High Performance Computing for the Masses

Mark Clement, Quinn Snell, and Glenn Judd

Computer Science Department
Brigham Young University
Provo, Utah 84602-6576
(801) 378-7608
{clement, snell, glenn}@cs.byu.edu
http://ccc.cs.byu.edu

Abstract. Recent advances in software and hardware for clustered computing have allowed scientists and computing specialists to take advantage of commodity processors in solving challenging computational problems. The setup, management and coding involved in parallel programming along with the challenges of heterogeneous computing machinery prevent most non-technical users from taking advantage of compute resources that may be available to them. This research demonstrates a Java based system that allows a naive user to make effective use of local resources for parallel computing. The DOGMA system provides a "point-and-click" interface that manages idle workstations, dedicated clusters and remote computational resources so that they can be used for parallel computing. Just as the "web browser" enabled use of the Internet by the "Masses", we see simplified user interfaces to parallel processing as being critical to widespread use. This paper describes many of the barriers to widespread use and shows how they are addressed in this research.

1 Introduction

Supercomputers and clusters of computers have long been used by parallel processing experts to solve computationally demanding problems. Users from other fields often find parallel application development and use too complex and confusing. Five years ago the Internet was a realm for a few technical people who could deal with remembering Internet addresses and obscure command line driven programs. Internet Web Browsers changed the face of the Internet and allowed non-technical people access to a wealth of information. This research intends to change the face of parallel processing in much the same fashion.

Imagine a typical evening at a major university or corporation, between the hours of midnight and 8am. The offices are all closed and thousands of computers are sitting idle. At the same time, researchers in several departments are waiting for results from applications that have been running for months on a single machine. Several avenues of research are unexplored because it takes so long to complete computations on a single machine. These applications could take advantage of a large number of processors if they were parallelized and the computing resources were available. This research proposes a way to make idle

computing resources and dedicated clusters available for non-computer-experts (the masses) to use.

Barriers to widespread use of parallel computing include:

- Difficult installation of system software on cluster nodes. It is often necessary to have a development environment and extensive system understanding in order to compile and link applications for use on a compute node. Recently the creation of distributed/parallel programs has been eased somewhat by developments such as MPI[8], PVM[9], and CORBA[15]. These standards have allowed parallel/distributed programs to be written that will function on several different platforms in both tightly and loosely coupled environments. If you are using MPI or PVM, the ".rhosts" file must be correctly configured and other system resources must be manipulated in order to utilize the system. The names of available machines must be known and configured into the application or execution environment. The DOGMA system described here relies only on a screen saver and an installation of Java on processors. No other compilation or configuration must be performed. Since most workstations will have Java installed for the web browser, all necessary configuration can generally be performed with an access to a web page.
- Heterogeneous computing environments can also complicate the use of clusters. Runtime libraries, differing processor architectures, obscure security configurations and networking connectivity can discourage naive users from attempting configuration. Separate binary directories must be maintained for each processor architecture, and applications must be compiled several times in order to create binaries for all candidate architectures. In many cases it is difficult or impossible to determine the cause of these problems from system error messages. Because the DOGMA system is based on Java, most of these problems are not present. Any machine with a Java Virtual Machine can run Java Byte code so only one version of executables must be maintained.
- Selecting which cluster nodes will be involved in the computation can also be difficult for non-computer specialists. The DOGMA system includes scheduling software to determine optimal placement. The user need not participate in determining which processors, or how many processors should be involved in the computation
- Users have traditionally not wanted other users to have access to their machines because of the security risks involved. DOGMA provides two answers to this problem. The security measures inherent in the Java sandbox protect the machine and local file system by not allowing access and encrypting messages. Secondly, the DOGMA system allows a user to set up a group of machines that can only be used by his own applications. The researcher would then be protected from any outside use of his machines.

Although this work is still in progress, we have received enthusiastic support from researchers in Statistics, Zoology and Rendering groups. Each of these groups can use large quantities of processor cycles, but is prevented from making progress in their research area because of the limitations of a single processor.

These researchers are aware of efforts in their fields to use, parallel processing, but do not have the technical expertise to set up clustered systems. It is difficult for them to justify the expense involved in developing expertise in their groups when the benefits are hard to quantify. With the widespread use of high performance sequential processors, many new applications have been developed that have demanding processing requirements. Clusters present a powerful option for these applications if current barriers to implementation can be overcome.

2 Applications

Several application areas have been targeted for initial work with DOGMA. These applications require months or years to complete on available machines. Three statistical applications are being implemented to analyze large insurance databases. Current applications are unable to process more than approximately 30,000 individual records. Several million records are available which can not be analyzed by current technology. Although the Statistics department is reluctant to allow an installation of PVM or MPI, the are enthusiastic about using the DOGMA screen saver. Most system administrators are convinced that the Java sandbox will protect their machines from modification and their installation procedure does not have to be modified to allow DOGMA to run.

Phylogenetic Inference requires significant computational resources. Researchers in the Zoology department at Brigham Young University (BYU) are currently working with tens of individual DNA sequences. Analysis requires months of compute time even on these small data sets. They would like to experiment with hundreds of sequences and require faster turn-around time in order to effectively test new algorithms. Researchers speculate that they may be able to determine which cancer drugs to use for particular patients by examining DNA. They are also involved in analyzing AIDS virus DNA to determine epidemiological information. In both cases the research is being stymied by the lack of computing power.

The College of Fine Arts is currently operating a laboratory of high-end graphics workstations for use in animation and rendering. The workstations are often not available for student use due to the long rendering times when the machines cannot be used in an interactive mode. The DOGMA rendering application will free up these machines for modeling and visual development. The rendering stage will be performed by a DOGMA application in much less time, thus allowing students faster turn-around time and more appropriate use of resources. The resulting animation sequences will be of higher quality because the students will have more opportunities for interactive refinement.

Once the DOGMA system is being used widely, our research can focus on optimizations and more efficient resource allocation and utilization. These algorithms are important and this will be the first opportunity to investigate large-scale use of idle computational resources. Once we have completed evaluations at BYU, we feel confident that widespread use will continue throughout

the Internet in these specific application areas. Additional applications should be possible once the "Masses" view parallel processing as an attainable technology.

3 DOGMA

The DOGMA system has been designed to take advantage of idle computing cycles as well as dedicated clusters of computers. Figure 1 shows a screen shot of a DOGMA rendering application. The user selects the application type and the specific application from the tree diagram shown on the left. When the application is selected, the user clicks to select the input file shown in the lower left corner. The application requirements are then shown in the lower right box. The application is run by clicking on the execute button. At this point the DOGMA system launches the program on available nodes. If additional machines become idle, their screen savers may contact the DOGMA system and become involved in the computation. The only requirement placed upon the user is some familiarity with the application and its inputs. Other applications may ask the user to simply cut and paste input files into the DOGMA interface or access input files on a web server and via a URL.

DOGMA is built using Java Remote Method Invocation (RMI) as a foundation. RMI allows Java objects to interact with other Java objects residing in separate Java Virtual Machines (JVM's); however, the programmer is required to handle issues such as object location at a low level. DOGMA's runtime system is known as the Distributed Java Machine (DJM) layer. It allows several JVM's to act as one distributed machine while freeing the programmer from needing to worry about the location of objects, the names of nodes, the location of nodes, etc. This layer also allows application binary code to be served via an arbitrary number of http servers, and eliminates the need for applications to be installed on local file systems.

DOGMA has a decentralized architecture divided into units known as "configurations". Each configuration is managed by a "configuration manager" which is capable of linking to other configuration managers in a hierarchical fashion. Configuration managers may optionally allow other configuration managers to join them and use their local resources. For instance in Figure 2 configuration manager A can utilize the resources of B but the reverse is not true as indicated by the unidirectional arrow. However, configuration manager B and C can both utilize each other's resources as indicated by the bi-directional arrow. Configuration managers periodically exchange information such as total number of nodes and total number of nodes. This information then propagates through the hierarchy.

As an example, consider Figure 2. An arrow from one configuration manager to another indicates that the source configuration manager may run remote jobs on the destination configuration manager. Note that configuration managers never see the global view. So from configuration manager A's point of view there is only one configuration manager attached to it, but this configuration manager

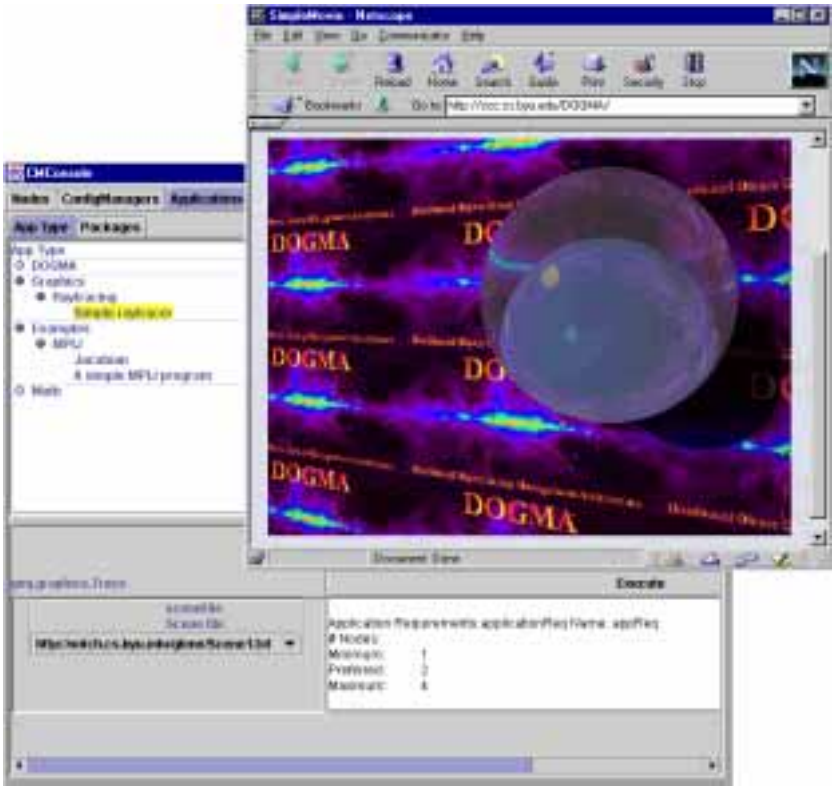

Fig. 1. DOGMA Application Interface

has 35 nodes which is the sum of all nodes downstream in the configuration graph.

Within each configuration three types of nodes may contact the configuration manager to join the system:

- Dedicated nodes - these nodes have access to DOGMA source code and are launched from the configuration manager's system console.
- Browser-based nodes - these nodes join the system by "browsing in" to a specific web-page served by the configuration manager.
- Screen-saver based nodes - these are nodes which have the DOGMA screen saver installed on them, and join the system automatically when they are idle.

Also within each configuration are "code servers" which are http servers used to distribute binary code for a given set of applications; they allow DOGMA to function with no shared file system. Optionally, "data servers" may be added to the configuration to serve application data files.

When nodes join a configuration they sign a "contract" with the configuration manager informing the configuration manager of the duration for which they will

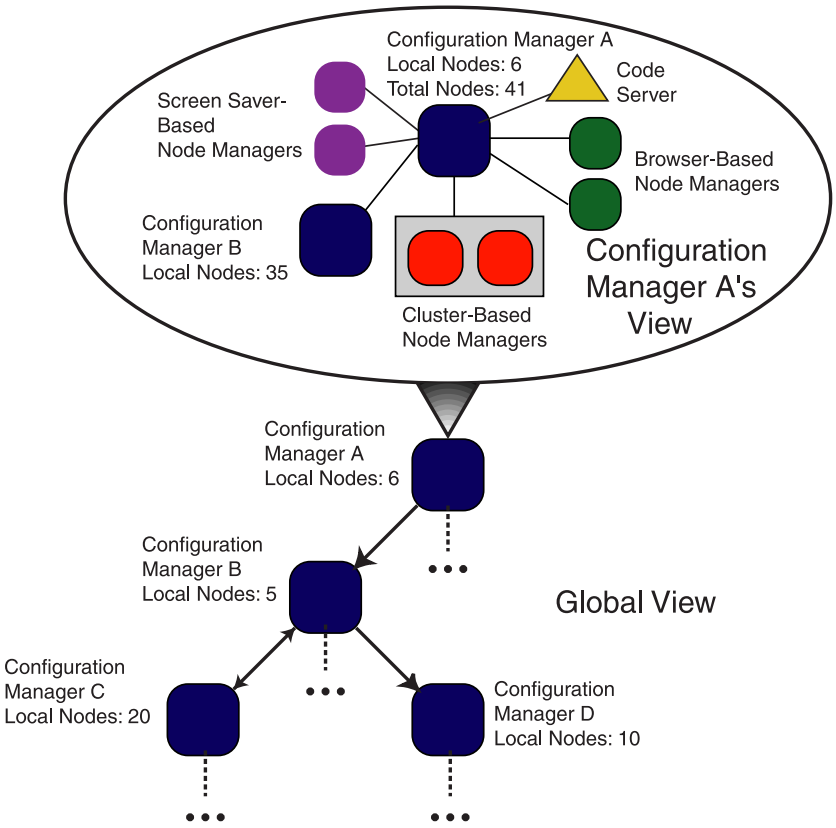

Fig. 2. DOGMA Architecture

stay in the system. This prevents the configuration manager from scheduling long running jobs on transitory nodes. Nodes also inform the configuration manager of certain attributes such as processing power.

To improve application performance and enhance usability, applications have an associated definition file which informs DOGMA of the application's system requirements and startup arguments. Application requirements may consist of the number of nodes, processing power required, networking connection required, whether or not the application allows a varying number of nodes, how long the application expects to run, etc.. Startup arguments free the user from the need to remember which arguments are required by which application, and can be simple text entries, a list of choices, etc..

When an application is run, the system attempts to match the application with a set of nodes which meet the application's requirements. If it cannot find a set of nodes locally, it may (depending on the application requirements) look remotely for nodes to run the application on. The information exchanged peri-

odically by configuration managers helps them make intelligent decisions about where to remotely run a job.

4 Writing DOGMA Code

DOGMA is programmed using one of three programming API's: MPIJ, Distributed Object Groups, or Dynamic Object Groups. MPIJ is DOGMA's implementation of the MPI standard (we are working with other members of the Java Grande Forum on a formal proposal for MPI bindings). Distributed Object Groups allow programs to be written which will automatically partition and reassemble data. Dynamic Object Groups allow master/slave style programs to be written in a very simple fashion which can scale to large numbers of nodes where the number of nodes is allowed to vary at runtime.

4.1 Writing DOGMA Code

Writing Distributed Object Group programs is fairly simple. As in RMI, an interface file must be written defining which methods in a given object (or object group) may be invoked remotely. Upon group method invocation, arguments of type Partition or one of its subclasses are automatically partitioned by the system while other arguments are simply sent to every node in the group. For example:

```
interface SkelElement extends Element {
        public OneDArrayFloatPartition
        addFloatToArray(
        OneDArrayFloatPartition myPartition,
        float floatToAdd)
        throws RemoteException;
}
```

In this case, addFloatToArray is defined to accept a Partition data structure, myPartition, that will contain only the processor's required subset of the data. Group elements then extend the element base class for the data configuration scheme to create the corresponding implementation of the distributed object. A master (a class which has a distributed object group as a member) may then instantiate a distributed object group as follows:

```
skelGroup =
          new SkelElementGroup("my.pkg.SkelElementImpl",
          minNodes, preferredNodes, maxNodes,
          new OneDArrayController(),
          DistObjectGroup.CCFG_NONE);
```

This example uses the OneDArrayController to allocate a one-dimensional array and automatically perform the data distribution. Methods may then be invoked on the entire group with their arguments being partitioned if they are of type Partition or one of its subclasses. As the invocation is asynchronous, a handle to the invocation is returned:

```
handle = dog.addFloatToArray(data, 7.0f);
```

This call invokes the addFloatToArray method on all available processors, sending each processor a piece of the array. The programmer then uses the handle to request the results of the method. The programmer may request that the results be automatically reassembled, or that an array of results be returned. The statement

```
partition = (OneDArrayFloatPartition)
        handle.getAssembledReturn();
```

causes the results to automatically be reassembled into the array `partition`.

Group element communication is similar to other parallel systems. Note that objects may be sent as easily as primitive data as in the following statement. The object `histogram` is broadcast to all participating objects.

```
sendToAll(histogram);
Object[] otherHistogram = receiveFromAll();
```

The Distributed Object Group paradigm presented here provides a high level abstraction for parallelism that also achieves high performance.

4.2 DOGMA MPI

DOGMA's Java MPI implementation is based on the Java Grande Forum proposal for Java MPI bindings. MPIJ is a pure Java MPI implementation conforming to these standards. Other implementations use the Java Native Interface to link to native MPI implementations. However, as will be shown in later sections, Java communication performance is equivalent to that of native socket communication.

```
import mpij.*;
public class SampleMPIApplication extends MPIApplication {
        public void MPIMain(String[] args) {
        MPI.init();
        myNode = MPI.Comm_WORLD.rank();
        totalNodes = MPI.Comm_WORLD.size();
        // broadcast an array from node 0 to everyone in COMM_WORLD
        double[] d;
```

```
        d = new double[100];
        MPI.COMM_WORLD.bcast(d, 0, d.length, MPI.DOUBLE, 0);
        ...
        }
}
```

The above code fragment is a simple example of the MPIJ bindings. It allocates an array and broadcasts it to all members of the process group. This code sample is very similar to a C++ MPI implemented code. It is fairly easy to port existing MPI code to this system.

A DOGMA program can be written specifically for the use of distributed object groups or for message passing using the Java MPI calls. It is also possible to combine the two paradigms. Porting of MPI programs from C to Java has proven to be very simple, and the performance obtained is promising. A finite element program was recently ported from C to Java using our MPI calls. The only changes necessary were in the method calling conventions of Java and in the allocation of arrays.

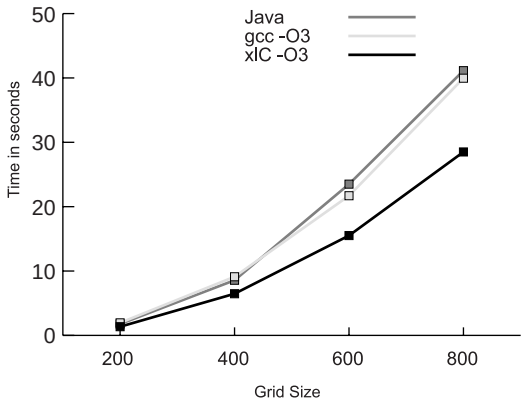

Fig. 3. Java Performance Comparison

5 DOGMA Performance

Despite the advances in source code level compatibility, traditional distributed programming environments require machine specific compilation and operating system customization. This makes the inclusion of a new architecture into an existing distributed computing platform non-trivial. With the advent of Java, code distribution is now relatively simple. Unfortunately, many have come to

view Java as incapable of the performance required for scientific applications. However, Java just-in-time compiler performance is quickly approaching that of optimized C. Figure 3 shows the performance of a finite element algorithm on a single node of an SP2. The performance of the Java code is nearly indistinguishable from the fully optimized gnu gcc compiled executable. The custom IBM xlc compiler achieves higher performance, but scheduled improvements in JIT compiler technology promise to close this gap. The DOGMA system provides the portability of Java as well as high performance for heterogeneous parallel computing.

DOGMA is portable across all Java virtual machines that fully support the JDK version 1.1.3 or greater. We have successfully run applications on large groups of heterogeneous nodes with no source code modification. The applications include: matrix multiplication, image FFT, image histogram equalization, finite differences, and ray tracing to name a few. The number of nodes included in the system has ranged from 1 to 40 (including several dual-processor machines) with the nodes residing in 4 different domains. The system has also incorporated both browser-based nodes, which were able to execute applications and donate CPU power, as well as nodes running the DOGMA screen saver to donate idle CPU time to DOGMA. In addition, DOGMA is capable of seamlessly harnessing the power of multiprocessor machines by assigning multiple objects to the machine. Local objects communicate via direct method invocation rather than through RMI.

5.1 Computation Performance

The results shown previously in Figure 3 demonstrate similar findings for a single node of the IBM SP-2. The graph shows execution times for a finite differences code with varying size grids. In all grid sizes, JIT compiled code is comparable to the performance of fully optimized `gcc` compiled code. There is still room for improvement, as is demonstrated by the performance of `xlC` compiled code. Because of dynamic runtime optimizations, many predict superior performance as JIT compilers mature.

The previous results are not always typical. JIT compiled code that is heavily object oriented does not currently provide similar performance. JIT compilers need more improvements handling object oriented code, but clearly they have the potential for high performance computing. The Hotspot JIT compiler from Sun promises to solve some of these performance issues by using adaptive compilation techniques pioneered by the Self project at Stanford [4, 11].

5.2 Communication Performance

High communication performance is essential in a supercomputing environment. In Java, communication between nodes can take place via two mechanisms: sockets and Remote Method Invocation (RMI). The DOGMA environment uses both schemes. Group method invocation is accomplished via RMI, and inter-node message passing using the MPI library is performed using Java sockets.

Most messages in scientific codes consist of floating point data that is either double or single precision. Java communication is based on the InputStream and OutputStream classes; neither of which support communication with data types other than bytes. Therefore, in order to send an array of floating point data, the data must be converted into a byte array. Java is a strongly typed language and, as such, does not allow type casting of pointers. Unlike C programs which allow a floating point array to be directly treated as an array of characters, each Java data element must be type cast and copied into a byte array to be transmitted. This technique is termed data marshaling, and adds significant overhead.

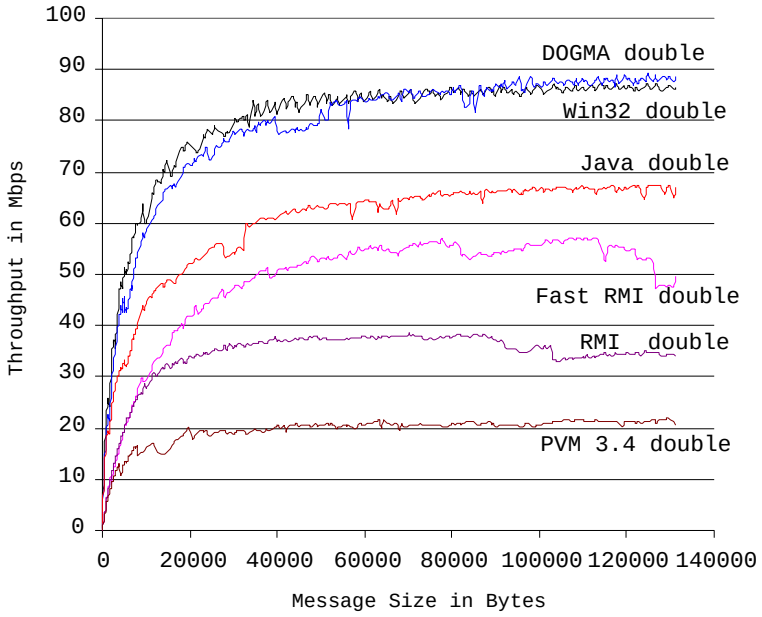

Fig. 4. Native communication versus Java for floating point data.

Figure 4 clearly shows the overhead due to data marshaling. However, the graph also demonstrates that significant performance increases are possible by performing the data marshaling in native code. The "Fast RMI double" line on the graph was obtained using native data marshaling. This simple routine is linked to the Java code using the Java Native Interface (JNI). This routine could be directly added to the Java virtual machine or incorporated into the InputStream and OutputStream classes to allow for communication of other data types. The graph also shows the comparison of message passing performance between two DOGMA nodes versus communication between two PVM nodes. Both tests were run on the same two workstations. PVM uses native sockets as a communication medium, while DOGMA communication is based on Java

sockets and the JNI marshaling methods. Amazingly, DOGMA communication performance is better than PVM.

Even DOGMA latency (160 microseconds) is much better than that of a PVM message (1147 microseconds). Such a result implies that communication intensive applications will fare better in the DOGMA environment. This hypothesis will be demonstrated later in this paper.

5.3 Aggregate Performance

The above results are meaningless unless they translate into aggregate cluster performance for parallel applications. In this section, we demonstrate DOGMA application performance in comparison with PVM and MPI applications. In each case, the target applications were coded using the same algorithm for each environment. For comparison purposes, the applications were also run on the cluster using `mpich` on the Linux operating system. We also ran the applications using the WinMPICH message passing system. However, the system is still very unstable, and the results were very poor. Thus we chose the Linux system as our base-line. The applications of choice were: 2 dimensional FFT, Gaussian elimination, and a finite differences code. The applications were chosen for their varying communication demands.

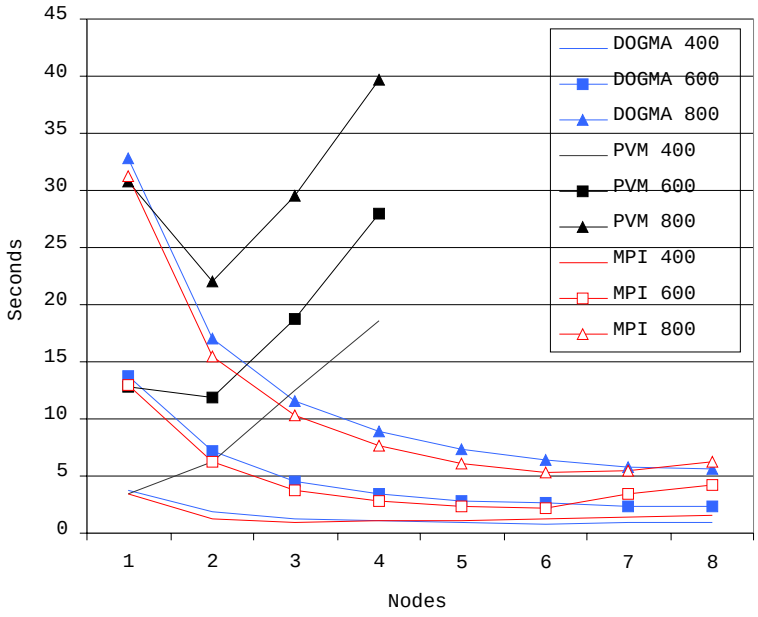

Fig. 5. Performance Comparison for Gaussian Elimination

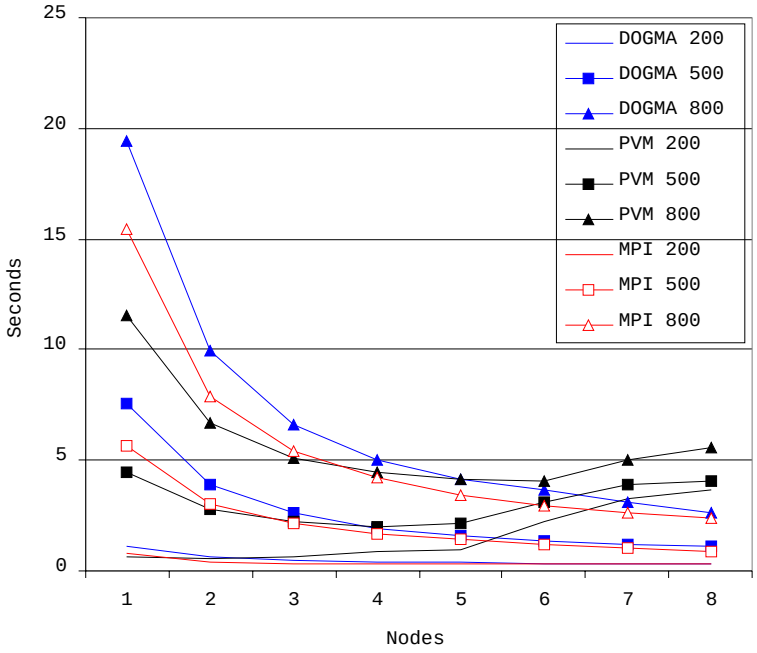

Fig. 6. Performance Comparison for Finite Element Calculation.

The application performance graphs in Figures 5 and 6 show the comparative performance results for the Gaussian elimination and finite element calculations. Each of the applications require successively less communication. As the communication demands decrease, the computational performance of the individual nodes is more important. In these cases, DOGMA application performance is better than PVM performance for larger numbers of nodes, but does not exceed Linux MPI application performance. It does, however, approach that level of performance.

6 Related Work

Java's platform independent/Internet-centric nature make it natural for use in distributed computing. This has lead several groups to begin research on distributed/parallel programming with Java. The following section reviews these other avenues of research and differentiates DOGMA from these existing systems.

The JavaPVM[19], JPVM[7] and IceT[10] message passing schemes do not provide automatic data distribution nor do they provide the object group method invocation found in DOGMA. DOGMA supports traditional message passing via an MPI implementation, but features object-group oriented programming as

its preferred paradigm. Linda derivatives such as Jada[18], JavaSpaces[12] and Javelin[5] have not been able to achieve comparable performance. Java/DSM[20], Spar[13], JavaParty[17] and Charlotte[3] provide a shared address space, while DOGMA seeks to make the developer aware of the distributed nature of the program in order to increase performance.

Systems such as JET[16], SuperWeb[1]/Javelin[5], IceT, and Charlotte are similar to DOGMA in that they incorporate Web browser functionality; however, DOGMA also adds cluster based computing in order to utilize groups of nodes for which a user has login privilege. DOGMA also makes use of signed applets to allow for direct communication between browser-based nodes rather than requiring browser-based nodes to communicate only with a web server. In addition, DOGMA adds screen saver based browsing to allow nodes to automatically donate idle CPU time rather than requiring explicit user donation of CPU time.

More recent implementations[14] such as mpiJava[2] link native MPI implementations to Java via the Java Native Interface. These efforts are also concentrating on pure Java implementations. Condor[6] is a distributed batch system that provides for a hierarchical integration of clusters of workstations. It is suited well to batch processing and has extensive load balancing features. DOGMA is designed for non-technical users and does not require the installation any software.

DOGMA seeks to do more than simply provide a Java version of a message passing standard. Rather, DOGMA has three main goals:

- The creation of a distributed computing platform which can harness the power of clusters of computers as well as anonymous idle computers in both Internet and intranet settings.
- Providing mechanisms for easily writing programs capable of efficiently using the power of such a system.
- Providing a point and click interface that non-technical users can use. DOGMA is usable by people who don't feel comfortable with writing any code or performing any task other than accessing a web page.

7 Conclusions

The DOGMA system allows idle machines to be incorporated into supercomputing clusters irrespective of their CPU or operating system. Since use of these clusters is much less expensive than traditional supercomputing platforms, more researchers and average users can have access to high performance compute resources. Applications such as phylogenetic inference, statistical analysis and large database indexing (which often require much more compute power than is available on a single system) can be approached using DOGMA clusters.

Though there is much future research to be done, DOGMA has proven to be very capable of enabling clusters of computers and individual anonymous computers to act as a single powerful networked computing entity. DOGMA

volunteer facilities also provide an excellent means to increase workstation utilization. Object migration is currently being incorporated to allow for increased performance, and useful inclusion of transitory nodes in a large number of applications. Other issues such as security and fault tolerance enhancements are also being examined.

The availability of significant CPU resources will allow more users to solve complex optimization and simulation problems in a reasonable amount of time. The DOGMA system provides a significant improvement over existing parallel environments. The point-and-click interface will open up new scientific frontiers for researchers in other disciplines.

See the DOGMA homepage at for more current information:
`http://ccc.cs.byu.edu/DOGMA/`.

References

1. A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman. Superweb: Towards a global-based parallel computing infrastructure. In *Proceedings of the 11th International Parallel Processing Symposium*, April 1997.
2. M. Baker, B. Carpenter, S. H. Ko, and X. Li. mpijava: A java interface to mpi. In *First UK Workshop on Java for High Performance Network Computing, Europar*, September 1998.
3. A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff. Charlotte: Metacomputing on the web. In *Proceedings of the ISCA International Conference on Parallel and Distributed Computing*, 1996.
4. C. Chambers. *The Design and Implementation of the Self Compiler, an Optimizing Compiler for Object-Oriented Programming Languages.* PhD thesis, Stanford University, 1992.
5. B. O. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu. Javelin: Internet-based parallel computing using java. In *ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation*, June 1997.
6. D. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of condors : Load sharing among workstation clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.
7. A. Ferrari. Jpvm. Technical report, http://www.cs.virginia.edu/~ajf2j/jpvm.html, 1997.
8. M. Forum. Mpi: A message-passing interface standard. Technical report, University of Tennessee, June 1995.
9. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM 3 user's guide and reference manual. Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, September 1994.
10. P. A. Gray and V. S. Sunderam. IceT: Distributed computing and java. In *Proceedings of the ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation*, June 1997.
11. U. Hoelzle. *Adaptive Optimization for Self: Reconciling High Performance with Exploratory Programming.* PhD thesis, Stanford University, 1992.
12. Javasoft. Javaspaces. Technical report, http://chatsubo.javasoft.com/javaspaces/, 1997.
13. H. J. S. Kees van Reeuwijk, Arjan J.C. van Gemund. Spar: A programming language for semi-automatic compilation of parallel programs. In *ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation*, June 1997.
14. S. Mintchev and V. Getov. *Recent Advances in PVM and MPI.* Springer Verlag, 1997.

15. OMG. The common object request broker: Architecture and specification. *2.0 ed.*, July 1997.
16. H. Pedroso, L. M. Silva, and J. G. Silva. Web-based metacomputing with jet. In *Proceedings of the ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation*, June 1997.
17. M. Philippsen and M. Zenger. Javaparty - transparent remote objects in java. In *Proceedings of the ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation*, June 1997.
18. D. Rossi. Jada: Multiple object spaces for java. Technical report, http://www.cs.unibo.it/~rossi/jada/, 1996.
19. D. Thurman. Javapvm. Technical report, http://homer.isye.gatech.edu/chmsr/JavaPVM.html/, 1997.
20. W. Yu and A. Cox. Java/DSM: A platform for heterogeneous computing. In *Proceedings of the ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation*, June 1997.