The IceT Environment for Parallel and Distributed Computing *

(An Extended Abstract)

Paul A. Gray and Vaidy S. Sunderam
Dept. of Mathematics and Computer Science
Emory University, Atlanta, GA USA
e-mail: `gray@mathcs.emory.edu`, `vss@mathcs.emory.edu`

June 6, 1997

keywords: heterogeneous computing environment, distributed computing, Java, process mobility, parallel programming environment

1 Introduction

Parallel programming, as it pertains to distributed computing over local networked workstations, has reached a solid level of maturity in recent years. Coupled with recent and considerable gains in processor performance and network capabilities, parallel environments consisting of clustered workstations remain viable platforms for high-performance computation. These recent gains in hardware capabilities have evolved hand-in-hand with significant developments in software tools, programming languages and programming methodologies — most notably, the languages and paradigms associated with Internet programming.

For the most part, the respective programming models, tools and environments associated with Internet programming and parallel, high-performance distributed computing have remained detached from one another. The focus of the IceT project has been to bring together the common and unique attributes of these areas. The result is a confluence of these technologies and a parallel programming environment with several novel characteristics. This IceT substrate provides an initial message, process, and data passing interface which allows for multiple users, the sharing of architecturally-independent processes and data, and a somewhat arbitrary pool of computational resources. These characteristics are provided by facilitating the use of heterogeneous, inter-networked computers as a means for parallel and distributed computing; extending the current view of distributed computing by providing a more flexible interpretation of the areas of *computational resources* (software and hardware), *data* (input, output, and dynamic, including output *visualizations*), and invoking *users* and *processes*.

Loosely speaking, the resulting confluence of technologies provides users with a parallel, multi-user, distributed programming environment; upon which processes and data are allowed to migrate, be shipped throughout owned and unowned resources, under security measures imposed by the owner of the local resource. This environment consists of a dynamically changing virtual machine which is constructed by assembling multiple hosts or by merging of, or recursive aggregation of existing virtual machines. Upon these combined resources, data *and* processes are mutually accessible, transferable and manipulatable amongst multiple users.

The remaining portion of this paper provides a brief summary describing the advantages of IceT, including those gained by using Java — a highly portable, object-oriented language — to *extend* distributed and parallel programming to widely distributed, multi-user clusters of workstations. Section 2 describes how IceT utilizes the portability of Java bytecode to setup a multi-user collaborative environment, where collaborative tools are uploaded to remote users for subsequent interaction. Section 3 describes how the message-passing substrate of IceT facilitates scientific computing using a distributed master/slave matrix-vector multiplication example, and Section 4 provides insight on performance associated with various aspects of the IceT system.

*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 No. ASC-9527186.

2 Collaborative Distributed Computing with IceT

In Internet programming, users are able to access and interact with processes on far-removed systems. For example, a (local) user can execute a remote process located in an http server's cgi-bin directory or can download a Java applet located on a remote system for local execution and interaction. In the current parallel distributed computing paradigm, processes to be executed must be made "accessible"[1] by the owner of the process and are restricted in their execution to resources on which the user has certain privileges.

In IceT, processes and users of the processes are based upon a more liberal paradigm. In addition to the respective descriptions in the above paragraph, users are allowed to access and execute processes on arbitrary resources in the established environment. Moreover, processes are able to interact with multiple users. For example, the owner of a collaborative tool can *just-in-time* install, or *soft-install*[2] his/her process upon colleagues' resources, whereby the colleagues are able to join in the collaboration and interact with the tool. An example of a user's multiple soft-installation of a an actual collaborative IceT "Chat" tool is shown below in Figure 1. IceT manages the soft-installation of the `IceTChat` tool on two of the systems.

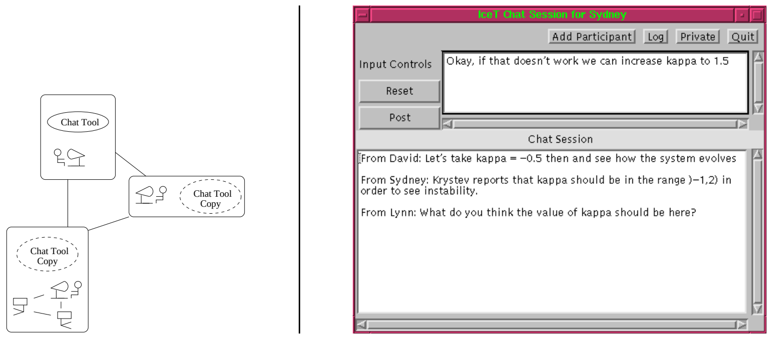

Figure 1: *The* `IceTChat` *tool can be soft-installed on remote resources. The IceT environment depicted to the left above represents the collocation of three separate virtual machines, yet the IceT process – located only at single user's resource – remains portable and accessible to all. IceT provides the message-passing substrate upon which the individual tasks interact and communicate.*

Note: The Java programming language plays a significant role in providing the desired level of process portability and the provisional security imposed on local and remote execution of unowned processes. Java's language-level support of threads and its object-oriented tendencies make it a strong candidate for the thread-based, object-oriented, distributed environment desired in IceT. Further comments and details on specific aspects and consequences of IceT's use of Java can be found in [2].

3 IceT's Suitability for Distributed, Parallel Computing

In many ways, IceT's distributed parallel programming protocols are modeled after PVM. Users can `add`, `delete`, and otherwise configure the resources within their virtual environments and can `spawn`, `kill` and likewise manipulate processes within the environment. In addition to the traditional paradigm, users are able to `merge` with existing virtual environments of others.

For illustrative purposes, consider a master program which is faced with the problem of using slave processes to multiply a matrix $\mathbf{A}$ and a vector $\mathbf{x}$. One way of distributing this problem is to split $\mathbf{A}$ by rows and to pass these sub rows of $\mathbf{A}$ and the entire vector $\mathbf{x}$ 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 message-passing

[1] Placed in appropriate directories based upon architecture on the respective file systems, for example.

[2] Upon termination of processes which have been "uploaded" to remote hosts, the classes associated with the process are garbage-collected by the Java Virtual Machine and no longer reside on the remote system. Thus, the "soft-installation" terminology alludes to this feature of just-in-time installations.

functions such as **`send, recv,`** etc. In this example, the master and three slaves will be responsible for the distributed multiplication (below, left column).

```
import IceT.*;
import java.util.*;

public class MatrixMultiply extends TaskProtocols {
  final static int NUM_SLAVES = 3;
  final static int NUM_WORKERS = NUM_SLAVES+1;
```

. . . continued in the next column . . .

```
try {
  TaskElement myTaskElement = IceT();
  TaskElement[] taskArray = new TaskElement[NUM_SLAVES];
  int numSpawned = spawn("Slave",NUM_SLAVES,taskArray);
```

The program initializes its IceT environment (above, right column), and spawns `NUM_SLAVE` copies of the `Slave` program (described subsequently). Problem parameters, the slave's portion of **A** and the entire vector **x** are packed and sent to the respective processes (below):

```
Buffer bufferx = new Buffer(),bufferA = new Buffer();
int block = (int) SIZE/NUM_WORKERS;
   /* Pack the entire vector */
bufferx.pack( x );

for (int k =0; k<NUM_SLAVES; k++){
  bufferA.reset();
  bufferA.pack(SIZE);
  bufferA.pack(NUM_WORKERS);
```

```
    /* Pack up the slave's A contribution */
   for (int i = 0; i< block; i++) {
     int row = k*block;
     bufferA.pack( A[row+i] );
   }
    /* Send off the information to the slave */
  Id taskId = taskArray[k].myId;
   send(bufferA,taskId,888);
   send(bufferx.clone(),taskId,889);
 }
```

The master program calculates the remaining portion of the matrix-vector multiplication while the slaves compute their results. Finally, all the slaves' computations are collected by the master and the result is processed:

```
/* Reap in the results from the slave */
/* Ax = bVector (a double[SIZE]) */
for (int k = 0; k<NUM_SLAVES; k++) {
  b = recv(777);
  try {
     System.arraycopy(b.unpackDouble(SIZE),0,block,
                      bVector,k*block,block);
  }
```

```
   catch (ArrayIndexOutOfBoundsException aioobe) {
      LogException(aioobe);
   }
   catch (ArrayStoreException ase) {
      LogException(ase);
   }
 }
```

The Slave Program 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:

```
import IceT.*;
import java.util.*;

public class Slave extends TaskProtocols {
```

Code snippet removed here. . .

```
     Buffer b;
     b = recv(888);
              /* Unpack the problem parameters */
     int size = b.unpackInt();
     int divisions = b.unpackInt();
```

```
int rows = (int) size/divisions;
          /* Allocate matrix variables */
double[][] A = new double[rows][size];
double[]   x = new double[size];
double[][] C = new double[rows];
          /* Unpack the partial matrix, A */
for(int i = 0; i < rows; i++)
  for (int j=0; j < size; j++)
    A[i][j] = b.unpackDouble();
          /* Unpack the full vector, x */
b = recv(889);
x = b.unpackDouble(size);
```

The slave then computes its respective matrix-vector multiplication (code omitted), packs up the results, and returns them to the master program:

```
b.reset();
b.pack(x);
/* Return results to the master program */
send(b,myTaskElement.mySpawnHostId,777);
```

Remark: The matrix-vector multiplication example given above illustrates IceT's adherence to the traditional paradigm for distributed computing. There are, however, aspects of the above example which are unique to IceT. One unique and noteworthy aspect is that copies of the `Master` and `Slave` programs need not be present on each of the resources in the IceT environment. IceT manages the task of providing missing processes to resources. Furthermore, through Java's platform independence, a separate compilation for each architecture involved in the conglomerate environment is not required.

4 Consequences of Java — Performance Comparisons

The status of Java as a scientific programming language has been the subject of debate since Java's early release. Consequently, the question of performance of such a system so dependent on Java is quite valid.

Table 2 gives some indication of how the IceT programming environment compares with PVM implementations of equivalent problems. The results depicted in Table 2 are based upon a simple extension of matrix-vector multiplication program listed above to compute a *matrix-matrix* multiplication problem.

# processes computations were distribued over:	*Java-Based Compute Engine supported by the IceT Environment*	*C-based Compute Engine supported by the IceT Environment*	*C-based Compute Engine supported by the PVM3.3 Environment*
1	291.796	41.480	37.134
2	161.589	45.116	24.608
4	87.044	28.920	16.575
5	72.927	26.354	16.506
8	50.695	21.287	16.272

Figure 2: *Benchmarks (in seconds) for computation of multiplying two 400x400 matrices. The distributed environment for this computation consisted of a cluster of Sun Sparc20 workstations.*

The categorization above is as follows:[3] The first column of times (the second actual column) represent times required for the distributed multiplication of two 400x400 matrices using Java-based Master and Slave programs. The next column shows the performance when the IceT substrate is used for task creation and communication but the actual computations are performed using `C`-based Master and Slave programs. The last column represents times for the equivalent `C`-based Master and Slave programs executed under PVM 3.3.

The second column requires some additional comment. One of the avenues in which IceT is evolving in is in a utility capacity. Specifically, in the near future, IceT will serve as a conduit over which system-*dependent* code is ported. This would mean that computationally-intense parallel processes already written in `C` or `FORTRAN` (to obtain significant performance gains over Java, as seen when comparing the first two columns of times above) could be ported amongst appropriate architectures within the extended IceT environment, where execution on remote, unowned systems would be contingent upon the security restrictions imposed by the local owner of the resource.

5 Future of IceT

The future of IceT looks quite promising. Recent improvements to the JDK-1.1 (the Java Native Method Interface specification [3], the Remote Method Invocation specification[5] and Object Serialization specification[4] to name a few) have immediate implications on portability and access of processes within IceT. Implications in the areas of active agents and active networks utilizing IceT are also being pursued. Issues associated with security, performance, load balancing, fault-tolerance, and the implications of nomadic virtual machines are also being investigated.

References

[1] 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.

[2] P. Gray and V. S. Sunderam, *IceT: Distributed Computing and Java.* Presented to the ACM 1997 Workshop on Java for Science and Engineering Computation. To appear in Concurrency, Experience and Practice.

[3] *Java Native Method Invocation Specification*, Sun Microsystems, Revision 1.1, May, 1997. ftp://www.javasoft.com/docs/jdk1.1/jni.pdf

[4] *Java Object Serialization Specification*, Sun Microsystems, Revision 1.3, February 10, 1997. ftp://www.javasoft.com/docs/jdk1.1/serial-spec.pdf

[5] *Java Remote Method Invocation Specification*, Sun Microsystems, Revision 1.4, February 10, 1997. ftp://www.javasoft.com/docs/jdk1.1/rmi-spec.pdf

[3] An additional column, showing times for a pure-Java implementation using just-in-time compilation will be provided as soon as a complete just-in-time compiler is made available for the JDK-1.1