DOGMA: Distributed Object Group Management Architecture †

Glenn Judd, Mark Clement, Quinn Snell
Computer Science Department
Brigham Young University
Provo, Utah 84602-6576
(801) 378-7608
FAX (801) 378-7775
{glenn, clement, snell}@cs.byu.edu

Abstract

DOGMA is a new system for object-group-oriented parallel programming using Java. In addition to providing effective object-group-oriented programming, DOGMA also provides powerful system configuration abilities such as dynamic code loading, dynamic system configuration, and web browser participation.

Though relatively new, DOGMA has proven effective at executing parallel applications on a several different platforms across Internet domains. Preliminary results indicate that DOGMA should be able to rival the speed natively coded parallel systems for applications in which Java just-in-time compilation is effective.

1 Introduction

Parallel and distributed computing have long been considered difficult for many reasons: efficient parallel algorithms are usually more difficult to develop than are efficient sequential algorithms, system architectural differences have often limited code portability, and system configuration has often been tedious. Nevertheless, parallel programs continue to yield useful results, and significant numbers of programmers are willing to expend efforts to develop, run, and maintain them.

Fortunately, the development of parallel programs has been eased somewhat be developments such as MPI[8], and PVM[10]. These standards have allowed parallel 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 parallel programming environments require unique binary code in order to function on any given architecture. However, developments currently are underway which stand to change the whole face of distributed and parallel computing.

Fueling these changes is the current rapid growth of the Internet. The number of computers connected to the Internet is increasing at a tremendous rate as Internet has become an integral part of society. With this growth came the development of a language capable of enabling cross platform web programming: Java.

Parallel Computing with Java

Java's platform independent/distributed nature make it natural for use in parallel/distributed computing, and this has lead several groups to begin research on parallel programming with Java. The following list is a

† This research was supported by Sprint.

fairly representative sampling of parallel Java systems that have been developed to date (classified very roughly according to architecture with some systems spanning multiple classifications):

- Message passing - JavaPVM[22], IceT[11]
- Frameworks for controlling/Integrating traditional parallel computing - JAVADC[4], Lilith[7], WebFlow[9]/WWVM[6]
- Distributed shared memory - Charlotte[2], Java/DSM[23], Spar[17], JavaParty[20]
- Remote objects - HORB[13], RMI[15], Voyager[18], JavaParty[20]
- Remote object groups - Parallel Java with Global Object Space[16]
- Linda derivatives - Jada[21], JavaSpaces[14], Javelin[5]
- Browser/Java applet based - JET[19], SuperWeb[1]/Javelin[5], Charlotte
- Collaboratory systems - Tango[3]

1.1 DOGMA

1.1.1 Relation of Previous Work to DOGMA

DOGMA seeks to do more than simply provide a Java version of a message passing standard. Rather, the philosophy of DOGMA is to allow object oriented parallel programming somewhat similar to that offered by Mentat[12], but in a system independent way. Like Mentat, DOGMA attempts to strike a middle ground between ease of programming and system performance.

DOGMA differs mainly from current parallel Java systems in that it is an object-group-oriented system designed mainly for running on clusters of workstations (and MPP support in the future) that also provides dynamic class loading and web browser participation. DOGMA is also designed to have performance reasonably comparable to that of traditional native code message passing systems.

1.1.2 Design Goals

DOGMA's overall design goals are to provide an easy-to-use, object-oriented system with usefully high performance that can perform well in both Internet and intranet settings. DOGMA is targeted towards applications where systems such as PVM and MPI are now used, but adds the capability of harnessing the power of nodes across the Internet. DOGMA seeks to be both a platform for further research into parallel programming with Java, and a tool capable of producing useful results.

Toward this end there are several requirements which DOGMA seeks to meet in both the runtime system and in its programming interface:

Runtime system design features

- Platform independence - This is a fundamental requirement for harnessing the full computational power attached to the Internet. Even within an intranet, platform independence is frequently a prerequisite to obtaining maximum computational power. The natural choice for a platform independent language is Java.
- Dynamic configuration - Within an intranet, systems with static configurations are often awkward to use. In Internet computing, static system configuration is severely limiting.
- Ability to manage clusters of workstations - Many users require the ability to include or disclude clusters of computers on the fly. Something more than running a browser on each node is realistically required for this.

- Web browser participation - The ability to launch clusters of workstations is all that is required in many settings, but not all settings. Web browsers can be very useful to the system in many ways:

  While at times users may only wish to include clusters of workstations in a system, at other times it can be very useful to allow arbitrary nodes from anywhere in the world to be included in the system to donate computational power.

  Web browsers provide an attractive and uniform way for users to interact with the system. A user creating an application can simply browse into the system, run an application, and get results. Subject to security policy, this can be done from anywhere in the world with an Internet connection.

  Web browsers allow the inclusion of nodes in the system without regard to login privilege (subject to DOGMA security).

- Fault tolerance - Some method of fault tolerance is required for any type of Internet system.

- Decentralized dynamic class loading - Application code should not need to be on every node in the system. Instead, application code should be loaded by nodes as needed (via http for instance). In this way application code maintained at nodes used by developers and not by system administrators.

- Performance - The system should take an active role in enhancing application performance through features such as good object placement.

API design features

- Platform independence - This is required for applications for the same reasons it is required for the system. Again, Java is the language of choice.

- Object oriented/Object group support - In addition to being the programming paradigm of Java, object-oriented programming is increasingly the paradigm of choice for programmers today. A very useful extension of this paradigm is the incorporation of object groups capable of group method invocation and other useful parallel programming features.

- Transparency - The API should not require knowledge of the location of nodes on which a program runs or the exact number of nodes on which it runs.

- Performance - DOGMA's API allows programmers to assist in data decomposition. This increases performance without the demands of a pure message passing scheme.

- Dynamic data partitioning - The system should consider node speed and load to dynamically partition data. In DOGMA this is accomplished through cooperation between the runtime system and the programming API.

- Ease of use - The system should be easy to use since developer time is often more costly than machine time.

1.2 Overview

The rest of this paper is organized as follows: Section 2 will discuss DOGMA's architecture. Section 3 will present a sample application. Section 4 will present performance results. Section 5 will present our conclusions, and Section 6 will outline future work.

2 DOGMA

DOGMA is built using Java Remote Method Invocation 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 etc. at a low level.

The Distributed Java Machine (DJM) layer 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 be served via an http server so that applications do not need to be installed on the local file system of every cluster included in the system.

Further, DOG allows method invocation, data partitioning, communication, etc. on large groups of objects without the programmer needing to worry about the exact number of nodes used etc.

Also, an MPI implementation written entirely in Java is currently under development as part of the DOGMA system. 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.

2.1 DJM

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, nodes are organized into families and clusters (Figure 1). Families of nodes are nodes that have a similar configuration and architecture. Clusters of nodes are nodes that are located near each other physically.

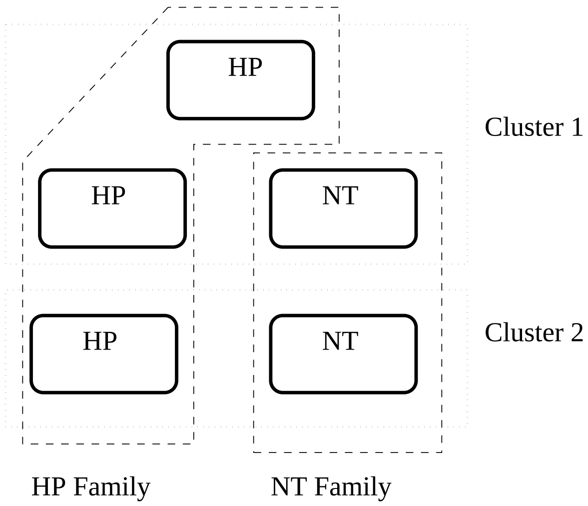

Figure 1:

System topology is configured dynamically in three ways:

- 1 - All nodes in a system do not need to be actively part of the system at all times. Nodes may be made active or inactive at any time (by an administrator). This may be done on an individual node basis, or by cluster.

- 2 - The system topology database may be updated at runtime (see Figure 2). 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.

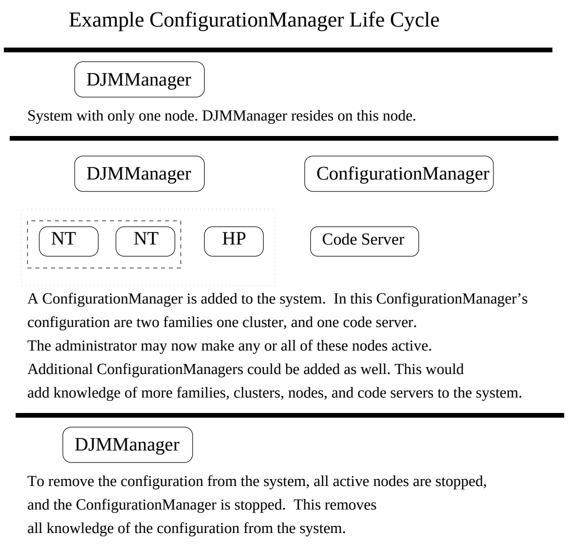

Figure 2:

- 3 - Web browsers can effectively become nodes in the system. Web browsers can load the NodeConApplet which starts a NodeManager running in the browser. 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.

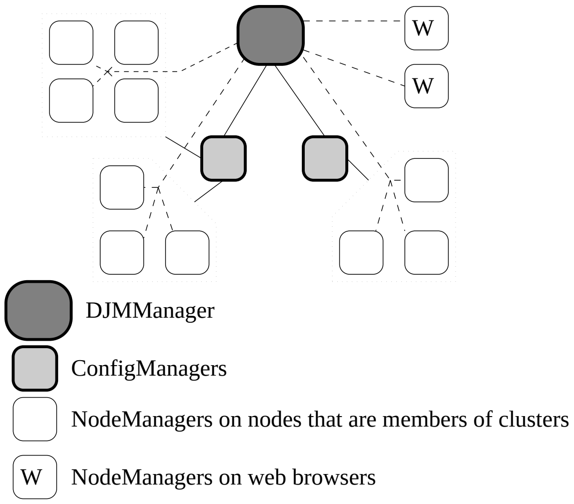

Figure 3:

2.1.1 DJM Components

- DJMManager - DJMManager is a centralized daemon that controls the entire DJM. Among its main responsibilities are overall system configuration management, and application setup. Within DJMManager is DJMFaultManager. Its job is to centrally handle faults detected by the system, communicating with NodeManagers and ConfigManagers as necessary.

- NodeManager - Every node in the system has a NodeManager running on it. DJM applications communicate with NodeManagers, and the NodeManagers communicate directly with the DJMManager as needed. (Note that applications ordinarily only need to communicate with NodeManagers at application startup. Objects in active applications communicate directly with each other.)

  NodeManagers also monitor the state of the node. Periodically informing the DJMManager of the node's new state. (If the DJMManager does not receive an update from a node for a certain amount of time, it will take the node out of the active DJM system, and ask DJMFaultManager to try to determine problem and fix it.)

- ConfigManager - ConfigManagers managers serve two main purposes: 1 - inform the DJMManager about nodes which may be included in the system and the attributes of those nodes. 2 - to manage system specific features of nodes such as launching nodes, etc.

  In this way ConfigManagers provide modularity. They free the DJMManager from needing to know the configuration of the system when it is started. They also handle all architecture specific issues, freeing DJMManager from any system specific functions. In this way arbitrary clusters of nodes can be added to and removed from the system dynamically without requiring them to all use browsers.

- DJMRemoteObjects - Applications that wish to run on DJM will typically use DJMRemoteObjects, but usually indirectly. DJMRemoteObjects are RMI server objects (objects with methods that can be invoked remotely) with added capabilities allowing them to easily be instantiated on arbitrary remote nodes in the DJM system.

  DJMRemoteObjects are typically not directly visible to the programmer. Rather, programmers deal with Distributed Object Groups which in turn use DJMRemoteObjects.

- Consoles - Users use consoles to interact with the system. System administrators work through a SysCon while ordinary users use a NodeCon.

- Code Server - A code server is an http server that is capable of serving requests for system and/or application .class files. DOGMA allows multiple code servers to be active at the same time with each code server serving up specific packages.

2.2 Distributed Object Groups

Distributed object groups allow programs to work with large groups of objects easily and efficiently.

A DOG is organized according to a data configuration scheme. The scheme defines how data will be partitioned, group elements will communicate, what type of coordinates are used to define group element position, etc.

DOG features include:

- Asynchronous group method invocation - Method invocations on the entire group only block when a return value is requested and unavailable.

- Dynamic data partitioning - Arguments to methods can be partitioned automatically by the system according to a group's data configuration scheme, and taking into account the speed and load of nodes on which group elements reside.

- Data re-assembly - Data returned by a group method invocation can also be reassembled automatically by the system. Alternatively an array of results can be returned.

- Group element peer communication - standard functions include point to point and multi-point send and receive methods with the message sends occurring asynchronously. Also, data configuration schemes can define additional methods of communication as needed.

2.2.1 Data Configuration Scheme Components

For a given data configuration scheme, each of the following abstract classes are extended: DataConfigController, DataPartition, ElementInfo, and ElementPosition.

- DataConfigController - Subclasses of this class are responsible implementing basic DOG functions for a given scheme such as data partitioning/re-assembly and group element peer communication.
- DataPartition - When data is partitioned and sent to an object group, elements receive instances of the scheme's extensions of this class. DataPartition subclasses contain the actual data that has been partitioned as well as information about the partition such as the partition start coordinates, partition end coordinates, partition size, etc.
- ElementInfo - Object group elements are sent the scheme's extensions of ElementInfo to inform them of their position in the group, and references to the other elements of the group.
- ElementPosition - ElementPosition is part of the ElementInfo class that each element receives. Each scheme extends ElementPosition to provide position information for group elements. Positions can be anything from a Cartesian coordinate to a non-dimensional position such as HIGH or LOW.
- ElementImpl - Each element in an object group is derived from the data configuration scheme's extension of this class. This gives group elements common capabilities such as peer communication.

3 Sample Application - Matrix Multiplication

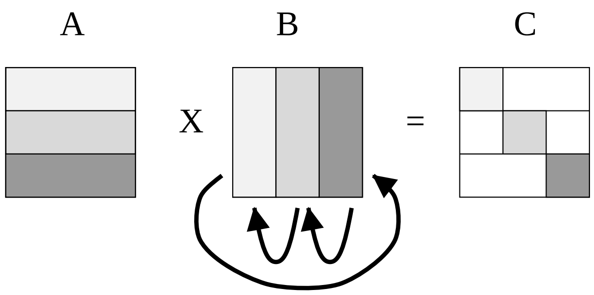

Figure 4:

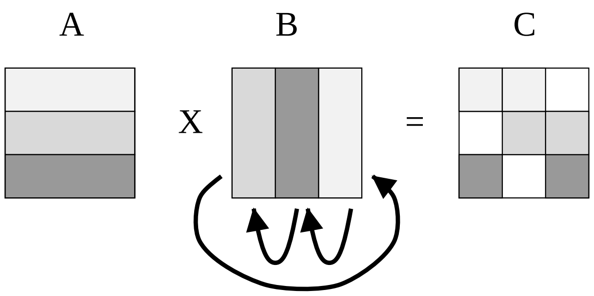

Figure 5:

As an example consider a simple matrix multiplication application. A typical way of coding this is to use a master/slave configuration that works as follows:

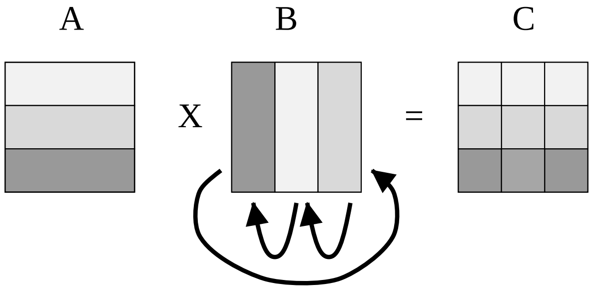

Figure 6:

The master sends **A** and **B** out to the slaves. **A** is partitioned by rows. **B** is partitioned by columns.
Each slave receives its partition **A** as well as its partition of **B** (Figure 4).
Each slave begins sending its partition of **B** to the next neighboring slave (i.e. 0-1 1-2 2-0).
As the communication is proceeding, each slave computes a portion of **C** by multiplying its partition of **A** and its partition of **B** together.
Each slave then receives a new **B** partition from its previous neighbor discarding its old **B** partition.
If n is the number of slaves this is repeated n times so that **C** is entirely computed (Figure 5 and Figure 6).
Each slave then returns its computed partition of **C**.
At the master **C** is automatically reassembled and the algorithm is complete.

4 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:

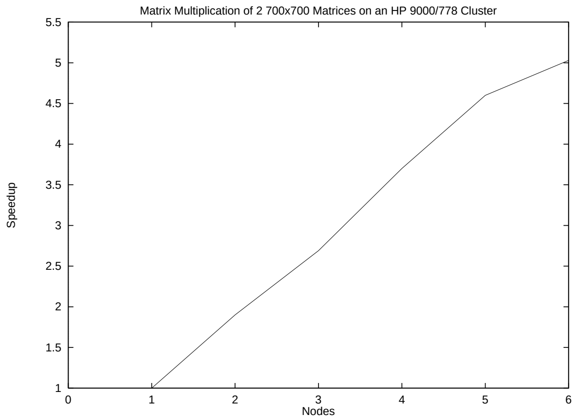

Figure 7:

Figure 7 shows speedup obtained for multiplying two 700x700 matrices on a cluster of HP 9000/778 workstations connected on an 10Mb Ethernet while Figure 8 shows speedup obtained multiplying various sized matrices on a cluster of 133 Mhz Pentiums connected on an 10Mb Ethernet. Even for applications with relatively small execution times (less than 300s for single node execution in all cases) the speedup obtained is fairly linear.

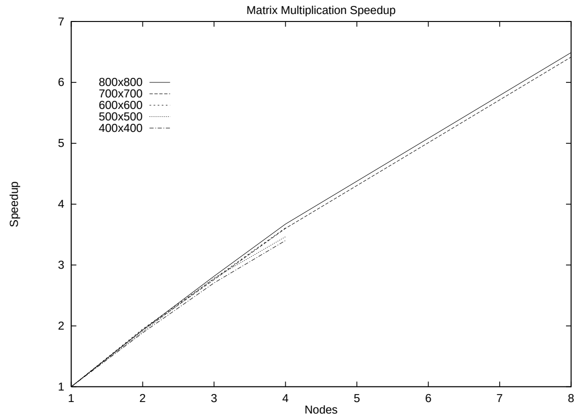

Figure 8:

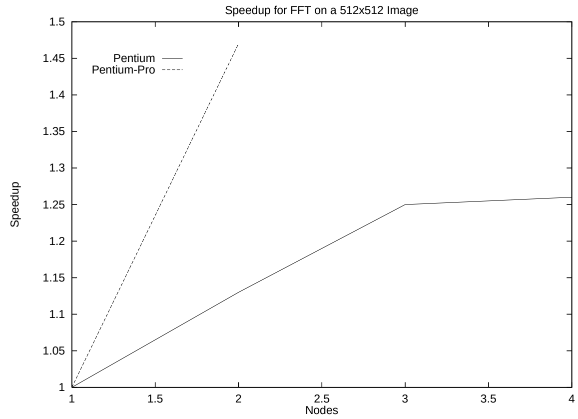

Figure 9:

Figure 9 shows speedup obtained on a cluster of 133 Mhz Pentiums with 10Mb Ethernet vs. a cluster of 200 Mhz Pentium Pros with 100Mb Ethernet executing an FFT algorithm for a 512x512 image. This program executes so quickly (2.1 seconds on a 200Mhz Pentium Pro) that the Pentium cluster was unable to produce very effective speedup. It appears that the Pentium Pro cluster may be able to obtain effective speedup, but conclusion this awaits validation on more nodes.

Figure 10 compares execution time for executing matrix multiplication of various sized matrices for DOGMA on 1 node, Java code w/ JIT, compiled C code, and DOGMA on 2 nodes. Note that DOGMA the difference between DOGMA and ordinary Java code seems to come from some type-casting that is normally done in DOGMA code. DOGMA code can be written without this type-casting, but DOGMA applications are much more elegant with it.

4.1 Power/Load sensitive dynamic data partitioning

As an example of the benefits of dynamic data partitioning consider speedups obtained executing an a matrix multiplication of 2 400x400 matrices:

4.1.1 Effect of different speed machines

In a configuration of 2 Pentium Pros 200Mhz, 1 Pentium 133 Mhz, 1 Pentium 133Mhz Dynamic partitioning resulted in a 33 percent speedup

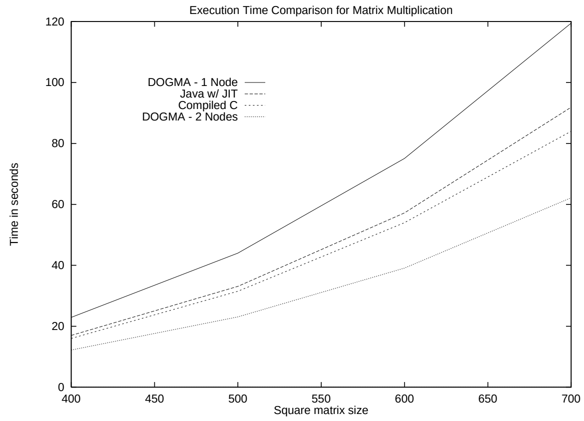

Figure 10:

4.1.2 Effect of different loads

On a cluster of HP 9000/778's with one node executing a 'make', one node lightly loaded, and 2 not loaded dynamic partitioning resulted in a 48 percent speedup.

5 Conclusions

DOGMA is a new, but promising system for practical parallel programming with Java. DOGMA provides both an effective object-group-oriented programming capability, and also provides powerful system configuration abilities such as dynamic code loading, cluster management, and web browser participation.

Preliminary results obtained with DOGMA show that while Java is still an immature language, it is nonetheless a promising for parallel processing.

6 Future Work

Parallel programming with Java is still in its infancy. Tools such as DOGMA open many avenues for research, and below we list a few we wish to focus on in the near term. First, however, it is important to know about current work by others that will have a direct impact on DOGMA in the future.

Work by others with direct impact on DOGMA:

The performance of Java as the basis of a parallel computing platform relies on many Java enhancements currently in development (mostly by Sun):

Java currently has no standard way of loading a new version of a class into a JVM. It is possible to work around this by using a custom class loader, but this is not a clean approach. The effect of this on DOGMA is that the system must be brought down in order to replace classes that have been loaded with new versions. Sun is addressing this issue, and DOGMA will be able to replace classes at runtime as soon as a new JDK allows this cleanly.

The current HP and NT JDK's do not use native threads. As DOGMA is heavily multi-threaded, performance should increase a noticeable amount when native threads are widely available.

As Java matures, compilers and just-in-time compilers should be able to generate code with speeds very close to that of native C even for somewhat large and complex applications like DOGMA. Also, RMI and object serialization are likely to improve in future JDK releases.

Future DOGMA enhancements:

While interpreted "natural" DOGMA code executes only slightly slower than equivalent plain Java code. When running on the Windows/NT system with a JIT current DOGMA code executes around 30 percent slower than does plain Java code. DOGMA code can be coded "unnaturally" to narrow this gap, but this is not desirable. Future research will attempt to resolve this difference.

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 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. Future research will investigate more sophisticated object placement algorithms.

Designing an efficient dynamic data partitioning algorithm is closely related to designing an efficient object placement algorithm. As mentioned earlier, the current dynamic data partitioning algorithm has realized significant improvements in performance from simply taking node load and node speed into account. In the future more sophisticated algorithms may be able to yield yet more performance especially on long running applications.

Future versions of DOGMA will feature object migration. Object migration is very useful for several reasons. Among these reasons are:

Certain objects perform much better on some nodes than on others. For instance, objects which communicate with each other frequently should be placed on nodes with a good network connection (high bandwidth or high latency depending on the needs of the application) between them. Ideally, the object placement algorithm would take care of this, but situations could arise where this would need to be done after object placement.

With a dynamically configurable system, it becomes very important that before nodes leave the system they migrate all active objects to nodes remaining in the system. This is especially true in the case where objects are running on web browsers.

DOGMA's current implementation is targeted for use of a maximum of several hundred nodes spread across an intranet or the Internet. This is due to the fact that DOGMA is centralized at the DJM layer since only one DJMManager may exist in a given system. Future versions of DOGMA will allow multiple DJMManagers in a manner somewhat similar to SuperWeb's brokers (while still retaining DOGMA's cluster management capability/philosophy).

Current implementations of DOGMA rely on native threading in the JVM to make use of multiprocessor machines. Future implementations of DOGMA will make use multiprocessor machines in a manner consistent with DOG architecture by placing multiple objects on them and optimizing the communication between them.

Current fault tolerance is limited to the system level. Faulty nodes are detected and removed from the system. There is much research to be done in both system and application level fault tolerance.

Internet security is nearly a contradiction in terms. Fortunately, Java makes many security issues approachable that would not be if security were not designed into the language. Current DOGMA security is minimal: a relaxing of the RMISecurityManager enough to allow the system to function. The fact that Sun has no standard way for classes to be reloaded means that system uptime is somewhat limited, and security requirements have not yet been very great. Future versions of DOGMA will address security much more thoroughly.

Again, there are many areas for new research in the area of parallel computing on intranets and the Internet that tools such as DOGMA make possible. Future papers will focus on the areas outlined above.

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] L. Beca, G. Cheng, G. C. Fox, T. Jurga, K. Olszewski, M. Podgorny, P. Sokolowski, T. Stachowiak, and K. Walczak. Tango - a collaborative environment for the world-wide web. Technical report, http://trurl.npac.syr.edu/tango/papers/tangowp.html, 1997.

[4] 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.

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

[7] 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.

[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] S. Hirano. The magic ccarpet for network computing. Technical report, http://ring.etl.go.jp/openlab/horb/, 1997.

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

[15] Javasoft. Remote method invocation. Technical report, http://java.sun.com/products/jdk/1.1/docs/guide/rmi/index.html, 1997.

[16] 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.

[17] 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.

[18] Objectspace. Voyager. Technical report, http://www.objectspace.com/voyager/, 1997.

[19] 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.

[20] 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.

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

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

[23] 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.