Hybrid Cluster Computing with Mobile Objects

Liang Cheng, Ajay Wanchoo, and Ivan Marsic
Department of Electrical and Computer Engineering, Rutgers University
94 Brett Rd., Piscataway, NJ 08854
{chengl,wanchoo,marsic}@caip.rutgers.edu

Abstract

The hybrid cluster computing is the computing based on the platform of hybrid clusters, which is the computer cluster consisting of both the stationary and mobile computers, interconnected by wireless and wired networks. The study presented in this paper contributes to the research of hybrid cluster computing in two aspects: one is that a prototype for the hybrid cluster computing is implemented based on the proposed framework with Java mobile objects and the mobile IP, the other is that the performance of the hybrid cluster computing is studied, especially the trade-off between the communication load and the computational load. Experimental results are presented.

1. Introduction

In recent years the technology advance with the ever-increasing performance of individual workstations and communication networks gives the momentum to the research on cluster computing. A cluster in the computing context is a collection of networked computer systems that form, in varying degrees, a single unified computing resource. With powerful personal computers (PCs) and high-bandwidth, low-latency network connections, such a cluster can achieve similar or better performance and reliability than traditional mainframes or supercomputers. For example, the Berkeley Network of Workstations (NOW) [1] combines commodity workstations and switch-based network components to a successful large-scale parallel computing system. Moreover, the cluster computing using commodity components is an appealing and cost effective way to serve the computationally intensive applications [2]. For example, the Beowulf project [3] uses multiple Ethernets with TCP/IP to cluster multiple PCs for the distributed and parallel computing.

With the popularity of portable computing devices with wireless links, mobile terminals are becoming commonplace in clusters. In this paper, the hybrid cluster, which is the computer cluster with both the stationary and mobile computers, is considered as the platform for distributed and parallel computing. Normally, the portable terminals have low processing power, small memory, and limited battery lifetime; and wireless links have low bandwidth, large transmission latency, and high packet loss rate. Because of these characteristics, distributed parallel applications that have good performance in wired networking environments may not have the same performance in the hybrid environment.

It is important for mobile users to access heavy computational resources to achieve ubiquitous computing [4]. Imagine, for example, an architect walking around a museum to re-design it. Using a wearable computer, the architect selects different design patterns and the proposed alterations get superimposed onto the 3D images presented on a head-mounted display as the surrounding view for evaluation. The wearable computer delegates the computationally intensive 3D rendering to powerful workstations and collects the results for rendering 3D views.

There are many other scenarios that have common features with the example above: the mobile users take advantages of remote computing facilities to fulfill computationally intensive tasks. Therefore, hybrid cluster computing, i.e., distributed and parallel computing involving the mobile wireless computing environment, has recently gained a lot of research interest. There are papers [5-7] addressing the potential problems of mobile wireless computing, such as timeliness issue of nodes migrating, termination detection in distributed algorithms, etc. The prototype performance of the hybrid cluster computing is an open research area. This paper contributes: (*i*) a framework and a prototype implementation for the hybrid cluster computing with Java mobile objects and mobile IP, and (*ii*) the performance study of the hybrid cluster computing, especially the trade-off of the communication load and the computational load based on the experimental results.

The paper is organized as follows. Section 2 proposes the framework of the hybrid cluster computing and presents the design concerns regarding its flexibility. Then details of the prototype implementation of the hybrid cluster are described in Section 3. Experimental results and corresponding analysis of distributed parallel applications using the prototype of the hybrid cluster are presented in Section 4. Section 5 concludes this paper.

2. Design of Hybrid Cluster Computing

The hybrid cluster computing is the computing based on hybrid clusters. Our design goal is to propose a flexible framework for hybrid cluster computing and address its flexibility concerns.

2.1. Framework Design

The framework is designed as in Figure 1. Commodity computers and networks provide the hardware support for hybrid cluster computing, and parallel and distributed applications take advantages of the single system image by the cluster middleware supporting the mobile objects and wireless awareness. Since it is unreasonable to expect the entire network to be available to the mobile user wherever and whenever the computational resources are required, it would be useful if a flexible framework can be deployed so that the mobile users can define their own hybrid clusters at runtime as needed. The hybrid cluster computing here is called the hybrid flexible cluster (HFC) computing because its framework is designed with the following flexibility features in mind.

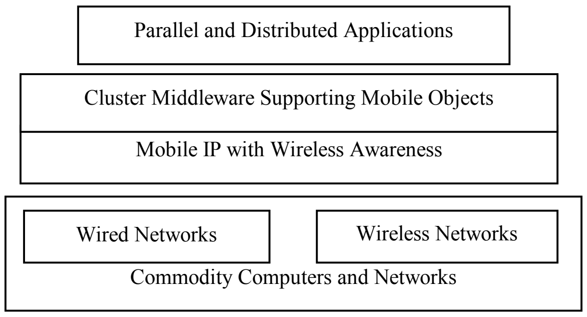
Parallel and Distributed Applications
Cluster Middleware Supporting Mobile Objects
Mobile IP with Wireless Awareness
Wired Networks
Wireless Networks
Commodity Computers and Networks

Figure 1. Hybrid cluster computing framework.

2.2. Commodity Flexibility

Instead of using the standard Ethernet network interface cards (NIC), some clusters in the existing research [1] chose a few high-performance NICs such as Myrinet cards for network connection. However, HFC uses standard Ethernet because of cost and availability. A commercial commodity wireless LAN such as Proxim RangeLAN is also used. HFC consists of personal computers and workstations running different versions of Windows and UNIX because of their popularity. Thus the commodity flexibility of the HFC is achieved.

2.3. Deployment Flexibility

The HFC framework embeds the techniques needed to build a flexible cluster that may be deployed easily by a mobile user given that computational nodes and relevant networking support exists for them. A hybrid cluster may be set up using any available node over the Internet. This has been made possible with the introduction of interactive advertisements by the potential computational nodes that wish to offer service to any cluster coordinator who broadcasts or multicasts a request seeking nodes for the hybrid cluster. With mobile IP the nomadic nodes can be connected into the cluster seamlessly across the Internet with roaming support. The cluster middleware is implemented in Java so that the middleware can provide the single system image of the hybrid cluster to any computer with different OS platforms.

2.4. Implementation Flexibility

The whole middleware package has been developed using Java. Since Java allows one to easily load and define runtime classes, new mobile objects can be implemented, migrated, and processed at the remote computers in the hybrid cluster. A mobile object is defined as an object communicating between two applications, such as middleware agents in the HFC case. The object is not required to be autonomous as with most mobile agent architectures. It normally does not have its own threads and is not capable of moving itself autonomously between processes. Moreover, no semantic information is exchanged regarding the capabilities of mobile objects with the exception of their interfaces. This makes them flexible for implementation.

Mobile object incorporates both the code, which may be loaded using a ClassLoader, and the state, which may be externalized using Java object serialization. Both are required elements to represent a mobile object.

2.5. Management Flexibility

A user is guided to create and manage a hybrid cluster via friendly interfaces. This liberates the user from identifying the networking topology of the cluster. For example, the available nodes identify and register themselves through advertisements at startup and also reply requests from the coordinator, which is a repeated polling operation for detecting new nodes that might become alive. Thus, the user can freely select the nodes to be included into the cluster. Wireless awareness [8] provides mechanisms for wireless media detection so that wireless characteristics can be managed automatically at the node startup time. Moreover, log-files for debugging and performance analysis are used for the management.

2.6. Extension Flexibility

The framework has been designed so that incremental changes to it can enhance the generality and usability of the hybrid cluster. The design is object-oriented and comprises clear interfaces and abstract classes that may be used to define new classes for the applications or the framework itself for extensions.

3. Prototype Implementation

Based on the framework design and flexibility features presented above, a prototype of the hybrid cluster is implemented. Since the cluster middleware is an essential component, first its structure is proposed, then its coordinator and computational agents are described, and finally its Java implementation are summarized. In addition, the hardware testbed of the hybrid cluster computing is described in details.

3.1. Structure of Cluster Middleware

Figure 2 illustrates the structure of prototype implementation of the cluster middleware. The two major parts, the coordinator and computational agents, communicate with mobile objects.

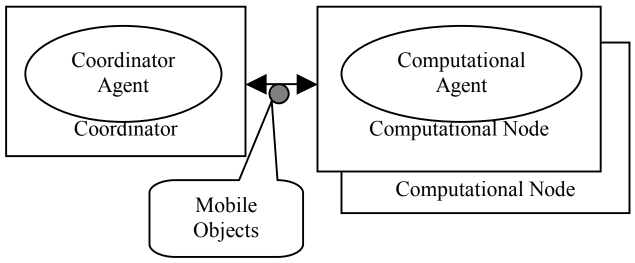

Figure 2. Structure of cluster middleware

Coordinator agent is the software that initiates the cluster, manages all the nodes that take part in the cluster processing, and coordinates all the functions related to job description and distribution. It may be used to coordinate the communication among the cluster nodes. It is referred to as coordinator or coordinator node in this paper.

Computational agent is the software that listens to the requests from different coordinators and services those requests, by being a part of the clusters, loading and executing the mobile objects that defines the distributed jobs submitted by the respective coordinators, and replying the results back.

3.2. Coordinator Agent

Normally, the mobile host acts as the coordinator and fixed nodes as computational stations. The coordinator coordinates the nodes in its corresponding hybrid clusters, i.e., it performs cluster setup and management. When a task comes, it dispatches jobs to the computational nodes according to task specifications and job distribution strategies. It also collects the feedback from the nodes and finalizes the task. The functionality of the main components of the coordinator agent is described below.

Cluster Setup. When coordinator agent is activated, it discovers all the connected nodes and their associated resources in its domain. When setup is called, it broadcasts or multicasts query messages, which also include the authentication codes, for the availability of computational nodes. According to the replies it collects, a hybrid cluster may be established. Its front end displays the current status of the hybrid cluster and allows the user to define a new cluster comprising a subset of nodes visible to this coordinator. There is also error-checking scheme for cluster setup.

Cluster Monitoring and Management. As some of the computational nodes leave the cluster, or the wireless link bandwidth and bit error rate change (wireless links get corrupted), the topology of the hybrid cluster changes. Therefore the coordinator has to collect the information about the nodes, and maintain the cluster status so that computing jobs can be dispatched accordingly.

Task Management. When the hybrid cluster has been setup, the user can use task manager wizard to specify the task or choose from the task type list, which includes all the task types the coordinator agent can deal with, and specify the parameters and provide the corresponding data or data files. Moreover, task monitoring is provided for multi-task scenario.

Job Distribution. Once the task has been presented, the coordinator agent may interactively or programmatically break the task into smaller sets and dispatch them to the computational nodes according to the current status of the hybrid cluster and predefined job distribution strategies.

Communication: Sending and Receiving Data. A basic functionality of the coordinator agent is pushing data to the computational nodes, which includes the mobile code object and data object that the nodes need for parallel computing, and pulling data from them to finalize the result. Two separate threads are spawned to accept data and emit data. Also the protocol for data exchange such as the data format and the port addresses are standardized for all the nodes in the cluster. Communication channels provide connection-oriented and connectionless communication between the coordinator and the nodes.

3.3. Computational Agent

Security and Availability in Node Setup. A front end for setting up the computational node and guaranteeing the secure usage of the computing resource of this node has been designed to provide (*i*) the availability of the current node for participation in a cluster; (*ii*) the authentication code editing and checking for joining clusters; and (*iii*) the values specifying the node's resource usage and the upper-bounds on the resource usage.

Information Service and Node Monitoring. A separate thread monitoring requests for information is spawned in the agent. Information regarding the node is supplied on demand to a query from the cluster coordinator in a pre-defined packet format. Both the query and corresponding reply are addressed to well-known ports. The query-service and node-monitoring information mainly includes but is not limited to: (*i*) cluster affiliation, associated coordinator, and job information; and (*ii*) CPU load, memory usage, and etc.

This component may interface with an SNMP agent for monitoring the computational node. The front end displays the information of the clusters in which the node has the membership. Also it shows the name of the job running on this node, and the availability of the computing resource offered to different clusters.

Execution Engine. Conceptually, this facility executes the jobs in the computational nodes, as submitted by the coordinator. The jobs will be submitted in the form of Java classes and loaded into the engine to do the necessary computation. Execution engine is just a set of classes that allow other classes to be specified, loaded and instantiated at runtime. Distributed jobs (mobile objects) are distributed to the respective nodes by the coordinator.

Communication: Sending and Receiving Data. The same functionality as that of the coordinator agent.

3.4. Middleware Java Packages

Three Java packages comprise the cluster middleware:
edu.rutgers.caip.hc.application The main classes are the two defined by the user, each identifying its problem domain. One class is the coordinator class; the other is the computational class. These two can be easily and flexibly implemented based on the abstract class interface provided by the package.

edu.rutgers.caip.hc.coordinator These are middleware classes for the coordinator: the GUI class interacts with users and the coordinator class, which retains the data and controls the respective cluster. Other supporting classes exist here too, which include node discovery, task manager, communication threads, and etc.

edu.rutgers.caip.hc.node The driver classes for the application's node. There is also a GUI class interacting with users and the node class, which holds the execution engine object and retains the job information for different clusters. In addition, there are classes for loading runtime mobile objects, monitoring node status, communicating with coordinators, and so on.

3.5. Testbed hardware

The testbed hardware comprises commodity devices. The topology of the testbed is shown as Figure 3. The fixed components of the testbed are a Cisco 3620 router, Cisco Catalyst 2916 switches, two personal computers with Windows NT 4.0 (*demain.rutgers.edu* and *tarquinia.rutgers.edu*), a workstation with Solaris 2.6 (*biko.rutgers.edu*), and Ethernet cables and NICs. The wireless components include Proxim spread spectrum wireless LAN. The mobile laptop with Windows 95 (*Kliu.rutgers.edu*) uses RangeLAN2 7400 PC cards to communicate with the rest of the networks via the RangeLAN2 7510 Ethernet Access Points as the base stations in the slave-master mode. Roaming is enable and mobile IP is supported [9].

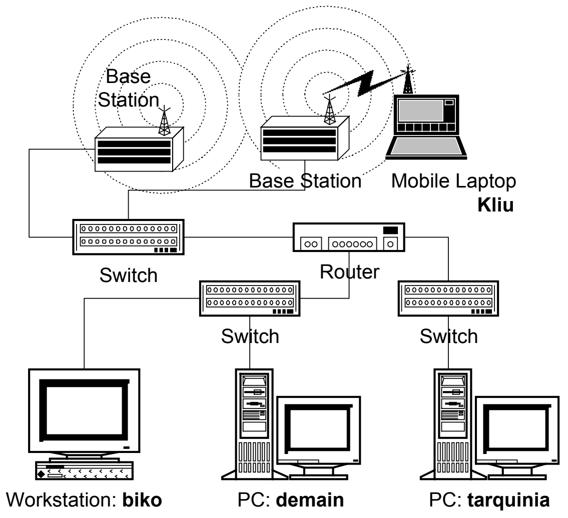

Figure 3. Topology of the testbed

4. Experimental Results and Analysis

Experiments have been carried out based on the prototype implementation to study the performance of hybrid cluster computing. We quantitatively analyze the experimental data, the trade-off between the communication load and computation load of the hybrid cluster computing. In the experiments, the mobile laptop, *Kliu*, acts as the coordinator of hybrid clusters. And there are three available computational nodes, *demian*, *tarquinia*, and *biko*, which are stationary computers.

Two kinds of application have been tested: (*i*) prime number generation, where the communication load between the coordinator and the computational nodes is not heavy but scales up with the problem size, and (*ii*) matrix multiplication that requires a lot of communication between the coordinator and the nodes for exchanging the matrix information. Both applications consume a great amount of computational resources and can be processed in the parallel and distributed manner.

4.1. Prime Number Generation

The task here is to determine all the prime numbers between 1 and a certain positive integer, which is defined as the problem size. The job load is distributed evenly among the computational nodes in the cluster. After the mobile object is instantiated on a node based on the byte stream of its Java class that is transmitted from the coordinator (*Kliu*), it asks the coordinator for the computing range. The coordinator sends the upper and lower ranges, and the node computes the prime numbers inside those ranges and transmits them back to the coordinator. Finally, the coordinator collects all the prime numbers from the nodes. Two-way handshake is used to guarantee a certain degree of communication reliability.

Before the computation begins, the coordinator starts a timer and after all the prime numbers have been collected it stops the timer. For each cluster size and problem size, ten cycles of experiments have been conducted. Then the approximate time range consumed in the whole computation is displayed and recorded as in Table 1.

Table 1. Time (ms) for prime number generation

Cluster Size	Problem Size	Total time	Time at Demian	Time at tarquinia	Time at Biko
1	5000	560	N/A		
	30000	4700			
	100000	26000			
	300000	276300			
	1000000	775400			
3	5000	2570	400	380	N/A
	30000	4400	2740	2090	
	100000	15700	13570	7900	
	300000	61390	59380	36400	
	1000000	368000	365000	283600	
4	5000	3800	250	210	200
	30000	6060	2270	1410	1600
	100000	13000	12200	9500	9110
	300000	45160	41200	29190	28200
	1000000	239000	235000	161700	167000

Because the mobile node *Kliu* is always the coordinator in the experiments, the case of cluster size 1 means that all computation is conducted on *Kliu* and no communication among the nodes is required. When cluster size is 3, the computational nodes are *demian* and *tarquinia*. *Biko* is added as another node to increase the cluster size to 4. Figure 4 illustrates Table 1.

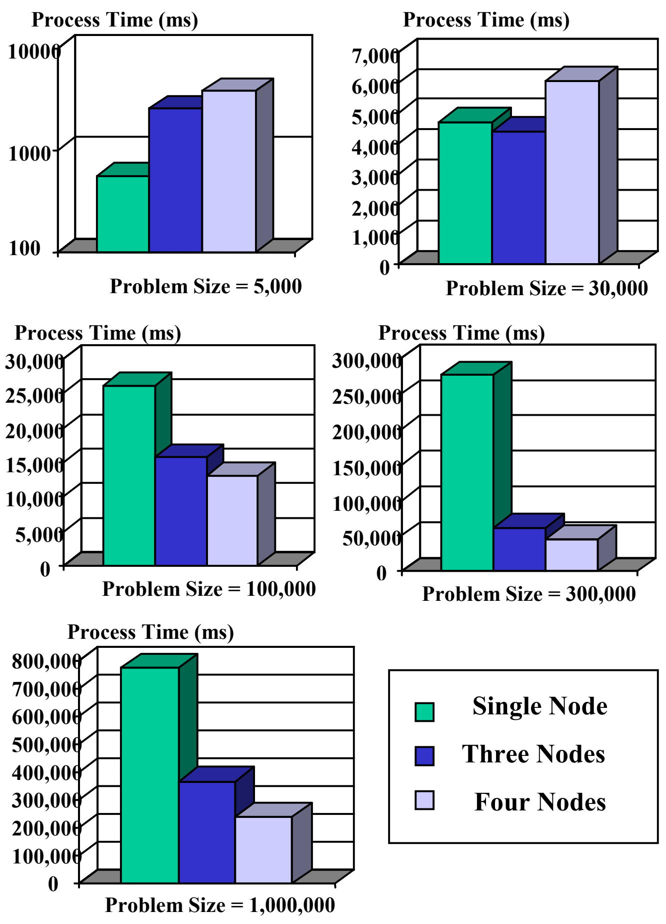

Figure 4. Time for in prime number generation

4.2. Matrix Multiplication

The task is to compute the product of two matrices with large dimensions. It requires heavy communication between the coordinator and the computational nodes to exchange the matrix information. The job load is also distributed evenly or almost so to the nodes of the cluster.

After the mobile object is instantiated on a computational node, it asks the coordinator for the information about the matrices. The coordinator sends the dimension and the data of the split matrices, and the node computes the resulting sub-matrices and feeds them back to the coordinator. Finally, the coordinator collects all the sub-matrices from the nodes and forms the result matrix.

Two-way handshake is again used for reliability. The timer is used as in the previous experiment.

The results show that the hybrid cluster computing does not enhance and sometimes even deteriorates the computing performance of the matrix multiplication for huge matrices because of the great communication overhead, especially over the wireless links.

4.3. Analysis of the Results

We compare the baseline computation time of the single-node case of prime number generation with that of the three- and four-node cases (Figure 5). The computing performance is analyzed in terms of the trade-off between the computational load and communication load.

When the problem size is small, the disadvantage caused by the communication overhead overrides the advantages gained by the high computational power of the nodes. Thus, the time consumed by processing the task in a distributed manner is several times longer than that of processing it on a single machine.

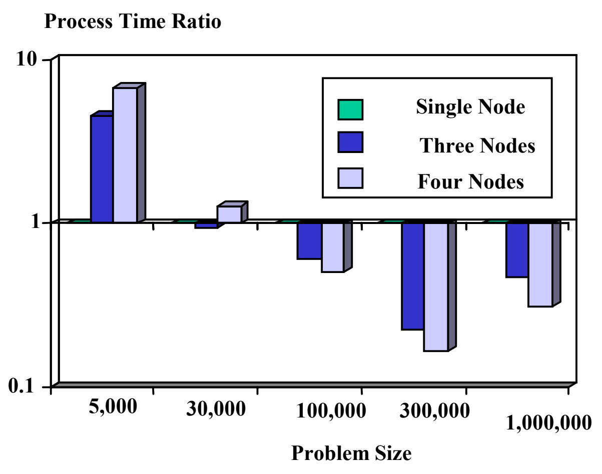

Figure 5. Time for in Prime Number Generation

When the problem size becomes medium, the advantages gained by the high computational power of the nodes overrides the disadvantage caused by the communication overhead. Therefore, processing jobs in a distributed manner becomes a better choice.

As the problem size grows, the communication load scales up. In this example, the coordinator has to gather a huge amount of prime numbers data from the nodes, so the performance of distributed computing decreases.

5. Conclusions

This paper contributes a framework for the hybrid cluster computing implemented with Java mobile objects and the mobile IP. The performance of hybrid cluster computing is studied, particularly the tradeoff between the communication and the computational load. Future work will extend the existing prototype with:

- Replacing node discovery and advertisement by a directory service protocol like SLP (Service Location Protocol) or LDAP (Lightweight Directory Access Protocol). This would allow a standard way of advertising and retrieving node information.
- Support for process migration. This would allow the process on one node to be transferred onto other nodes in the event that the first node desires to leave the cluster.
- Applying encryption schemes for the cluster security.

Acknowledgments

This research is supported by DARPA Contract No. N66001-96-C-8510, NSF KDI Contract No. IIS-98-72995, and the Rutgers Center for Advanced Information Processing (CAIP).

References

[1] D. E. Culler, A. Arpaci-Dusseau, and etc. Parallel Computing on the Berkeley NOW. *9th Joint Symposium on Parallel Processing*, Kobe, Japan, 1997.

[2] M. Baker and R. Buyya. Cluster Computing: the Commodity supercomputer. *Software-Practice and Experience*, 29(6):551-576, May 1999.

[3] D. Ridge, D. Becker, and etc. Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs. *Proceedings, IEEE Aerospace*, 1997.

[4] M. Weiser. Some Computer Science Problems in Ubiquitous Computing. *Communications of the ACM*, July 1993.

[5] T. Imielinski and B. R. Badrinath. Mobile Wireless Computing. *Communication of the ACM,* 37(1):19-28, October 1994.

[6] Y. Tseng and C. Tan. On Termination Detection Protocols in Mobile Distributed Computing Environment. *Proceedings of the International Conference on Parallel and Distributed Systems – ICPADS 1998*, pp. 156-163, 1998.

[7] H. Zheng, R. Buyya, and S. Bhattacharya. Mobile Cluster Computing and Timeliness Issues. *Informatica: An International Journal of Computing and Informatics*, 23(1), 1999.

[8] L. Cheng and I. Marsic. Wireless Awareness for Multimedia Applications. Accepted by *International Workshop on Wireless Networks and Mobile Computing*, Taipei, April, 2000.

[9] C. Perkins. IP Mobility Support. *Request for Comments (RFC) 2002*, October 1996.

[10] J. Farley. *Java Distributed Computing*. O'Reilly Publishers, 1998.

[11] J. Nelson. *Programming Mobile Objects with Java*. John Wiley & Sons, 1999.