Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, 26-29 August 2004

DEVELOPMENT OF A JAVA-BASED DISTRIBUTED PLATFORM FOR THE IMPLEMENTATION OF COMPUTATION INTELLIGENCE TECHNIQUES

CHUN-CHE FUNG[1], JIA-BIN LI[1], KIT PO WONG[2]

[1] School of Information Technology, Murdoch University, Murdoch, W.A., AUSTRALIA
[2] Department of Electrical Engineering, Hong Kong Polytechnic University, HONG KONG SAR
E-MAIL: L.Fung@murdoch.edu.au, J.Li@murdoch.edu.au, eekpwong@polyu.edu.hk

Abstract:

This paper reports three phases of development of a Java-based distributed system for the implementation of computational intelligence techniques to solve practical engineering problems. The first design of the Java-based system is called Para Worker. It is a library of Application Program Interface (APIs) to facilitate parallel computation. Next, a new system, termed Para Worker 2 aimed at improving ease of programming and performance. It also adds enhanced features of improved dynamic object reallocation, adaptive consistency protocols, and location transparency. However, neither of these systems addressed the fundamental issues of Java Virtual Machine (JVM) in distributed environment. The third phase of development of the Java-based system is a new form of distributed JVM in order to execute Java program efficiently. The new system is termed Java Distributed Platform (JavaDP).

Keywords:

Java-based parallel computer; Java Virtual machine; Message-passing systems; Distributed Shared Memory; Cache coherency; Actor model

1. Introduction

Computational Intelligent (CI) techniques primarily concern with three major disciplines: Artificial Neural Network (ANN), Fuzzy Logic (FL) and Evolutionary Computation (EC). Such techniques are frequently used to solve many real-life and complex problems in system engineering. Particularly, practical problems with non-linear and nondeterministic nature are often not applicable via traditional solution processes. Examples are many problems in electrical power system engineering which generally require massive computational efforts.

Although an advanced sequential computer is able to provide enough computational power to solve those complicated problems, a parallel computer structure is more viable in practice due to two reasons. First, most of the CI techniques, such as genetic algorithms, are inherently parallel. Their structures are thereby well-suited to some of the parallel computer structures such as cluster systems. Further, it is more cost-effective to build a parallel computer than to improve performance of a single sequential computer. Physical limitations and increasingly sophisticated chips design are two major challenges in the development of high performance processors [1]. Chip developers across the industry now agree that clock speed is no longer the key measure of processor performance. Rather, the aim of improving the overall throughput of a system has become the first priority of today's chip developers. This requires new approach to improve the system performance without putting too much emphasis on the speed of the processors.

1.1. Distributed Computing Structure

In computing, one of the ways to improve throughput of a system is by deploying parallelism. To illustrate this principle, Instruction-level parallelism (ILP) is the key technique which has been applied to modern processors in order to improve throughput. Another major technique is to integrate multiple processors or computers to work cooperatively.

Following the success of many cluster systems, distributed computing structure which bases on message-passing mechanism has become a dominant approach to construct a large-size parallel system. In particular, distributed system with COTS-based (Commodity-off-the-shelf) technologies plays a major role in redefining the concept of supercomputing. Features of the COTS-based systems are cost effective, system availability; fault tolerance and reliability, and scalability.

1.2. Trends in Software Development

While most people recognize the significant advances in hardware that have taken place, there has also been a

0-7803-8403-2/04/$20.00 ©2004 IEEE

dramatic change in the software. Software packages have become very much larger, but they have also become far more sophisticated and offer an increasingly wider range of services. In many cases, the ability to meet these increasingly complex demands on software has only been achieved through the powerful paradigm of the Object Oriented Programming (OOP) model.

Java is one of the most popular object-oriented languages created by the Sun Corporation. It was originally developed as means for developing highly sophisticated intelligent appliances. Such examples are next generation Televisions, PCs and other systems with high degree of Connectivity, Interactivity and Functionality. However, shortly after its development, the internet arose and for a time Java become better known as "a language for networked applications".

The importance of threads should also be emphasised which is used to facilitate parallel algorithms. Java provides a standard Thread class for the purpose of implementation of such algorithms. In addition, Java also provides several synchronisation mechanisms to access share variables or objects. For this reason and following the previous work in Para Worker [2], Java has been adopted as the development language for the present study.

In the subsequent sections, this paper gives general descriptions, as well as the motivation, for each generation of Java-based distributed platform. This paper is organized as follows: Section 2 gives an overview of the development of the parallel platform and Section 3 discusses the design of the first version Java-based parallel platform – Para Worker. Section 4 discusses the development of Para Worker 2 followed by the qualitative comparison between the two versions of Para Worker is presented in Section 5. Section 6 illustrates performance of the versions of the Para Worker for executing the Economic Dispatch (ED) problem in electrical power systems engineering. Section 7 proposes a new form of a Java virtual machine – JavaDP – aimed at executing Java programs more efficiently. This is followed by a conclusion for this paper.

2. Overview

There are three major design approaches to implement parallel computers: Multiprocessors, Multi-computers, and Distributed Shared Memory (DSM). In the case of a multi-processor design, all the processors are confined and share the resources within a single, global address space. Read and write operations are both allowed for any processor to access the common address space. Communication among processors is thereby made through the shared memory. This architectural design is also termed as a tightly coupled parallel system. On the other hand, a multi-computers system comprises of a network of computers. Every computer in the network has its own private memory space and I/O modules. Hence communication is via message-passing. In practice, multiprocessors systems are costly and complicated to build but they are relatively easy to program. On the other hand, multi-computers are just the opposite. Distributed Shared Memory architecture can be considered as an intermediate solution which takes advantages of the two design philosophies. In DSM, each computer has both local and shared memory space. The shared memory space is visible to all remote processors located in the same network. From a programmer's perspective, the read or write operations on the shared or local variables are being treated the same irrespective to their physical locations. In other words, variables stored in the shared memory space can be accessed by a remote processor in a same manner as variables being stored in the processor's local memory space.

2.1. Consistency Protocols

Similar to cache coherence problem in multiprocessor design, the shared memory locations of a DSM system must be protected and synchronized to ensure data consistency. There have been a number of consistent protocols for the DSM design proposed in the past years. Examples of such developments are Sequential Consistency, Lazy Release Consistency and Entry Consistency.

Sequential Consistency follows closely to a single processor environment. On the other hand, the lazy release consistency (LRC) model allows postponement of the update until a synchronisation variable is acquired by a remote processor. The false sharing problem in Sequential Consistency is overcome by using different modes of multiple writer protocol which allow concurrently access by different processors to different sets of data within a page. This has resulted in an improvement in the system performance. The entry consistency model associates a synchronisation variable with every shared variable. Similar to LRC, the EC approach updates or invalidates shared variables associated with the acquired synchronisation variable.

Based on these consistency models, two new consistency protocols were subsequently developed: Scope of Consistency (SoC) and OMW [3]. In general, SoC bridges the gap between the inefficiency in LRC and the hard programming requirements in the Entry Consistency model. However, OMW, namely an object-based, multiple-writer consistency protocol, associates a synchronization variable with every shared object. Unlike paged-based

consistency models, OMW allows varied lengths of memory cells. It is also characterised as the update-based, multiple-writer lazy release consistency protocol.

3. Introduction to Para Worker

Para Worker is a command line driven program designed to facilitate implementation of parallel algorithms over the defined structures [2]. Commands, or controls, are issued to the program via the standard input and output streams. There are four distinct mechanisms that facilitate the implementation of parallel algorithms, namely, Master Mechanism, Peer Mechanism, Class Transfer Mechanism and the Message Passing Mechanism. Each mechanism performs a fundamental function that eases development of parallel and distributed algorithms.

3.1. Master Mechanism

The Master Mechanism is intended to be used by those nodes that are acting as a master node to a number of slave nodes. It provides a storage location for information about each of the slave nodes. It can also control whether or not other potential slaves can connect to this master, and allows slaves to be disconnected at will. Facilities are included that allow the master to issue instructions to one or all of the nodes.

3.2. Peer Mechanism

The Peer Mechanism is the logical opposite to the Master Mechanism. Instead of defining relationships between nodes as definite master-slave, the two nodes are of equal importance.

Like the Master Mechanism, the Peer Mechanism allows establishment and destruction of peer interconnection networks between nodes. It also facilitates simple message passing between these nodes.

3.3. Class Transfer Mechanism

The Class Transfer Mechanism provides facilities to transfer executable classes – tasks – between nodes. The premise behind this Mechanism is that a slave node need not already possess a class that a master node intends it to run. If this event occurs, the master node can easily transfer the appropriate class to the slave, and then order the slave to execute the class.

3.4. Message Passing Mechanism

The Message Passing Mechanism facilitates the exchange of messages between nodes. It uses the UDP communications protocol to do so. The messages that are transmitted are not guaranteed to arrive at the destination, nor are they guaranteed to be correct. The advantage of this mechanism is that the message transfer is extremely fast and efficient. It also operates as a connection-less service. There is no need to establish a tangible link between nodes before transmission commences, and messages can be immediately sent.

4. Development of Para Worker 2

The aim of the development on Para Worker 2 is to improve on the programming interface and the performance of its ancestor. The design of the Para Work 2 combines features from DISK and Java/DSM, which they are introduced by Surdeanu [3] and Yu [4] respectively. The proposed system improves the previous system by adding:

- adaptive protocols, and
- a dynamic object migration technique

Taking the advantages of the adaptive protocols [5], Para Work 2 automatically chooses between the different protocols depending on the access patterns of the application and the workload in the network. It is recognized that the performance of a protocol is application dependent. In contrast to the original Para Worker, the proposed system allows dynamically allocate processes across different computers to balance the workload on the system.

In addition, Para Worker 2 adds the Resource Monitoring module to the original system to enhance effectiveness of the automated process migration. The module is to monitor:

- network environments, such as link congestion and processor and recourses failures;
- workload on its processor; and,
- access patterns of share objects by threads.

The function of the object allocation is important in DSM systems, especially in a dynamical varying environment. Furthermore, the Monitor module also informs the Consistency module the access patterns of the share objects accessed by the remote threads. The system in turn considers the adaptation of appropriated protocols accordingly to order to improve the performance of the

application under execution. Similar to the system presented in [2], Para Worker 2 is to implement the following protocols:

- Single- and multiple-writer protocols;
- Invalidate and update protocols.

First, in the single-writer protocol, it allows only one single writable object in the system at any given time; however, there are possible several read-only copies of the object stored in other processors. In contract, the multiple-writer protocol allows multiple writable copies of the share object to coexist within the system [5].

Secondly, the major difference between invalidate and update protocols is the invalidate protocol which allows postponing of the modification of a share object until it is accessed by another thread. More detailed implementation issues of Para Worker 2 are discussed by Fung et al [6] and they are not the intension of this paper.

5. Comparison with Para Worker

The differences between two proposed systems are summarised with reference to the following aspects – Programming Interface and Run-time Environment.

With respect to programming interface, Para Work requires programmers to write separate codes for different machines such as servers and clients and carefully analyse the network topology prior to the execution of the program. The task is generally difficult and error-prone. Taking advantage of object allocation management, Para Worker 2 allows location of objects to be handled automatically. At the start of the application, threads are distributed throughout the network. It is unnecessary to write codes for different machines, even though these machines are heterogeneous.

More importantly, in Para Worker 2, a programmer is free from explicit control definitions, similar to four mechanisms implemented in Para Worker. The control flow of a program is viable by consistency protocols. The system automatically detects the status of any object, such as share and unshared [3]. For share objects, consistency function calls are inserted before object and class accesses instructions.

In terms of Run-time Environment, Para Worker 2 allows objects and threads reallocating dynamically for the application to adapt changing run-time environment. For instance, threads stored in the faulty processor are transferred to other processor without restoring the system. In contrast, Para Worker does not consider dynamic changing of the network. Consequently, any failure occurred in the Para Worker system will require restoration of the application.

6. Solving the ED Problem with an EC Approach

To test the performance of two versions of the Para Worker systems, an Evolutionary Computation (EC) based approach to the problem Economic Dispatch (ED) was adopted as the benchmark tool to evaluate their performance. The principal objective in economic dispatch of thermal generators in a power system is to resolve the economic loadings of the generators so that the load demand can be met and the loadings are within the feasible operating regions of the generators [7].

Simulation results from execution of the EC approach of the ED problem using both Para worker systems are illustrated in Figure 1 and Figure 2. The graphs draw the total computation time on which a system solving the ED problem against varied parameters. It is clear that performance of Para worker 2 is better than that of Para worker. In Para worker, all tasks were evenly distributed to every machine in spite of their different capacities. In such case, some of machines might experience heavier network traffic or higher CPU utilization. Faster machines thereby have to halt to wait for responds from those slower machines. Under such circumstance, the overall computation time is lengthened.

However, Para worker 2 is able to compensate such resource diversities by distributing tasks to appropriate machines according to their status. That is, workload assigned to a machine can be different from another in the system. It effectively minimizes the waiting time wasted in the fast machine. Further, the update consistent protocol added improvement in performance of Para worker 2 by allowing shared objects being cached in the local memory without accessing them remotely. As a result, Para worker 2 is a more advanced parallel platform compared to Para worker as shown.

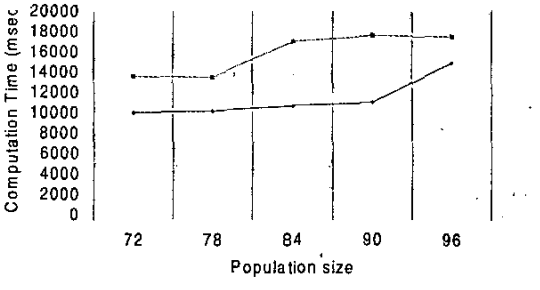

Figure 1: Performance of PW and PW2 against varied population size.

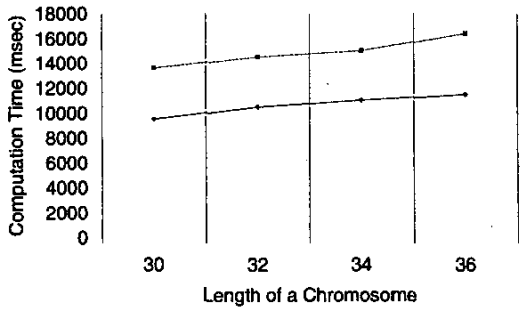

Figure 2: Performance of PW and PW 2 against varied chromosome length.

Table 1: Comparison of performance of PW and PW 2 on various population (Chromosome length = 30)

Population size	Performance (msec)	
	PW 2	PW
72	10025	13610
78	10195	13499
84	10695	17034
90	11016	17605
96	14912	17535

Table 2: Comparison of performance of PW and PW 2 on various length (Population size = 72)

Chromosome length	Performance (msec)	
	PW 2	PW
30	9504	13609
32	10506	14521
34	11026	15021
36	11487	16413

Table 3: Summary of three proposed Java-based distributed systems

Proposed Systems	Programming style	Synchronization	Message type	Message counts	Network topology
Para Worker	explicitly message passing	Explicit	complex structure	small	Static
Para Worker 2	DSM approach	Implicit	complex structure	medium	static and dynamic
JavaDP	concurrent OOP	Implicit	simple structure	large	static and dynamic

7. Third Generation JavaDP

As mentioned previously, the motivation for development of Para Worker 2 is to ease the task of programming on a distributed environment and to improve the performance of the previous Para Worker systems. The key feature of Para Worker 2 is to adopt the DSM approach to stimulate concurrent object-oriented programming. This is contradictory to Para Worker, where it relies on the mechanism of explicit message-passing, Para Worker 2 allows a programmer to write a program as if it is running on a single processor machine. However, the programmer has to differentiate between computation agents – usually threads and data objects during programming. The object/thread module automatically detects the states of data objects [6]. If a data object shared by various computational agents, the module provides necessary protection to those shared objects via proper consistency protocols.

The design philosophy of Para Worker 2 is heavily influenced by the CSP (Communicating Sequential Processes) computational model. The basis of the CSP model is that concurrently executing processes operate on their own private data and only interact with one another using the communication and synchronization mechanisms.

However, both proposed parallel systems fail to address the key weaknesses exist in the Java language and JVMs. First of all, while the feature of inheritance improves modularity and reusability of a program, it however adds difficulties to concurrent Java programming. To illustrate the effects, assuming two objects inherit several data and methods from the same class are allocated to remote processors. According to the Java virtual machine specification, the super class must be instantiated and loaded into the common runtime data area – heap – prior to instantiation of any child object. Consequently, the class file of the super class, as well as its super classes, has to be sent to remote machines through the network. Further, any shared data defined in the super class must be protected by a means of synchronization to enforce data consistency.

Secondly, a stack is used as the only communication channel for objects within the same thread to exchange messages. The use of stack is to guarantee the sequential order of message arrival. That is, the stack ensures sequential execution of Java programs since the arrival of a

message for an object is the basic cause of computation in object-oriented models [8]. Further, a communication channel has to be explicitly defined by the programmer to allow an object to send messages to a remote object. That is, the normal JVM implementation assumes a single processor environment.

Naming object is another issue associated with JVMs. Although Java supports remote object naming, such as RMI servers, all these techniques are not convenient for programming and inefficient for use in a large-size distributed system.

To overcome these difficulties, a new form of Java virtual machine is proposed, namely JavaDP (Java Distributed Platform). The project of JavaDP encompasses the following aspects:

- an efficient message-passing mechanism, especially for communication with remote objects;
- investigating decentralized class structures;
- investigating dynamic naming allocation algorithms, as well as their interaction with other machine functions; and,
- load balancing algorithms.

It is likely that the design of JavaDP will be based on the design of Para Worker 2 as the concepts of threads and DSM are important to distributed computing. As a result, how to combine features of the DSM approach and Actor models [8] – the computation model for parallel object-oriented computers – is the key goal of the JavaDP project. Finally, a qualitative comparison of three proposed Java-based systems is presented in Table 3.

8. Conclusion

This paper reports the development of low-cost Java-based distributed platforms. Two generations of Para Worker have been implemented and benchmarked with an engineering problem. The original Para Worker implemented a library of APIs to facilitate parallel computation. On the other hand, Para Worker 2 aimed at improving ease of programming and performance. It also adds enhanced features of improved dynamic object reallocation, adaptive consistency protocols, and location transparency. However, neither of these systems addressed the fundamental issues of Java Virtual Machine (JVM) used in a distributed environment. The third phase development of Java-based systems is proposed to overcome these difficulties. The new system, termed JavaDP, will have an efficient means to execute Java programs.

Further, evaluation on two versions of Para worker systems was performed by simulating the Economic Dispatch approach on these systems. Simulation results clearly showed that Para worker 2 is superior to Para worker on executing various workloads. This is due to Para worker 2 incorporates with both static and dynamic aspects of a program.

References

[1] Hamilton S, "Taking Moore's Law Into the Next Century", Computer, vol. 32, no. 1, pp. 43-48, 1999.

[2] C. C. Fung, S. Y. Chow, and K. P. Wong, "A Low-Cost Parallel Computing Platform for Power Engineering Applications," 1999.

[3] M. Surdeanu and D. Moldovan, "Design and Performance Analysis of a Distributed Java Virtual Machine," IEEE Tran. on Parallel and Distributed Systems, vol. 13, 2002.

[4] W. Yu and A. L. Cos, "Java/DSM: A Platform for Heterogenous Computing," presented at Proc. ACM 1997 Workshop Java for Science and Eng. Computation, 1997.

[5] C. Amza, A. L. Cox, S. Dwarkadas, L.-J. Jin, K. Rajamani, and W. Zwaenepoel, "Adaptive Protocols for Software Distributed Shared Memory on Networks of Workstations," Computer, vol. 29, 1999.

[6] Fung C.C., Li J.B., Wong K.W. and Wong K.P., "A Java-based Parallel Platform for the Implantation of Evolutionary Computation for Engineering Applications", accepted for presentation at the 2003 Congress of Evolutionary Computation (CEC 2003), 8-12 Dec. 2003, Canberra, Australia, 2003.

[7] Wong K.P, Wong Y.W; "Genetic and genetic/simulated-annealing approaches to economic dispatch"; IEEE Proceedings of Genetic Transmission Distribution Vol.141, No.5, September 1994.

[8] Agha, G. A., "Actors: A Model of Concurrent Computation in Distributed Systems", Cambridge, MIT Press, 1986.