International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:2, 2008

Performance Analysis of Parallel Client-Server Model Versus Parallel Mobile Agent Model

K. B. Manwade, and G. A. Patil

***Abstract*—**Mobile agent has motivated the creation of a new methodology for parallel computing. We introduce a methodology for the creation of parallel applications on the network. The proposed Mobile-Agent parallel processing framework uses multiple Java-mobile Agents. Each mobile agent can travel to the specified machine in the network to perform its tasks. We also introduce the concept of master agent, which is Java object capable of implementing a particular task of the target application. Master agent is dynamically assigns the task to mobile agents. We have developed and tested a prototype application: Mobile Agent Based Parallel Computing. Boosted by the inherited benefits of using Java and Mobile Agents, our proposed methodology breaks the barriers between the environments, and could potentially exploit in a parallel manner all the available computational resources on the network. This paper elaborates performance issues of a mobile agent for parallel computing.

***Keywords*—**Parallel Computing, Mobile Agent.

I. Introduction

TRADITIONAL solutions to large-scale computation involve massive supercomputers consisting of multiple processors. Data is processed in a pipelined fashion that can incorporate multiple machines and numerous computing stages. The limitations to this approach include flexibility, scalability, cost and fault tolerance. Our proposed work is focused on a new approach for computation that utilizes a computing cluster; a network of small personal computers connected via a network medium. In this system, a processing task is partitioned and divided among a family of lightweight agents [1]. These agents are distributed throughout the cluster and compete for computing resources.

This approach of computation is advantageous in that the system operates as an autonomous entity. Agents execute as a collaborative team, working around node failures and system bottlenecks. Additional computing resources can be added and exploited dynamically, enhancing both the flexibility and scalability of the system.

K. B. Manwade is with the Department of Computer Science & Engineering., Shivaji University, Kolhapur 416113, (MS), India (Corresponding author to provide phone: 91-9975634528; e-mail: mkarveer@gmail.com).

G. A. Patil is with the Department of Computer Science & Engineering., Dr. .D. Y. Patil College of Engineering, Kolhapur, 416006 (MS), India (e-mail: gasunikita@yahoo.com).

The goal of this effort is to implement parallel computing using a family of mobile agents. To this end, task parallelism serves as the foundation for our agent-based parallel computing model. At the system level, task parallelism represents an object-oriented design strategy, whereby both data and the operations on those data are encapsulated in a single entity. Adhering to this paradigm, each task has the ability to operate independently, yet communicate and exchange data with other tasks.

Agent task patterns are of particular interest in parallel processing [2], where the focus is on partitioning and delegating tasks among agents. The Master-Slave [3] pattern is a common task design model incorporated in a broad domain of parallel applications. This Master-Slave model is based on a divide and conquers strategy in which a master delegates tasks to one or more slaves that in turn are distributed throughout the system and work in parallel. The standard agent-based implementation involves master and slave agents. The execution sequence is as follows:

1) The master agent creates a slave agent.
2) The slave agent moves to a remote host and performs a task.
3) The slave agent returns to the master.
4) The slave agent passes task results to the master.

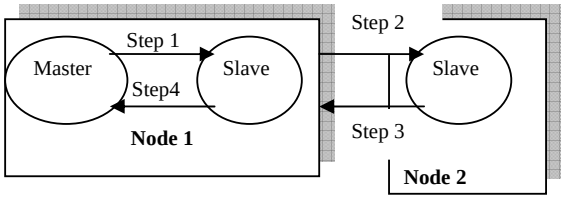

Fig. 1 Event diagram of MABPC Model

II. Overview of MABPC

Fig. 1 shows the basic block diagram of MABPC. Each node of the network has an agent execution environment (AEE)**,** which is responsible for accepting and executing incoming agents. A client called Master agent [4], submits the agent on behalf of the user to the AEE.

A user, who wants to perform a task, submits the task to the master agent on the graphical user panel system. The master agents then divide the task into subtask and assign it to individual Slave agent. After assigning the task it dispatches [6] the slave agent i.e. mobile agent to the specified servers.

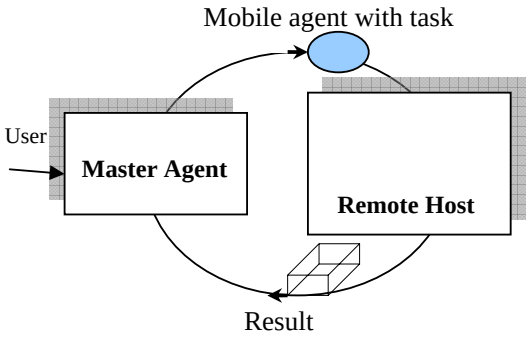

Fig. 2 Block Architecture of MABPC Model

At remote site the slave agent does the given task using the CPU resource of remote server. A slave agent then constructs the message and embeds the result into it and sends it to master agent using master agents proxy. The master agent who is waiting for the result from slave agent receives the message and extracts the result and combines all results together. Then it calculates the turnaround time for the computation which will be used for comparison with client server model later.

III. Architecture and Design of MABPC System

In our system the master agent dispatches one or more MAs, each with its parameters. The MA visits specified server to perform the required task. After completion of task the MA returns result of the operations performed to the master agent. Two types of agents are implemented. They differ mainly in the different roles they can cover and/or services they can offer computation.

1. **Master agent** which act as task handler in the system. After getting the parameters from the user interface, it creates the slave agents. Then it reads the database for the list of servers, and then to check the availability for server it sends the echo agent. After preparing the available server list it sends the slave agents to those servers.
2. **Slave agent** which act as mobile agent and migrate from one host to another host and does the assigned task on behalf of master agent.

The next section gives details of our MABPC [4] application, the architecture of which is shown in Fig. 3.

A. Master Agent

The master agent plays the role of task handler. It dispatches [6] two types of mobile agents first of it dispatches the echo agent to all servers from list to check the availability of the server, which provides the reliability to the system. After collecting the result from echo agents it maintains the available servers list which will be used further. After that it creates the slave agents and assigns the parameters from the user interface, and then it dispatches the slave agents to the servers from available server list database. Then it handles the messages from the slave agent which is executing at remote site and extracts the result from the message. Once it receives the result from all slave agents then it displays it.

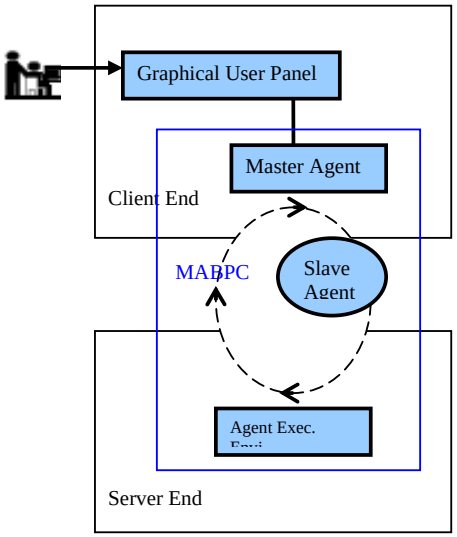

Fig. 3 Architecture of parallel computing application

B. Slave Agent

The slave agent is the mobile agent which migrates from client machine to server machine on behalf of user. At remote site it does the given task and sends the result to master agent by embedding it into message. After sending the result the slave agent disposes itself.

C. Agent Execution Environment (AEE)

The Agent Execution Environment (AEE) consists of the Tahid server and Java 1.1 runtime for execution of java mobile agents. Each server and the client in the proposed model is having this server. This environment is developed by the IBM.

IV. Implementation and Performance Study

We have implemented the matrix multiplication for performance analysis of two network computing paradigms. The matrix multiplication task is divided on row wise basis. Based on number of available servers and dimension of matrix the block size is decided. The matrix of block size is assigned to each mobile agent. For execution of mobile agent on remote machine the Tahid server is used. The mobile agent contains the data such as matrix class, index number of the agent & the proxy of the master agent. The data flow for the mobile agent is as shown in the Fig. 4.

The CS [7] implementation consists of multi-threaded client and multi-threaded servers. The client and the server have been implemented in Java using socket [5]. In CS model matrix class is passed from client side to server side and same class with result is returned from server to client as shown in Fig. 5.

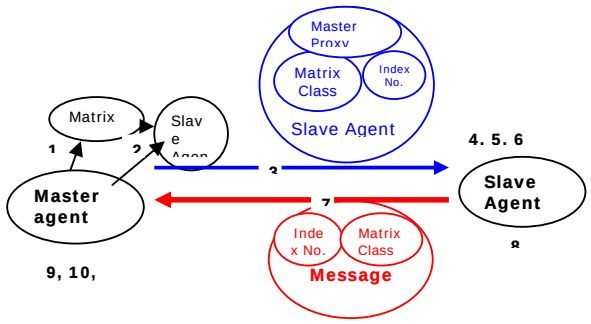

Fig. 4 Parameter passing in MA model

The execution flow of MA model is as shown below,

1. Initialize Matrices Class
2. Create Slave Agent
3. Dispatch Slave Agent
4. Extract the (3) parameters
5. Call matrixClass.Operation()
6. Create Message object
7. Send massage
8. Dispose itself
9. Extract Index No. & matrixClass.
10. Do assignment as this.matrixClass[index]=matrixClass
11. matrixClass[index].ReadResult()

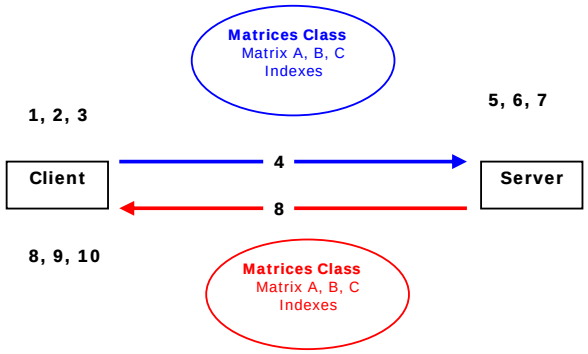

Fig. 5 Parameter passing in CS model

The execution flow of CS model is as shown below,

1. Initialize matrix A & B
2. Create MatrixClass object with A, B & indexes as parameter
3. Create socket connection
4. Send / write this object to socket
5. Read matrixClass object
6. Call Operation() method
7. Send matrixClass object (which contains result now)
8. Read result
9. Join threads
10. Calculate response time

We used various application parameters that influence performance, such as, size of CS messages, size of the MA, number of remote information sources, etc, and performed experiments to study their effect on performance. We used trip time, i.e., time elapsed between a user initiating a request and receiving the results, as the metric for performance comparison. This includes the time taken for agent creation, time taken to execute task and processing time to extract the required information.

We have performed experiments to determine:

(a) **The effect of data size on trip time:** The number of servers was kept constant and the dimensions of the matrices are varied. The turn around time for this variation was measured for different dimensions from 100 to 100,000.

(b) **The effect of number of servers on trip time:** The dimensions of the matrices are kept constant and the number of servers is varied. The trip time was measured for different number of servers from 3 to 45. Results are shown in Figs. 6 and 7, from which the following observations can be made:

- The performance of the MA based application remains the same for different matrix sizes while the performance of the CS based application degrades with increase in data size.
- CS implementations perform better than MA implementations for matrix dimension less than 100.
- MA performs better than CS when the matrix dimension is greater than 200.

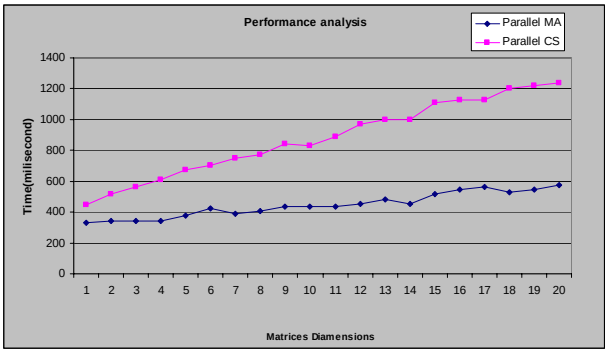

Fig. 6 Turn around time for varying matrix dimension

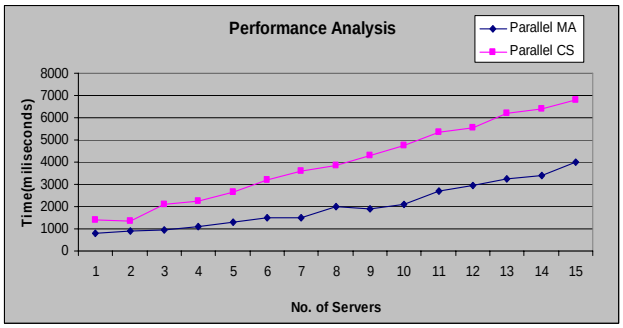

Fig. 7 Turn around time for varying number of servers

V. Conclusion

Our experiments suggest that CS implementations are suitable for applications where a small amount of data (matrix

of size less than 100) is to be processed on remote servers (less than 3). For large dimensional matrices multiplication on large number of servers MA gives better performance. Scalability being one of the needs of net-centric computing, we find that MAs are an appropriate technology for implementing network centric applications.

ACKNOWLEDGMENT

I express my deep sense of gratitude and appreciation towards my research guide Prof. G. A. Patil for his continuous inspiration and valuable guidance in throughout of my Project work.

REFERENCES

[1] W. Cockarine, M. Zyda, "Mobile Agents", Manning, Greenwich, 1998.
[2] Panayiotou Christoforos, George Samaras, "Parallel Computing Using Java Mobile Agents", *Greece.*
[3] Jason Byassee, "Agent-Based Distributed Parallel" Processing, 2001.
[4] Manwade K.B, Prof. Patil G. A., Mobile Agent Based Parallel Computing, NCTIAC-2007, Mumbai.
[5] Aderounmun, G.A. (2004). "Performance comparison of remote procedure calling and mobile agent approach to control and data transfer in distributed computing environment". *Journal of Network and Computer Applications*, 113 – 129.
[6] Danny B. Lange and Mitsuru Oshima, "Seven Good Reasons for Mobile Agents", Communications of ACM, vol. 42, no. 3, March 1999.
[7] R. Orfali and D. Harkey, "Client/server Programming with Java and *CORBA"*, Wiley, 1999.

K. B. Manwade received the B.E. degree in Computer Science & Engineering from Walchand College of Engineering, Sangli, India in 2004. He is doing his MTech in Computer Science & Technology at Shivaji University, Kolhapur, India. From 2004 to 2006, he was working as Lecturer at Annasaheb Dange College of Engineering & Technology, Ashta, India. He has published various papers in the area of Distributed Computing & Parallel Computing.

G. A. Patil received the BE degree in Computer Science & Engineering. He has done ME in Computer Science from Walchand College of Engineering, Sangli. From 1992 to till date he is working as Assistant Professor at D. Y. Patil College of Engineering, Kolhapur, India. He has published various papers in the area of Cluster Computing & Information Security.