

JavaNow: A Framework for Parallel Computing on Networks of Workstations

*Shahzad A. Bhatti
Illinois Institute of Technology
Chicago, Illinois*

*Phillip M. Dickens
Illinois Institute of Technology
Chicago, Illinois*

*George K. Thiruvathukal
Loyola University
JHPC Research Group
Chicago, Illinois 60626
http://www.jhpc.org/george_thiruvathukal@acm.org*

Keywords: parallel and distributed computing, Java, Linda, and network of workstations.

Abstract

Networks of workstations are becoming a dominant force in the distributed computing arena, this due primarily to the excellent price to performance ratio of such systems when compared to traditional massively parallel architectures. It is therefore critical to develop programming languages and environments that can help harness the raw computational power available on these systems. The JavaNow (Network of Workstations) system, a Java based framework for parallel programming on a network of workstations, is one such project. It creates a virtual parallel machine similar to the PVM (Parallel Virtual Machine) model, and provides distributed associative shared memory similar to Linda memory model but with a richer set of primitive operations.

JavaNow provides a simple yet powerful framework for performing computation on networks of workstations. In addition to the Linda memory model, it provides for shared objects, publisher/subscriber based event notification, implicit multithreading, implicit synchronization, object dataflow, and collective communications similar to those defined in the Message Passing Interface (MPI).

1 Introduction

Java has been established as one of the preferred languages for writing distributed applications due to its many features which support programming on distributed platforms. In particular, Java provides automatic garbage collection alleviating programmers from memory management. Java provides compile time and runtime security that can be used as the basis for writing secure applications. Java has inherent support for multithreading, and Java 1.2 supports kernel-level threads allowing the overlapping of computation with communication or I/O. Java can save the state of an object and recreate that object on another machine, supporting both persistent objects and object migration. Probably the biggest benefit of Java is its portability; a Java application can be run on any machine with Java support without recompilation.

While using a network of workstations as a virtual parallel computer is not a new idea, there are many features of JavaNow which make it

2 The JavaNow Framework

In this section, we briefly explain the key tenets of the JavaNow framework.

2.1 Model of Computation.

JavaNow provides a framework for building a virtual parallel machine by connecting workstations over a network. Similar to the other tuple space systems based on Linda, JavaNow spawns one daemon process per virtual machine processor. These daemon processes initialize the application processes and begin their execution. Another similarity to other such systems is the assumption of an SPMD model of computation.

2.2. Distributed Associative Shared Memory

The application processes coordinate and communicate through distributed associative shared memory similar to the Linda memory model [CAR92]. In JavaNow, the shared objects are referred to as *Entities* and the repository where these objects are stored is termed the *EntitySpace*. Each Entity in the JavaNow system consists of two components: the **name** of the Entity and its **value**. It is important to note that **name** and **value** both denote Java objects, which unlike Linda objects, carry both state and behavior information.

Unlike the Linda memory model where there is a single repository, JavaNow permits multiple EntitySpace objects. Similar to the Linda memory model, JavaNow allows *active Entity* objects that perform a computation and then convert into passive Entity objects.

JavaNOW uses a simple load balancing scheme to distribute the Entities belonging to a given EntitySpace. In particular, JavaNow uses a hashing function from the name of an Entity to a machine ID. In the case where multiple Entities are created with the same name, the

JavaNow supports both blocking and non-blocking retrieval from an EntitySpace. This is in contrast to Linda where the retrieval operation is blocking (that is, the process performing the retrieval must block until the tuple becomes available). JavaNow (and Linda) support retrieval of an Entity/tuple based on partial information (i.e. when the full name is not known).

A key difference between JavaNow and Linda (and many of its derivatives), is the implementation of the distributed associative shared memory. Most Linda implementations have performed poorly because of the distribution scheme used for tuple space representation (a scheme based primarily on replication), and the fact that Linda's goal of pure location transparency prevents several optimizations. JavaNow also supports location transparency, but (we believe) does so with an implementation that is more efficient and allows for more optimizations than most Linda implementations. This issue will be discussed in detail in the full paper.

2.3 Event Notification

JavaNow provides publisher/subscriber based event notification for any changes to the state of an EntitySpace. Thus a user can subscribe to receive notification whenever a new Entity is added to the EntitySpace, or when an Entity is removed. Further, the user can specify an interface that acts as a filter on the types of events for which a process is informed.

Event notification has been perfected with the introduction of the Java programming language. The most prominent example of event notification in Java occurs, interestingly, in the Abstract Windowing Toolkit (AWT) package, where a design pattern called "listeners" is supported for handling events. A prospective listener implements a particular interface which

we have generalized it to work with workstation clusters using JavaNow.

2.6. Implicit Multithreading

JavaNow creates a new thread to execute each of the ActiveEntity objects. In JavaNow, these threads are created and maintained by the system without involvement from the user. Thus the application can get the performance benefits of thread-based computing without the difficulty of managing the threads directly.

2.8 Implicit Synchronization

All accesses to Entity instances within the EntitySpace are guaranteed to be mutually exclusive, thus freeing the programmer from concerns related to the synchronization of the EntitySpace. The actual workings of the synchronization and blocking semantics are no more complicated than the workings of classic synchronization problems such as the bounded buffer or producer/consumer problems.

The (user-free) synchronization implementation of an EntitySpace is based on an abstraction called a shared directory of unordered queues, a concept which has long been described in operating systems textbooks. This principle is explained in detail in a paper describing Memo [CON94], a predecessor to the JavaNow system. We will provide details in the full paper.

2.9 Collective Communication/Computation

JavaNow supports collective communication operations on Entities in a manner similar to those defined in MPI. All familiar MPI collective communication operations are supported, including broadcast, scatter/gather, and reduce. These operations all work with primitive types, arrays of primitive types, or objects.

performance computing) such as suspended evaluation, lazy evaluation, task graphs, and many commonly used techniques used for high-performance parallel and distributed computing.

2.10. Choice of Communication Providers

JavaNow is designed and implemented as a set of Java interfaces, and it separates all networking code from the rest of the implementation code. This allows the use of different providers for communication. Versions of JavaNow have been built using Sockets, Remote Method Invocation, and (soon) CORBA

3. FUTURE PLANS

Several applications are under development to test JavaNow on a variety of platforms. We have identified the following list of enhancements that will increase JavaNow's performance and usability.

- Performance: The current release of JavaNow is implemented primarily as a proof of concept, and we have yet to pursue any significant performance tuning. Performance issues will be of paramount importance in future releases.
- Dynamic resource management: The current release of JavaNow requires that the user statically specify the list of machines on which the application will run. In the next release, JavaNow will allow users to add/delete machines dynamically.
- Dynamic load balancing: The current release of JavaNow uses a simple hashing scheme for load balancing. In the future, JavaNow will provide dynamic load balancing to more effectively share computation and memory resources.

4. Related Work

Several other projects are using network of workstations for building parallel applications. The most significant work includes Linda [CAR92], PVM [SUN92] and MPI [LUS94].

As noted above, Linda provides a concurrent programming model and an associative shared memory based on the concept of a “Tuple Space.” A Tuple is a sequence of fields, each of which has a type and contains a value or a variable. There are two kinds of Tuples, passive Tuples and Active Tuples. Passive Tuples store data, and active Tuples perform a computation. Once an Active Tuple has finished its execution, it turns into a passive Tuple. JavaNow also uses associative shared memory similar to the Linda model. However, and as noted above, JavaNow distributes the Entities across *all* of the processors in the virtual machine, provides for non-blocking retrieval and allows multiple EntitySpaces.

Other parallel frameworks based on Linda include C-Linda [NAR90], Glenda [SEY], Memo [CON94B], and JavaSpaces [SUN98]. C-Linda is C based implementation of Linda, Glenda is Linda implementation on top of PVM. Memo is a C library that implements Linda like data structures for storing associative shared memory. JavaNow is distinguished from all of these systems in that it is a Java based, rather than a C based system. As noted above, there are some significant advantages to using Java for the development of distributed applications.

In contrast to the C-based Linda-like systems, JavaSpaces [SUN98] is a Linda-based framework written in Java. JavaSpaces uses a server object to manage a space and users can create multiple spaces. However, JavaSpaces does not offer facilities for the creation of active tuples and does not provide an infrastructure for parallel programming. JavaSpaces is specifically

message passing based system and the other is a shared memory based system. It has been argued that writing applications for a shared memory machine is easier than writing applications for shared memory systems. It has also been argued that shared memory code is smaller than equivalent message passing code.

Systems that use Java to enhance PVM include JPVM [FER98], jPVM¹ [THU96]. Like PVM, these systems use message passing for processor communication and coordination. This is in contrast to JavaNow which provides distributed associative shared memory.

MPI [LUS94] is a proposed standard library specification for message passing systems. MPI is available on most large-scale computers and several implementations such as LAM [LAM] are available for network of workstations. Java-MPI [CHA97] is Java-based wrapper library for MPI and uses standard MPI underneath. Java-MPI thus provides a message passing rather than shared memory communication model.

Other Java based projects that use the Internet and Web for parallel computing include JavaParty [PHI97], Javlin [CHR97], ParaWeb [BRE96], JavaDC [CHE97], Bayanihan [KAL96, KAL97], KnittingFactory [BAR98], IceT [GRA97], and WebFlow [FOX97].

The major difference between these systems and JavaNow is that most of the aforementioned systems are targeted for Web based parallel computing and are applet based. JavaNow is targeted for workstation clusters, and is run as a stand-alone application. Also, most of these systems are essentially intended as a framework for distributing tasks, and use paradigms such as PVM, MPI, Linda and Charlotte as a *separate* layer on top of those frameworks. JavaNow offers a framework for both distributing *and* coordinating parallel tasks. It also uses associative shared memory for task

parallel, or data-flow style. Most competing frameworks do not offer this kind of flexibility.

5. CONCLUSION

This paper describes the JavaNow framework for developing parallel applications in Java for execution on a network of workstations. JavaNow offers a conceptually simple and easy to use model for process communication and coordination. Additionally, the architectural neutrality of Java allows JavaNow to execute on any machine with Java support without the need for recompilation.

There are of course concerns related to the performance of Java given that it is an interpreted language. However, with the just-in-time compilers and Java hot spot technology, the speed of Java applications has been improving. We will provide performance results for JavaNow in the full paper.

REFERENCES

- [AND95] T. E. Anderson, D. E. Culler, and D. A. Patterson. *A case for NOW*. *IEEE Micro*, February 1995.
- [BAR98] Arash Baratloo, Mehmet Karaul, Holger Karl, and Zvi Kedem. *An Infrastructure for Network Computing with Java Applets*. In Proceedings of ACM Workshop on Java for Science and Engineering Computation, February 1998.
- [BLU95] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, H. Randall, and Y. Zhou. *Cilk: An Efficient Multithreaded Runtime System*, in Proceedings of the 5th ACM SIGPLAN Symposium on Principles of Parallel Programming, 1995.
- [CAR92] N. Carriero and D. Gelernter. *How to write parallel programs*: 46, 1992.
- [CAR98] Bryan Carpenter, Guansong Zhang, Geoffrey Fox, Xinying Li, and Yuhong Wen. *HPJava: Data Parallel Extensions to Java*. In Proceedings of ACM Workshop on Java for Science and Engineering Computation, February 1998.
- [CHA97] Chang, Y-J., and Carpentar. MPI Java Wrapper Download Page, March 27 1997. <http://www.npac.syr.edu/users/yichang/javaMPI>.
- [CHE97] Zhikahi Chen, et al. *Web Based Framework for Distributed Computing*. In Proceedings of ACM Workshop on Java for Science and Engineering Computation, Las Vegas, NV, June 1997.
- [CHR97] Bernd O. Christiansen, et al. *Javlin: Internet-Based Parallel Computing Using Java*. In Proceedings of ACM Workshop on Java for Science and Engineering Computation, Las Vegas, NV, June 1997.
- [CON94] W. T. O'Connell, G. K. Thiruvathukal, and T. W. Christopher. *Distributed Memo: A Heterogeneously Parallel and Distributed Programming Environment*. In Proceedings of the 23rd International Conference on Parallel Processing, August 1994.
- [CON94B] O'Connell, Thiruvathukal and Christopher. *The Memo Programming Language. Proceedings of the International Conference on Parallel and Distributed Computing Systems*, October 1994.
- [CRA93] Phyllis E. Crandall, Michael J. Quinn. *Data Partitioning for Networked Parallel Processing*. IEEE, 1993. pp. 376-379.
- [DON93] Jack Dongarra, Al Geist, Robert Manchek, and Vaidy Sunderam. *Integrated*

Laboratory: A Web-Based Parallel Programming Environment
Concurrency: Practice and Experience 9:485-508, 1997.

[GEI92] G. A. Geist and V. S. Sunderam. *Network Based Concurrent Computing on the PVM System*. *Journal of Concurrency: Practice and Experience*, 4, 4, pp 293--311, June, 1992.

[GEI93] G.A. Geist and V.S. Sunderam, *The Evolution of the PVM Concurrent Computing System*, Proceedings - 26th IEEE Compcon Symposium, pp. 471-478, San Fransisco, February 1993.

[GEL85] Gelernter, David, *Generative Communication in Linda*, ACM TOPLAS, 7:1, Jan. 1985.

[GRA97] Paul A. Gray and Vaidy S. Sunderam. *IceT: Distributed Computing and Java*. In Proceedings of ACM Workshop on Java for Science and Engineering Computation, June 1997.

[KAL96] L. V. Kale and Joshua M. Yelon. *Threads for Interoperable Parallel Programming*. Proceedings of the conference on Languages and Compilers for Parallel Computing, 1996.

[KAL97] L. V. Kale, Milind Bhandarkar, and Terry Wilmarth. *Design and Implementation of Parallel Java with Global Object Space*. Proceedings of Conf. on Parallel and Distributed Processing Technology and Applications, Las Vegas, Nevada, 1997.

[KAL98] L. V. Kale, Milind Bhandarkar, Robert Brunner and Joshua Yelon. Multiparadigm, *Multilingual Interoperability: Experience with Converse*. Second Workshop on Runtime Systems for Parallel Programming (RTSPP), March 1998.

[LUS94] Gropp, Lusk and Skjellum. *Using MPI: Portable Parallel Programming with the Entity-Passing Interface*, 1994.

[NAR90] James Nareem. *An Informal Operational Semantics of C-Linda V2.3.5*. Technical Report 839, Yale University Department of Computer Science, Dec. 1990.

[OTT95] S. W. Otto, M. Snir, and D. Walker. *An Introduction to the MPI Standard*, J. Dongarra, CS-95-274, January 1995.

[PHI97] Michael Philippsen and Mathias Zenger. *JavaParty: Transparent Remote Objects in Java*. In Proceedings of the ACM PpoPP Workshop on on Java for Science and Engineering Computation, Las Vegas, NV, June 1997.

[SAR98] Luis Sarmenta, Satoshi Hirano, and Stephen Ward. *Towards Bayanihan: Building an Extensible Framework for Volunteer Computing Using Java*. In Proceedings of the 2nd Intl. Conference on Worldwide Computing and its Applications, Tsukuba, Japan, March 1998. <http://www.cag.lcs.mit.edu/bayanihan>.

[SEM98] Yelick, Semenzato, Pike, Miyamoto, Liblit, Krishnamurthy, Hilfinger, Graham, Gay, Colella, Aiken. *Titanium: A High-Performance Java Dialect*. ACM 1998 Workshop on Java for High-Performance Network Computing, Stanford, California, February 1998.

[SEY] Ray Seyfarth, Jerry Bickham and Suma Arumugham. *Glenda*.
<http://sushi.st.usm.edu/~seyfarth/research/glenda.html>

[SIN93] Amitabh B. Sinha and Laxmikant V. Kale. *A Load Balancing Strategy For Prioritized Execution of Tasks*. International Symposium on Parallel Processing, Newport Beach, CA, April 1993.

Concurrency: Practice and Experience, 2(4), pp. 315-339, December 1990.

[SUN92] G.A. Geist and V.S. Sunderam, *Network Based Concurrent Computing on the PVM System*, Journal of Concurrency: Practice and Experience, (4), pp. 293-311, June 1992.

[SUN94] V. Sunderam, J. Dongarra, A. Geist, and R. Manchek. *The PVM Concurrent Computing System: Evolution, Experiences, and Trends*. Parallel Computing, Vol. 20, No. 4, April 1994, pp 531-547.

[SUN98] Sun Microsystems, Inc. *JavaSpaces Specification*, July 1998.
<http://java.sun.com/products/javaspaces/>

[THU96] David A. Thurman. *JavaPVM: The Java to PVM Interface*, December 1996.
<http://www.isye.gatech.edu/chmsr/jPVM>.

6. [YU97] Weimin Yu and Alan Cox. *Java/DSM: A Platform for Heterogeneous Computing*. In Proceedings of ACM Workshop on Java for Science and Engineering Computation, Las Vegas, NV, June 1997.