IceT: Distributed Computing and Java *

Paul A. Gray and Vaidy S. Sunderam
Emory University Dept. of Mathematics and Comp. Sci.
{gray,vss}@mathcs.emory.edu

Abstract

Numerous factors relating to the massive embracing of the Internet by the general public have clearly influenced network programming techniques and paradigms. The Java programming language both influences and is influenced by the requisite and dynamic aspects of network programming, yet it's viability as a programming language for the scientific community is still to be determined. This paper describes IceT, a novel framework for collaborative and high-performance distributed computing which has been built upon a Java substrate. The IceT project incorporates the advantages of Java with the established techniques of distributed computing, while maintaining a deference to what each discipline is best suited for. The result is a dynamic distributed computing environment upon which data and processes are highly portable amongst heterogeneous platforms and multiple users.

1 The IceT Objective

There are many contrasts to be found when comparing *Internet programming* and *high-performance distributed computing*. Consider, for example, Internet programming as it relates to the World Wide Web. Users are able to anonymously access processes and data which are located at remote locations for remote execution (e.g. a cgi-bin perl script) or download processes for local execution (e.g. a Java applet). In such settings, topics which need to be addressed include the consequences and implications of executing processes remotely; anonymously; securely; on machines which the person executing the process commonly has no privileges[1]. On the other hand, in high-performance distributed computing, where users' programs are local, executed semi-locally (within localized intranets), the topics of "remote execution of processes" and such are not found in the mix of security issues and the like.

The aim of the IceT project has been to mutually incorporate approaches and techniques found in Internet programming with the established and evolving distributed computing paradigmns. Necessarily, this includes the ideas of harnessing geographically-remote resources for anonymous utilization, portability of processes and data, and the consequential security issues that arise. The incorporation of respective Internet and distributed programming techniques would lead to a natural environment for collaborative computing and an extention to the traditional distributed computing environment. Examples of extended features incorporated into IceT are: dynamic *merging* and *splitting* of virtual machines, mutli-user awareness, portability of processes and data across multiple virtual machines, and a provisional framework for mutli-user programs.

The subsequent portion of this document describes some of the novel features of IceT and the role played by the Java programming language. Details on what is currently implemented and the future of IceT are also presented along with some Java-based examples which serve to further illustrate the novel features currently included in IceT.

* Research supported by Army Research Office grant DAAH04-96-1-0083 U. S. Department of Energy Grant No. DE-FG05-91ER25105 NASA Grant NAG 2-828 and the National Science Foundation Award Nos. and ASC-9527186.

[1] Lack of login privileges, for example

2 Features of Java Benefitial to the IceT Design

Loosely speaking, the aspects of Internet programming that have been introduced to distributed computing viz-a-viz IceT relate to uploading and porting of processes and data; and providing combined, provisional access to unowned, remote and heterogeneous resources. The `ClassLoader` class, `SecurityManager` class, and the JVM were components of Java which were relied upon for this task.

2.1 The Java ClassLoader

From the IceT perspective, the Java `ClassLoader` class is the primary feature of Java which can be used to complement distributed computing. Through the Java `ClassLoader`, processes represented as Java bytecode can be uploaded, instantiated, and executed on remote hosts where the "owner" of the bytecode has no privileges whatsoever and where the execution is subject to the security restrictions imposed by the owner of the local resouce.

In IceT, portability of Java bytecode is achieved by extending the `ClassLoader` class to load, resolve, and instantiate new instances of classes located on remote hosts. Loosely speaking, this component of IceT is similar to (albeit functionally the reverse of) a web browser which downloads applets from wide-spread resources across the Internet. A user wishing to upload his/her process to a remote IceT host would do so through an IceT `ClassBootStrapper`. The `ClassBootStrapper` is an extention of the Java `ClassLoader` and is responsible for obtaining and resolving bytecode representations of class files to be executed and is configurable to impose different levels of security restrictions to users of the local computational resources through extensions to the Java `SecurityManager` class.

2.2 Security

Java's language-level security support offers many additional advantages. By allowing provisional usage of local resources to possibly anonymous users, significant security issues arise. The `SecurityManager` class provides a quick and easy means for imposing elementary security restrictions. By extending the Java `SecurityManager`, `SocketImpl`, `Thread` and `File` classes, the local IceT environment can be configured to accommodate varying levels of restrictions placed upon workload usage, external and local socket connections, process portability and file system access.

2.3 The JVM

Java is an interpreted language, which means that Java programs must be parsed in some fashion to be executed. "Compiled" Java programs exist in the form of Java bytecode, and execution of a Java program is accomplished through running the bytecode through a Java bytecode interpreter. Java provides a least-common denominator in providing a uniform "*Java Virtual Machine*" (JVM) for managing the task of interpreting the bytecode. Java's JVM is responsible for verification and execution of the Java bytecode, conditional access to system-dependent attributes, and strong ordering of primitive data types. Consequently, the standardized JVM provides a uniform substrate for IceT processes and data. Upon the JVM substrate, processes and data are entirely portable. This provides IceT with the mechanisms needed for stand-alone realization of the features described in the opening section. In addition, this allows IceT to serve as a utility for porting system-*dependent* processes and data to remote locations for anonymous (or governed) linkage/binding/execution and to serve as the mechanism through which users and processes maintain communication with and awareness of remote processes, data, resources or users.

Reiterating, the JVM provides Java bytecode with a uniform, system-independent view of the underlying architecture. With the guarantee of uniformity of the JVM, bytecode is truly architecturally independent. This aspect of the Java programming language allows platform-independent execution of the Java bytecode regardless of the platform which on which the bytecode was generated. This allows *any* IceT user to execute *any* IceT process.

3 IceT Illustrated

The objective of the IceT project has been to blend the current distributed computing framework with new and evolving technologies; including but not limited to the aspects and features found in Java. This section serves to illustrate the suitability of IceT for traditional distributed computational tasks — in the form of a matrix multiplication example — and to illustrate the novel attributes and collaborative features brought about by the enhanced portability of tasks — through example of a just-in-time installation of an IceTChat Tool.

The examples below assume that an IceT environment has already been established. For details on the *adding* and *merging* of resources to the IceT environment, see [3].

3.1 IceT and Traditional Distributed Computing

Consider a master program which is faced with the problem of using slave processes to multiply matrices **A** and **B**. One way of distributing this problem is to split **A** by rows and to pass these sub rows of **A** and the entire matrix **B** to the slave process for subsequent multiplication. The following shows how one might address this problem (albeit naively) utilizing IceT and the Java programming language.

The Master Program The first task is to make the master program IceT-aware, this is done by `import`-ing the IceT-specific commands. In order to interact with IceT components, the program takes the form of an extension to the `TaskProtocols` class, which defines various IceT functions such as `send, recv,` etc. In this example, the master and four slaves will be responsible for the distributed multiplication.

```
 1  import IceT.*;
 2  import java.util.*;
 3  public class MatrixMultiply extends TaskProtocols {
 4    final static int NUM_SLAVES = 4;
 5    final static int NUM_WORKERS = NUM_SLAVES+1;
...
12      try {
...
17        TaskElement myTaskElement = IceT();
18        /* Prepare to spawn the slaves */
19        TaskElement[] taskArray = new TaskElement[NUM_SLAVES];
20        int numSpawned = spawn("Slave",NUM_SLAVES,taskArray);
...
27        int block = (int) SIZE/NUM_WORKERS;
...
34        /* Pack the entire matrix b */
35        for (int i = 0; i< SIZE; i++)
36          bufferB.pack( B[i] );
37
38        for (int k =0; k<NUM_SLAVES; k++){
39          bufferA.reset();
40          bufferA.pack(SIZE);
41          bufferA.pack(NUM_WORKERS);
...
45          for (int i = 0; i< block; i++) {
46            int row = k*block;
47            bufferA.pack( A[row+i] );
48          }
49          /* Send off the infomation to the slave */
50          send(bufferA,taskArray[k].myId,888);
51          send(bufferB.bufferClone(),taskArray[k].myId,889);
52        }
...
68        // Reap in the results from the slave
69        for (int k = 0; k<NUM_SLAVES; k++) {
70          b = recv(777);
71          for (int i = 0; i < block; i++) {
72            int row = k*block;
73            C[row+i] = b.unpackDouble(SIZE);
74          }
75        }
```

The program initializes its IceT environment and spawns `NUM_SLAVE` copies of the `Slave` program (lines 1-27). Problem parameters, the slave's portion of **A** and the entire matrix **B** are packed and sent to the respective processes (lines 34-52). The master program calculates the remaining portion of the matrix multiplication (omitted) while the slaves compute their results. Finally, all the slaves' computations are collected by the master and the matrix is processed (lines 68-75).

The slave program follows a similar incorporation and initialization of the IceT environment as the master program, and waits to receive the buffers containing the matrix information. The slave then computes its respective matrix multiplication, packs up the results, and returns them to the master program.

This matrix multiplication example is intended to demonstrate the suitability of IceT for traditional parallel/high-performance applications. Additionally, this example shows the consistency with traditional distributed computing programming interfaces of the IceT programming interface, which holds a strong similarity to the existing programming interfaces of PVM[2] and MPI[6]. However, IceT provides some additional features to the above task. Noteworthy aspects of the matrix multiplication problems which are unique to IceT include:

- Copies of the IceT `Master` and `Slave` programs need not be provided to each of the computational resources in the user's IceT environment. In the absence of finding a process locally, IceT manages the task of locating the process and uploading it to the appropriate resource(s) as necessary.
- In PVM, the user must supply a compilation of the program for each distinct architecture of the resource pool. In IceT, any process can be freely ported and executed within the IceT environment. By using the system-independent language of Java, processes enjoy a wide breadth of architectural freedom.

For performance benchmarks which compare the native IceT/Java matrix multiplication above with variations of equivalent `C` and PVM[2] implementations, see [4].

3.2 *Just-In-Time* Process Installation

Whereas the focus of the matrix multiplication example was to show that IceT adheres to traditional mechanisms of distributed computing, IceT's purports additional benefits in process/data portability and in bringing together unrelated or external resources.

As in traditional distributed computing, local processes in IceT can be executed on semi-local resources (as described in the first section). This paradigm is extended in the IceT environment to further allow local processes/data to be uploaded to remote, unowned resources (a colleagues system, for example), to locate remote processes/data and to download them locally, or any dynamic combination thereof. Processes themselves have the ability to port themselves amongst the resources of the environment provided they adhere to the security impositions imposed thereon. All processes possess the ability to maintain persistent communication with all other processes within the IceT environment (unlike applets) with communication to restricted elements of the environment also governed by the security imposed upon the resources involved. The degrees of portability, local execution of anonymous processes, and permitted communication channels are governed by the security measures imposed by the owner of the local resource.

"Just-in-time" process installation refers to the ability for a process to enter a resource, perform its computation within the restrictions imposed by the environment, and to leave no residual[2] of the process upon termination or migration from the resource. In IceT, a process has the ability to upload itself to a remote resource, become instantiated as a separate thread of execution[3], complete its computational task, and terminate only to be garbage-collected by the JVM. Moreover, this may be accomplished without accessing or storage of bytecode onto the filesystem. This process may also communicate with the various entities enrolled in the conglomerate IceT system without actually opening direct socket connections or being made aware of the ports these entities are listening on. If stringent communication measures are imposed upon the process, the local IceT daemon governs and manages the routing of communication to permitted external entities.

3.3 Examples of Just-in-time installations

The usefulness and utility of the just-in-time installation of software has several immediate application areas:

- In the simplest sense, just-in-time installation of software could be used for local and remote *data mining*. In the situation where data pool is quite large and the data of interest is relatively small, transferring the data for subsequent parsing or manipulation would take more time than porting of the software which would parse the data and return select data of interest or possibly some visualization of the representative data.

[2]Leaving files associated with the process on the local filesystem, for example.

[3]This thread can be started, stopped, re-prioritized, or even serialized and possibly ported elsewhere or archived for postponed execution — all governed by the local IceT environment and its configuration by the owner of the local resource

- As a facilitating environment for collaboration, the just-in-time installation of software would mean that any person would be able to utilize a vast collection of collaboration software with fellow collaborators. In the event that software necessary for collaboration is lacking from a colleague's local system, the requisite software could be uploaded to that colleague for subsequent execution and involvement in the collaboration[4]. IceT provides the substrate upon which collaborative tools would be uploaded and subsequently provides the channels through which these tools would be allowed to communicate.

4 Comments on Related Work

IceT shares limited similarities with JavaPVM[7], and JPVM[1]. IceT also has attributes similar, by design, to PVM. The objective of IceT was to take the traditional features of distributed computing, like those found in PVM, and to extend them to include additional features such as the merging of virtual machines, the dynamic uploading of processes across virtual machines, etc. Both JavaPVM and JPVM address programming aspects and objectives orthogonal to those of IceT, and hence offer features not found in IceT. Like IceT, JavaPVM and JPVM provide users with an environment suitable for distributed computing. However, IceT's environment provides the additional functionality of harnessing and merging with remote resources where the user lacks privileges and directly supports the uploading of processes and data upon aggregated resources.

5 Conclusion

Through IceT, we've been able to incorporate some novel ideas borrowed from Internet programming into the realm of distributed computing. As a consequence, users are provided a wider breadth of resources, far-reaching access to processes and data, and an environment well-suited for distributed computing, collaborative computing and multi-user interaction. While IceT is in its relative infancy, the additional attributes brought to distributed computing as illustrated by IceT prove to be quite promising.

References

[1] Adam Ferrari, JPVM. Software package and further information is available at http://www.cs.virginia.edu/∼af2j/jpvm.html

[2] G. A. Geist and V. S. Sunderam, *The PVM system: Supercomputer level concurrent computation on a heterogeneous network of workstations*, Proceedings of the Sixth Distributed Memory Computing Conference, IEEE, April 1991, 258-261.

P. Gray and V. Sunderam, *The IceT Framework for Metacomputing*, Document available at http://www.mathcs.emory.edu/∼gray/IceT3.ps

[3] P. Gray and V. Sunderam, *The IceT Environment for Parallel and Distributed Computing*, Submitted to ISCOPE97.

[4] P. Gray and V. Sunderam, *Collaborative Programming using IceT*, In preparation.

[5] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra, *MPI, The Complete Reference*. MIT Press, November 1995.

[6] David Thurman, *JavaPVM*. Software package and further information is available at http://homer.isye.gatech.edu/chmsr/JavaPVM/

[4] For a detailed example invovling the steps involved the distribution and instantiation of an actual IceT collaborative tool, see[5].