DOGMA: Distributed Object Group Metacomputing Architecture †

Glenn Judd, Mark Clement, Quinn Snell
Computer Science Department
Brigham Young University
Provo, Utah 84602-6576
(801) 378-7608
{glenn, clement, snell}@cs.byu.edu

Abstract

The performance of Java just-in-time compilers currently approaches native C++, making Java a serious contender for supercomputing application development. This paper presents DOGMA–a new Java based system which enables parallel computing on heterogeneous computers. DOGMA supports parallel programming in both a traditional message passing form and a novel object-oriented approach. DOGMA provides support for dedicated clusters as well as idle workstations through the use of a web based browse-in feature or the DOGMA screen saver. This research provides a unified environment for developing high performance supercomputing applications on heterogeneous systems.

1 Introduction

Researchers have long sought to integrate the computational power of heterogeneous clusters of computers in order to achieve fault tolerance, utilization of idle computers and resource sharing. Unfortunately, the complexity of heterogeneous computing has made these goals elusive. The Java programming language has eliminated some of the problems by making high performance applications portable across nearly all architectures. However, the Java environment does not directly support supercomputing. The Distributed Object Group Metacomputing Architecture (DOGMA) provides system management, data distribution and messaging facilities for Java based supercomputing applications.

Harnessing the computational power of clusters of heterogeneous computers requires both a hardware system capable of fast, reliable communication as well as a software system capable of allowing applications to make use of large numbers of computers. Recently the creation of distributed/parallel programs has been eased somewhat by developments such as MPI[8], PVM[10], and CORBA[16]. These standards have allowed parallel/distributed programs to be written that will function on several different platforms in both tightly and loosely coupled environments.

Nevertheless, 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. Several significant obstacles still remain for heterogeneous parallel computing. The rest of this paper describes the DOGMA system and how it addresses these problems.

1.1 Distributed Computing with Java

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

The JavaPVM[20], JPVM[7] and IceT[11] 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[19], JavaSpaces[13] and Javelin[4] have not been able to achieve comparable performance. Java/DSM[21], Spar[15], JavaParty[18] and Charlotte[2] 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.

† This research was supported by Sprint.

JAVADC[3], Lilith[6], WebFlow[9], and WWVM[5] provide frameworks for controlling natively coded parallel programming systems. They are functionally orthogonal to DOGMA which seeks to provide a Java-based system that is completely platform independent. The Parallel Java with Global Object Space[14] sytem is object-group based as is DOGMA, but it does not provide for volunteer computing or dynamic class loading. Systems such as JET[17], SuperWeb[1]/Javelin[4] 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. Additionally, DOGMA adds screen saver based browsing.

DOGMA seeks to do more than simply provide a Java version of a message passing standard. Rather, DOGMA has two 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 a mechanism for easily writing programs capable of harnessing the power of such a system.

2 DOGMA

DOGMA is built using Java Remote Method Invocation (RMI) as a foundation. RMI allows Java objects to interact with 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.

The programming API is known as the Distributed Object Group (DOG) layer. It allows method invocation, data partitioning, communication, etc. on large groups of objects without the programmer needing to worry about issues such as the exact number of nodes used.

Also, DOGMA includes a partial MPI implementation written entirely in Java. Although it will probably usually be used in conjunction with Distributed Object Groups, its design is orthogonal to them and only requires the DJM layer in order to function.

Currently, DJM is a centralized system managed by a central entity, DJMManager, which coordinates the actions of all system entities. Each node (JVM) in the system has a NodeManager which coordinates the node's interaction with DJMManager. As in Mentat[12], nodes are organized into families and clusters. Families of nodes are nodes that have a similar configuration and architecture. Clusters of nodes are nodes that are located near each other physically. Unlike Mentat, there is not a single static system configuration database. Rather, the system is configured dynamically.

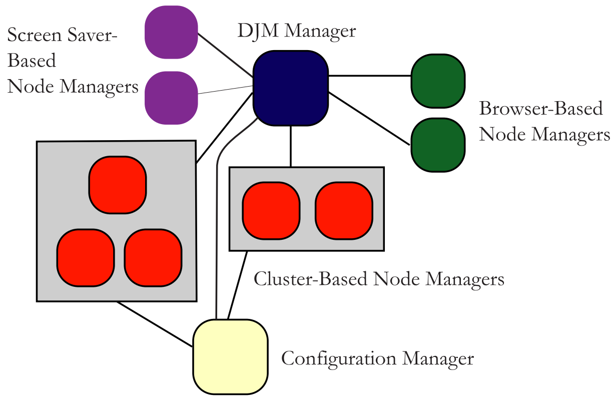

Figure 1: Sample System Configuration

System topology is configured dynamically in several ways. 1) Nodes may be made active or inactive at any time (by an administrator) on an individual node basis, or by cluster. 2) The system topology database may be updated at runtime. This is done by grouping families and clusters into groups known as "configurations" each managed by a ConfigurationManager. When a ConfigurationManager is made active, it contacts DJMManager and informs it of all families, clusters, and nodes that are in the ConfigurationManager's configuration. When a ConfigurationManager is shutdown it removes all knowledge of its configuration from DJMManager. In this way, ConfigManagers provide modularity. They free the DJMManager from needing to know the configuration of the system when it is started. Also, by handling all architecture specific issues, ConfigManagers free DJMManager from any system specific functions. In this way arbitrary clusters of nodes can be added to and removed from the system dynamically. 3) Web browsers can effectively become nodes in the system. This allows users to run applications by simply browsing into the system and running the desired application, or to donate computational power by browsing into the system simply to help. 4) The DOGMA screen saver can be used to add nodes to the system when they become idle.

Distributed object groups allow programs to work with large groups of objects easily and efficiently. A master instantiates a DOG made up of a number of objects which is distributed across the DJM. A DOG is organized according to a data configuration scheme. The scheme defines how data will be partitioned, group elements will communicate, and what type of coordinates are used to define group element position. Through asynchronous group method invocation, the master can invoke methods on the entire group. The master will only block when a return value is requested and unavailable.

Arguments to methods can be partitioned automatically by the system according to a group's data configuration scheme. The speed and load of nodes on which group elements reside can also be considered. DOGMA currently supports partitioning of object data along a single dimension. Data returned by a group method invocation can also be reassembled automatically by the system. Alternatively, an array of results can be returned.

Distributed Object Groups free the programmer from the traditional tasks of computing partition sizes, partition overlaps, element position, etc., and add the power of dynamic partitioning according to system topology and load.

DOGMA also supports group element peer communication. Standard functions include point to point, multi-point, synchronous, and asynchronous send and receive methods.

3 Results

DOGMA has proven to be an effective, easy-to-use parallel programming environment. Performance has been hindered somewhat by just-in-time compilers which either functioned poorly or crashed (i.e. had to be disabled). These problems should disappear as Java matures.

As Java implementations capable of supporting DOGMA have become available (JDK 1.1.3 and greater), DOGMA has easily run on them. 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, 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 has proven capable of seamlessly harnessing the power of multiprocessor machines by assigning multiple objects to them, and allowing local objects to communicate via direct method invocation.

DOGMA applications typically make use of dynamic data partitioning in order to ease programming and increase power. The increased performance from dynamic data partitioning is application dependent, but as an example of the benefits of dynamic data partitioning consider speedups obtained executing a matrix multiplication of 2 400x400 matrices:

- Effect with different speed machines- Configuration: 2 Pentium Pros 200Mhz, 1 Pentium 133 Mhz, 1 Pentium 90Mhz. Result: 33 percent speedup when dynamic data partitioning is used.

- Effect with different loads- Configuration: cluster of HP 9000/778's with one node heavily loaded, one node lightly loaded, and 2 not loaded. Result: 48 percent speedup when dynamic data partitioning is used.

3.1 Application Results

First some cautions: Java is relatively new. Performance is increasing drastically. Consequently, the actual performance obtained is frequently more dependent on the implementation of Java than the power of the machine on which it runs.

The result of this is that while in many instances the actual time required for algorithm completion is quite instructive, it can also be quite misleading. With this in mind some of the results obtained with DOGMA are listed below:

Figure 2 shows time required to complete a matrix multiplication of two square matrices on a cluster of Pentium II 266Mhz workstations runnning Windows NT and connected by Gigabit Ethernet. In all cases,

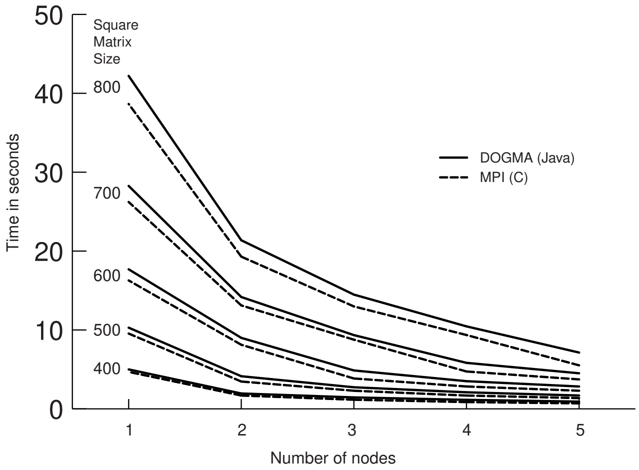

Figure 2: Matrix Multiplication Performance DOGMA vs. Native MPI

the performance of DOGMA is never less than 75% that of native MPI, and is within 95% for smaller numbers of nodes. Also note that both DOGMA and MPI exhibit superlinear speedup as the application working set fits into cache.

Figure 3 shows performance obtained executing a ray tracing application on a hetergeneous set of nodes. Performance predictions for multiple nodes are obtained simply by adding the speed in lines per second of the individual nodes. Note that the predicted times match the actual times observed almost exactly. This indicates that the speedup obtained from using these 10 nodes is almost perfectly linear.

4 Conclusions

DOGMA is a new but promising system for practical distributed programming with Java. DOGMA provides both an effective object-group-oriented programming capability and powerful system configuration abilities such as dynamic code loading, cluster management, web browser participation, and screen saver donation of idle CPU cycles.

Preliminary results obtained with DOGMA show that while Java is still an immature language, it is nonetheless a very promising language for distributed/parallel computing on a large scale. Improvements to the Java virtual machine to allow class reloading, better memory management, and native threading will improve the performance and usability of DOGMA.

Though there is much future research to be done,

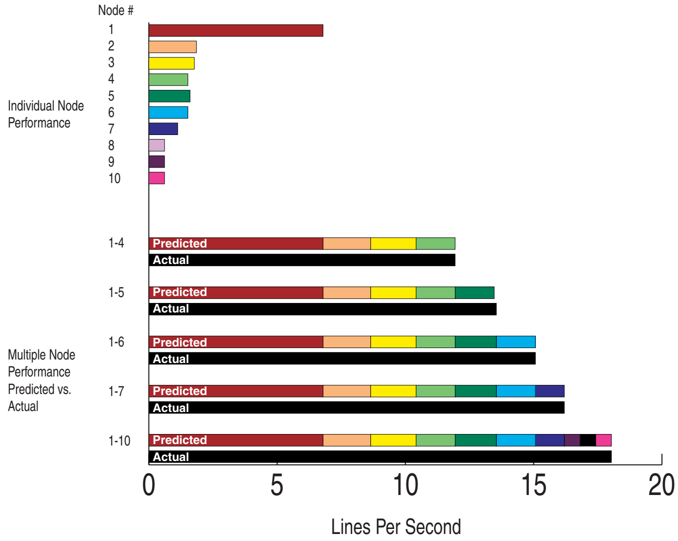

Figure 3: Ray Tracer Performance on Heterogeneous Nodes

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.

5 Future Work

Parallel programming with Java is still in its infancy. Tools such as DOGMA open many avenues for research, and below are listed a few that will be focused on in the near term.

There is much research to be done in the area of object placement for an application. DOGMA currently places objects on nodes which appear to have the most computational power (where power = speed*(1-load)) available. However, there are many factors such as network bandwidth, network latency, and an application's data configuration scheme that are not captured by this model. This work is closely related to designing an efficient object placement algorithm. Future work will investigate more sophisticated dynamic data partitioning algorithms.

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 worked on.

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

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] 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.

[3] Z. Chen, K. Maly, P. Mehrotra, P. K. Vangala, and M. Zubair. Web based framework for distributed computing. In *Proceedings of the ACM 1997 PPoPP Workshop on Java for Science and Engineering Computation*, June 1997.

[4] 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.

[5] K. Dincer and G. C. Fox. Building a world-wide virtual machine based on web and hpcc technologies. In *Proceedings Supercomputing '96*, November 1996.

[6] D. Evensky, A. Gentile, L. Camp, and R. Armstrong. Lilith: Scalable execution of user code for distributed computing. In *Proceedings of the Sixth IEEE International Symposium on High Performance Distributed Computing*, pages 305–314, 1997.

[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] W. Furmanski and G. Fox. Webflow - a visual programming paradigm for web/java based coarse grain distributed computing. Technical report, From handout material at Supercomputing '96, 1996.

[10] 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.

[11] 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.

[12] A. S. Grimshaw. An introduction to parallel object-oriented programming with mentat. Technical Report 91-07, University of Virginia, 1991.

[13] Javasoft. Javaspaces. Technical report, http://chatsubo.javasoft.com/javaspaces/, 1997.

[14] L. V. Kale, M. Bhandarkar, and T. Wilmarth. Design and implementation of parallel java with global object space. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 235–244, 1997.

[15] 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.

[16] OMG. The common object request broker: Architecture and specification. *2.0 ed.*, July 1997.

[17] 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.

[18] 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.

[19] D. Rossi. Jada: Multiple object spaces for java. Technical report, http://www.cs.unibo.it/rossi/jada/, 1996.

[20] D. Thurman. Javapvm. Technical report, http://homer.isye.gatech.edu/chmsr/JavaPVM.html/, 1997.

[21] 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.