by Air Force Office of Scientific Research under grant AFOSR F49620-94-1-0244.

References

[1] Caltech Indospheres Project
http://www.infospheres.caltech.edu/

[2] The Legion Project
http://www.cs.virginia.edu/~legion/

[3] The Condor Project
http://www.cs.wisc.edu/condor/

[4] The Berkeley Network of Workstations (NOW) Project
http://now.cs.berkeley.edu/

[5] Aglets—Mobile Agents in Java
http://www.trl.ibm.co.jp/aglets/