Multiplying Strings of N Matrices using Mobile Agents

Phyo Phyo Ko, Yin Ko Latt
University of Computer Studies, Magway
phyophyoko@gmail.com

Abstract

Mobile agent is one that is not bound to the system for their execution on a network. Each mobile agent can travel around the network and perform the tasks. We propose the use of mobile agent in parallel computation models such as pipelined model in matrix multiplication algorithm. A matrix multiplying is common in many real-world problems and it is a simple, general purpose way to improve the performance of many tasks. And it is also simple parallel structure that can be used to speed up many problems and can eliminate load unbalance problems. Mobile agent efficiently support parallel and distributed computing. The main aim of this paper is to save time for multiplying matrices and less cost using mobile agent. Mobile agent will be applied in multiplying matrices to save time than other. Agent is worked in two main parts, Master and Worker agents.

1. Introduction

Mobile Agent technology is a new networking technology that deals with both form of logical and physical mobility [6]. Mobile agents are an effective choice for many applications for several reasons, including improvement in latency and bandwidth of client-server applications and reducing vulnerability to network disconnection.

Mobile agents are being used already in a variety of Internet-based distributed computing applications: web database, cooperative environment, and information gathering systems, electronic commerce systems and so on.

The use of parallel processing as a means of providing high performance computational facilities for large-scale and grand-challenge applications has been investigated widely [5]. In some application areas, parallel computer may be easier to program, give performance unobtainable in any other way, and might be more cost-effective than serial alternatives. Serial computer have a number of conceptual drawbacks in some of the application. The main reason for creating and using parallel computer is that parallelism is one of the best ways to overcome the speed bottleneck of a single processor.

Mobile agent that makes its much easier to design, implement, and maintain distributed systems. It reduces network traffic and provides an effective means of overcoming network latency. Using mobile agent by parallel processing gives us save-time and cost-effective in larger problems.

In this paper mobile agent is developed on aglet software. This paper describe the development of parallel application using Java Mobile Agent based pipelined computing. The mobile agent execution environment used to realize is based on the Aglet Technology. Traditionally, software has been written for serial computation: to be executed by a single computer having a single Central Processing Unit (CPU); problems are solved by a series of instructions, executed one after another by the CPU. Only one instruction may be executed at any moment in time. In the simplest, parallel computing is the simultaneous use of multiple compute resources to solve a computational problems.

This paper is organized as follows. In section 2, a description of related work is given. Section 3 describes the background theories. Section 4 describes the proposed system design. Section 5 describes the experimental result and section 6 describes the conclusion.

2. Related Work

There are a few research projects that are similar to our mobile agent based pipeline. In [6], the performance of mobile agent in parallel and distributed computing in which developed and tested a prototype system with several applications such as computing PI value, generating prime number, an image processing and sorting numbers. H.A. Thant [5] presents the efficient load balancing method for cluster based parallel applications using mobile agents. In the system proposed in [2], a novel mobile agent based “push” methodology from the perspective of application. In the method, users declare their computation-bound jobs as autonomous agents. Obeloeer et.al [3] presented FLASH (Flexible Agent System for Heterogeneous Cluster) system, which use a mobile agent have the capabilities to travel through a heterogeneous cluster

and to fulfill jobs on the visited computation nodes. Each mobile agent can travel anywhere in the web to perform its tasks.

3. Background Theory

3.1. Mobile Agents

Mobile agents are processes that dispatched from a source computer to accomplish a specified task [4]. After its submission, the mobile agent proceeds autonomously and independently of the sending client. When the agent reaches a server, it is delivered to an agent execution environment. Then, if the agent possesses necessary authentication credentials, its executable parts are started. To accomplish its task, the mobile agent can transport itself to another server, spawn new agents, and interact with other agents. Upon completion, the mobile agent delivers the results to the sending client or to another server.

There are seven good reasons for using mobile agent. They are [4]:

1. They reduce the network load. Distributed systems often rely on communication protocols that involve multiple interactions to accomplish a given task. This is especially true when security measures are enabled. The result is a lot of network traffic. Mobile agents allow you to package a conversation and dispatch it to a destination host, where the interactions can take place locally as shown in Figure 1. Mobile agents are also useful when it comes to reducing the flow of raw data in the network. When very large volumes of data are stored at remote hosts, these data should be processed in the locality of the data rather than transferred over the network.

2. They overcome network latency. Mobile agents offer a solution, because they can be dispatched from a central controller to act locally and directly execute the controller's directions.

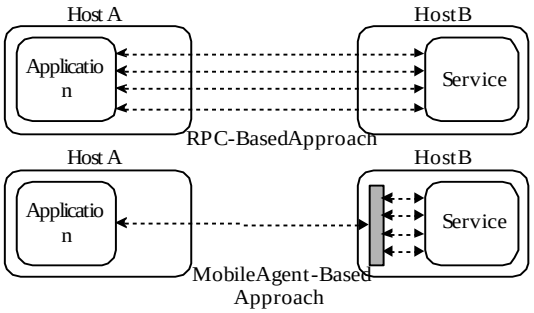

Figure 1. Mobile Agents and Network Load Reduction

3 They encapsulate protocols. When data are exchanged in a distributed system, each host owns the code that implements the protocols needed to properly code outgoing data and interpret incoming data, respectively. Mobile agents, on the other hand, can move to remote hosts to establish "channels" based on proprietary protocols.

4. They execute asynchronously and autonomously. Often, mobile devices must rely on expensive or fragile network connections. Tasks that require a continuously open connection between a mobile device and a fixed network probably will not be economically or technically feasible. To solve this problem, tasks can be embedded into mobile agents, which can then be dispatched into the network. After being dispatched, the mobile agents become independent of the creating process and can operate asynchronously and autonomously as shown in Figure 2. The mobile device can reconnect at a later time to collect the agent.

5. They adapt dynamically. Mobile agents have the ability to sense their execution environment and react autonomously to changes. Multiple mobile agents possess the unique ability to distribute themselves among the hosts in the network so as to maintain the optimal configuration for solving a particular problem.

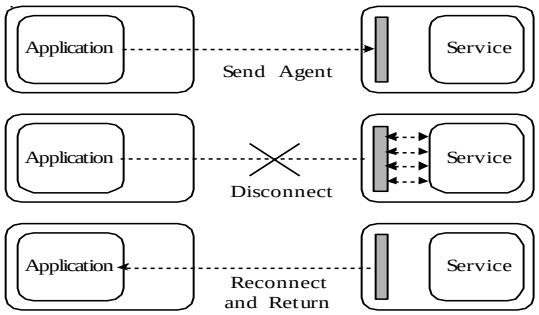

Figure 2. Mobile Agents and Disconnected Operation

6. They are naturally heterogeneous. Network computing is fundamentally heterogeneous, often from both hardware and a software perspective. Because mobile agents are generally computer and transport layer independent and are dependent only on their execution environment, they provide optimal conditions for seamless system integration.

7. They are robust and fault-tolerant. The ability of mobile agents to react dynamically to unfavorable situations and events makes it easier to build robust and fault-tolerant distributed systems. If a host is being shut down, all agents executing on that machine will be warned and given time to dispatch and continue their operation on another host in the network.

3.2 Pipeline Architecture

A key requirement for pipelining is the ability to send messages between adjacent processes in the pipeline. This suggests direct communication links between processors onto which adjacent processes are mapped. An ideal interconnection structure is a line or ring structure such as a line of processor connected to a host system as shown in Figure 3. The seemingly inflexible line configuration is, in fact, very convenient for many applications, yet at very low cost [5].

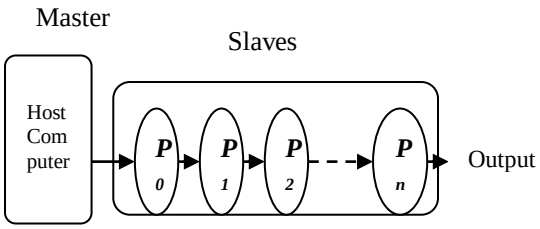

Figure 3. Basic Architecture of Pipeline Computing

Each task on each processor receives input data from its predecessor, performs its computation, and sends its output to its successor. The first step reads external input and the last generates the final output [7].

4. Proposed System Design

There are two main components in our system. These components are:
(1) Master and
(2) Worker agent.

They are worked by using dynamic programming algorithm to minimize the number of operations. Dynamic programming algorithm work at master agent.

4.1. Dynamic Programming

In essence, dynamic programming calculates the solution to all sub-problems. The computation proceeds from the small subprograms to the larger sub-problems, storing the answers in a table. The advantage of the method is in the fact that once a sub-problem is solved, the answer is stored and never recalculated.

Consider the evaluation of the product of n matrices [1]

$$M = M_1 * M_2 * \ldots\ldots\ldots\ldots * M_n,$$

Where each M_i is a matrix with r_{i-1} rows and r_i columns. The order in which the matrices are multiplied together can have a significant effect on the total number of operations required to evaluate M.

Trying all possible orderings in which to evaluate the product of n matrices so as to minimize the number of operations is an exponential process, which is impractical when n is large. However, dynamic programming provides as O(n3) algorithm. Figure 4 show the dynamic programming algorithm for computing the minimum cost order of multiplying a string of n matrices, $M = M_1 * M_2 * \ldots\ldots\ldots\ldots * M_n$, where n is number of matrices, r_{i-1} and r_i are the dimensions of matrix M_i and m_{1n} is the required minimum cost for the matrix order.

```
Begin
    for i ← l until n do
       mij ← 0 ;
    for l ← 1 until n-1 do
       for i ← 1 until n-l do
       begin
            j ← i+l;
            mij ← MIN(mik+mk+1,j+ri_1*rk*rj);
                   i<=k<=j
      end;
  write m1n
end
```

Figure 4. Dynamic Programming Algorithm for Minimum Cost Matrix Order

4.2. Matrix Sequence Multiplication

The multiplication of matrix sequence $M_1 * M_2 * \ldots\ldots * M_N$ can be performed by using the following algorithm according to the computation architecture.

```
//generate the random number for dimension of N matrices
1. for (1=0, i<=N; i++)
    rc[i]=random();
//generate the minimum computing cost matrix order by using dynamic programming
2. (…((( M1* ((( M2* (M3*((M4*M5)*M6)))*M7)*M8))….*MN-1)*MN)
//create communication objects for first worker w1
3. create (cli_socket1);
//dispatch the dimension for first two matrices to worker w1
4. send (d1, d2, NextIPAddr, Dimension);
//receive the final result from last worker wn
5. receive (Result, wN)
```

Figure 5. Master Algorithm for Sequence Matrix Multiplication

The Master Algorithm for Sequence Matrix Multiplication can be shown in Figure 5. The master algorithm, which generate the dimensions, rows and columns, for all matrices N and find

matrix order which use the minimum computing cost on the dimensions of the matrices by using the dynamic programming. To start the computation the master create the connection object for the first worker and send the required data for the computation to the first worker and have to receive the final computing result from the last computing step.

The Worker Algorithm for Sequence Matrix Multiplication can be defined in Figure 6. This algorithm intends for each computing worker wi which corporate in computing the Matrix Multiplication.

Since the applications considered for this system are executed in the pipelined form, interconnection between the computing workers must be configured to form the pipelined connection. The computing workers in the pipeline configuration can be both client and master conditions. In transferring the computed result from current worker to next worker, the current worker must be the client form to dispatch the result to next computing step and the next worker must be the server form in receiving the previous result to continue the computing steps.

```
//create the server object for each worker wi and listen their server port
1. ServerSocket.open();
//receive the corresponding data for the computing steps and generate the random number for matrices and multiply the matrix
2. if (first worker)
    Receive(Dimensions, NextIPAddr);
    Mi[rc[i]][rc[it+1]]=random();
    Mi+1[rc[i+1]][rc[i+2]]=random();
    R1=Mi*Mi+1;
3. else if (not first worker)
    Receive(R1, Dimensions, NextIPAddr);
    Mi=random();
    matrix_multiply(Ri-1, Mi, Ri);
//create the client object to connect next worker wi+1
4. create (cli_socketi+1);
//send the sub multiplication result to next worker wi+1
5. send (Ri, wi+1);
```

Figure 6. Worker Algorithm for Sequence Matrix Multiplication

In this computation, worker for the first computing step receives the dimensions for the matrix sequence from *Master*, generate the two matrices according to the corresponding dimensions to compute these two matrices and sends the computed result including the dimensions to next worker with the client object to next pipelined step. The next worker which received the result from previous worker generate matrix with its corresponding dimensions and compute with the previous result matrix and dispatches the result to next worker with client object for next computing step. The computing cycle is running repeatedly until the last computation is finished.

The dataflow and workflow diagram for sequence matrices multiplication can be depicted in Figure 7.

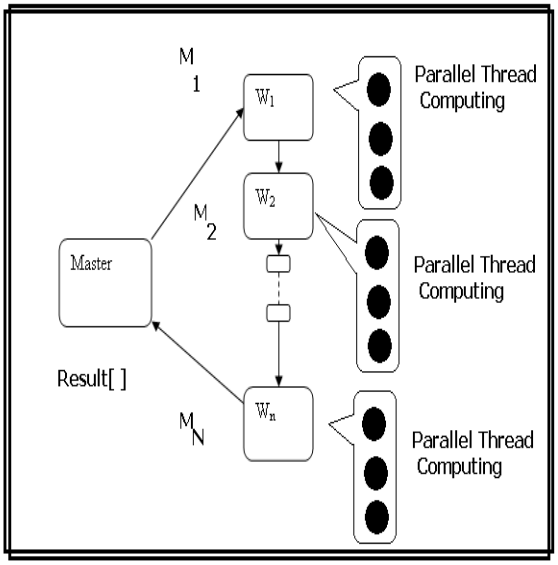

Figure 7. Sequence Matrices Multiplication

5. Experimental Result

In this section, experimental results of our system are described to evaluate the minimum cost matrix order by using dynamic programming algorithm. Multiplying matrices by using dynamic programming algorithm can less operation and cost.

Assume that, the implication of a p*q matrix by a q*r matrix, require pqr operations as it does in the "usual" algorithm, and consider the product.

M = M_1 * M_2 * M_3 * M_4
[10*20] [20*50] [50*1] [1*100]

Where the dimension of each M_i are shown in the bracket.
M in the order,
$M_1 * (M_2 * (M_3 * M_4))$
Require 125000 operations, while evaluating M in the order
$(M_1 *(M_2 * M_3)) * M_4$
This order requires only 2200 operations by using dynamic programming algorithm.
Solution:
M = M_1 * M_2 * M_3 * M_4
[10*20] [20*50] [50*1] [1*100]
M1 * (M2 * (M3 * M4))
[50*1] [1*100] = 5000

[20*50][50*100] = 100000
[10*20][20*100] = 20000

Require 125000 operations.

The order by using dynamic programming algorithm is

(M1 * (M2 * M3)) * M4
[20*50][50*1] = 1000
[10*20] [20*1] = 200
[10*1] [1*100] = 1000

Require 2200 operations by applying dynamic programming algorithm.

6. Conclusion

We evaluated and analyzed the usage of mobile agent in parallel computing models. It has shown that the better performance obtained as the number of agents increased. When one's resource is not enough for particular problems, mobile agent is one of the choices for distributed computing. In order to evaluate the benefit of mobile agents, our future work will concern a deeper analysis of the mobile agent based parallel computing by means of many matrix numbers. The more the matrix number increase, the better the utilization by using mobile agent.

REFERENCES

[1] A.V. Aho, J.D. Ullman and J. E. Hopcroft. The Design and Analysis of Computer Algorithms. Addison Wesely, 0029.

[2] B. Wims and C. Xu. A novel mobile agent based "push" methodology from the perspective of application, in Proceedings of the 24th EUROMCRO Conference, 1998, pp.1250-1254.

[3] C. Grewe, H. Pals and W. Obeloeer, "Load Management with Mobile Agent", in Proceedings of the 24th EUROMCRO Conference, 1998, pp.1005-1012.

[4] D.B. Lange and M. Oshima. Programming and Deploying Java Mobile Agents with Aglets. Addison Wesely 1998.

[5] H.A. Thant. "Efficient Load Balancing Method for Cluster Based Parallel Applications Using Mobile Agents", in Proceedings of the 3rd International Conference, 2005, pp.97.

[6] K.M.L. Tun. "Parallel and Distributed Computing Models for Mobile Agents", in Proceedings of the 3rd International Conference, 2005, pp.87-88.

[7] N.N. Oo. "Parallel Computing in Solving Numerical Problems Using Java Mobile Agent", in Proceedings of the 4th International Conference, 2006, pp.519-520.