Predicting X-Tree network performance using the Jade environment

Li Xining and Brian Unger
Department of Computer Science
University of Calgary
Calgary, Alberta
Canada

ABSTRACT

Jade provides an integrated set of tools which are designed to support the development of distributed software and systems. The Jade environment provides tools for the design, implementation, debugging, testing, maintenance, simulation, and performance analysis of distributed, concurrent programs. A network topology called X-tree has been implemented and simulated using this Jade environment. This paper presents an overview of the Jade environment, the X-tree network topology and a robust routing algorithm for this topology. The performance of the X-tree topology is also discussed.

1. INTRODUCTION

As computers have become smaller, cheaper, and more numerous, people have become more and more interested in connecting them together to form networks and distributed systems. In any network there exists a collection of machines intended for running user programs, and a number of transmission lines for exchanging messages. Broadly speaking, there are two general types of network topologies, one is the common bus structure (broadcast channels), the other is the interconnection structure(point-to-point channels).

When a point-to-point subset is used, an important design issue is what the interconnection topology should look like. Fig.1 shows several possible topologies. Five desirable properties of an interconnection topology can be defined:

(1). The distance of the network increases much more slowly when the network is extended to arbitrarily many nodes. Ideally, if N is the number of nodes, D the distance of the network, then

$$\lim_{N\to\infty} \frac{D}{N} = 0,$$

where the distance between a pair of nodes is calculated as the minimum number of communication lines needed to convey a message from one to the other. The distance of a network, i.e. D, is the longest one among distances of every pair of nodes in the network.

(2). There is a fixed constant P, independent of the overall size of the network , such that

$$\forall \text{ node} \in \text{network} \quad \text{degree(node)} \leq P.$$

(3). The routing algorithm is easy to realize and independent of the extension of the network.

(4). When some nodes or communication lines have failed, the network can still function properly, although with lower performance.

(5). The traffic load distribution is uniform for all nodes in the network.

The relative performance of the network topologies of Fig.1 can be classified as shown in Fig.2 using these five properties.

In this report, We discuss the performance of the X-tree topology [1] using a specific routing algorithm. A model of communication in X-tree has been implemented using the Jade system, and simulation results are presented.

2. THE JADE DISTRIBUTED SOFTWARE PROTOTYPING ENVIRONMENT

The Jade environment can be described in terms of four functional levels: a *hardware level*, a *kernel level*, a *programming level*, and a *prototyping level*. The facilities available at each level include those provided at lower levels. The prototyping level will thus provide the Jade user with facilities from the programming level, as well as tools that specifically support prototyping.

The three major areas of research in Jade, i.e. simulation, distributed systems, and the user interface, all support the higher environment levels.

The *hardware level* consists of different networks of different kinds of computers. Currently, a 10Mbs token passing ring network (Pronet) of Vax 11/780s running Unix 4.1 provide the host resources. The Jade user workstations are Corvus Concepts connected via the Corvus 1Mbs Omninet, an Ethernet like network. One of the Vaxes is connected to the Omninet which enables communication between any Vax and any of the

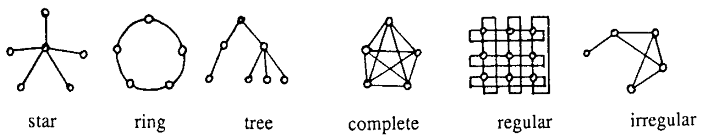

Fig.1 Several network topologies

	star	ring	tree	complete	regular	irregular
$\lim_{N\to\infty} \frac{D}{N} = 0$	yes	no	yes	yes	yes	?
$\forall$ n $\in$ network $d(n) \leq P$	no	yes	yes	no	yes	?
easy routing		yes	yes	yes	yes	yes
connec-tivity	bad	bad	bad	good	good	?
uniform loads	no	yes	no	yes	yes	?

Fig.2. Performances of some network topologies

workstations.

The *kernel level* consists of the Jade Inter-Process Communication facility called Jipc (pronounced as "gypsy"). Jipc supports message passing between processes which reside on any of the Vaxs or Corvus workststions via a synchronous protocol. A stand alone version of this kernel has been developed for the Corvus workstation and several sub-projects are underway to port the Jipc kernel to other M68000 based computers. A Unix version of Jipc has also been developed to enable message passing between Unix processes, and between a Unix process and workstation processes.

The Jipc protocol, its rationale, and experience with this message based protocol are presented in [2]. Jipc is based on the Thoth [3] communication protocol which includes blocking "send", "receive", "receive_any", "forward", and "reply" primitives. Interfaces to Jipc currently exist for five programming languages: Ada, C, Lisp, Prolog, and Simula. Distributed programs whose components are written in any combination of these languages can be developed and tested, e.g. one component in Ada, two in C, one in Lisp, several in Prolog, etc..

The *programming level* consists of the Unix environment with additional tools either in place or planned to support: 1) cross compilation and downloading for the Corvus workstations, 2) the monitoring and debugging of distributed programs, 3) a multi- media user interface to Jade, 4) general simulation languages and package, 5) an online document preparation system and 6) information management for projects and target systems.

The programming languages currently supported include Ada, C, Lisp, Prolog, and Simula on Vax/Unix and C on the workstations. Cross compilation and downloading of Prolog and Simula are under development. The user interface includes a window system that provides a rectangular area, or window, on the workstation screen for each process that the user wishes to interact with, or observe. Unix processes, perhaps residing on different Unix hosts, can be controlled via different windows, as well as, processes that reside on user's or other workstations.

A tool which graphically animates Jipc message interactions among the concurrent processes of a distributed computation has also been implemented [4]. Each message interaction, such as "send" or "receive", has a corresponding static graphical representation. The execution of a distributed computation can then be monitored as an animated sequence of these static images on the Corvus workstations.

Other general programming tools are planned, such as speech input and output; interactive debugging and profiling; a document preparation system; and an information management system [5].

The *prototyping level* supports the modelling and simulation of target distributed systems, including the target embedded computer system. Execution of the distributed program under development by the modeled target system can be simulated, including the system that are external to the embedded computer system. Performance information can be collected, displayed, and manipulated interactively [6] [7].

3. X-TREE AND ITS ROUTING ALGORITHM

3.1. What is X-tree?

X-tree is a kind of binary tree. The difference between X-tree and a general binary tree is that each layer of X-tree is connected into a ring, as shown in Fig.3.

Each node in X-tree has a unique identifier, an integer number. If we chose a node arbitrarily and define it as "me", then

```
parent = me/2;
left-brother = if (special(me))
                    then me*2-1 else me-1;
right-brother = if (special(me+1))
                    then (me+1)/2 else me+1;
left-child = me*2;
right-child = me*2+1;
```

where the boolean function special(x) will return true if x=2**i, else false.

The distance of X-tree is

$$D = 2*\log (N+1) - 4 \quad (N \geq 7)$$

and

$$\forall \text{ node} \in \text{X-tree} \qquad \text{degree(node)} \leq 5$$

3.2. Routing algorithm

There are three classes of routing algorithms, *static, centralized,* and *distributed* routing. *Static* routing is a simple algorithm which is widely used. Each node maintains a table with one row for each possible destination. A row gives the best, second best, third best, etc, outgoing line for that destination with a relative weight.

Before forwarding a packet, a node chooses among the alternatives, using the weights as probabilities. The tables are loaded into every node before the network is brought up, and not changed thereafter. So if the topology and message traffic changes dramatically and often, this algorithm will not perform well.

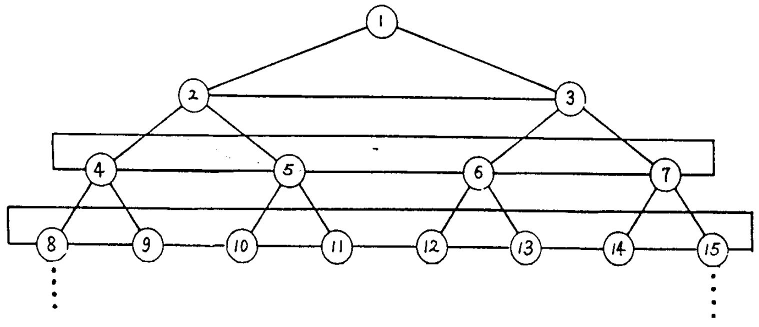

Fig.3. X-tree topology

Centralized routing is similar to static routing in that each node maintains a table telling how to forward packets. The difference between them is how the routing tables are constructed. When centralized routing is used, somewhere within the network there is a routing control center (RCC). Periodically, each node sends status information to RCC. The RCC collects all this information, and then, based on its global knowledge of the entire network, computes the optimal routes from every node to every other node, and then distributes these new tables to all the nodes. There are some disadvantages in centralized routing algorithm, one of them is heavy concentration of routing traffic on the lines leading into the RCC.

Distributed routing involves each node in the network periodically exchanging explicit routing information with its neighbours to construct a local table containing dynamic information about the network. Then each node can base routing decisions upon the current version of this table. No doubt, this will increase the communication traffic.

Our approach for an X-tree routing algorithm is a mixture of distributed and centralized routing. That is because in Jade there are no time_out facilities - each node cannot know whether its neighbour is active or not. The centralized controller (manager) is used to send information to affected nodes when the network topology has been changed.

Each node contains a local control process which attempts to find a shortest route to send or transfer messages, that is, each node local control process executes the same routing algorithm. This routing algorithm is (here we follow the terms defined in 3.1):

(1). if me=destination then OK, stop transferring.

(2). if me and destination are brothers, there are three cases:

a). if level distance from me to destination is longer than 4, as shown in Fig.4, then send message to parent.

b). if the distance from my left hand to destination is less than or equal to 4 then send message to my left brother.

c). else send to right brother.

(3). if the destination is my descendant, there are three cases:

a). if it belongs to my left subtree, then send message to my left child.

b). if it belongs to my right subtree, then send message to my right child.

c). if it does not belong to my subtree, then first find its ancestor who is my brother(see Fig.5), after that, go to step (2) to chose the shortest route from me to brother.

Besides the above basic rules, other rules to improve robustness have been added:

(1). if the shortest route is broken, i.e. the next node selected by the above shortest routing algorithm is not alive, then randomly select an active neighbour node (include the prior node that transferred the message to me), and send the message to the neighbour node.

(2). if the next node selected by the shortest routing algorithm is the same as the prior node (that means the shortest route was broken and the message was send back), then select an active neighbour node randomly provided that it is not the prior node. If there is no such neighbour node then send the message back to the prior node.

These two additional rules guarantee that whenever the shortest route breaks, the message will eventually be sent to the destination provided that there is a route (or routes) from the source to the destination. If the route from source to destination does not exist, i.e. the source node and destination node belongs to two isolated subnets respectively, the message will loop in the subnet which the source node belongs to, until the communication line between these two isolated subnets resumes.

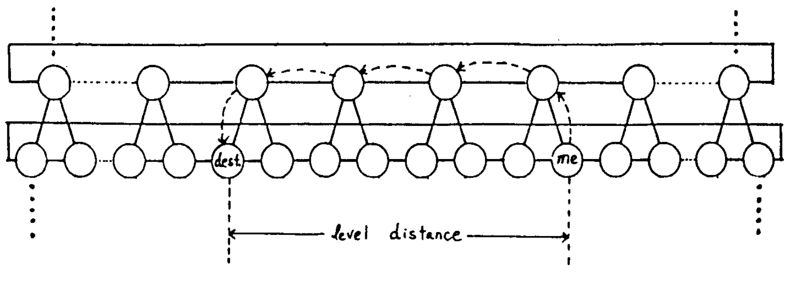

Fig.4. The level distance longer than 4

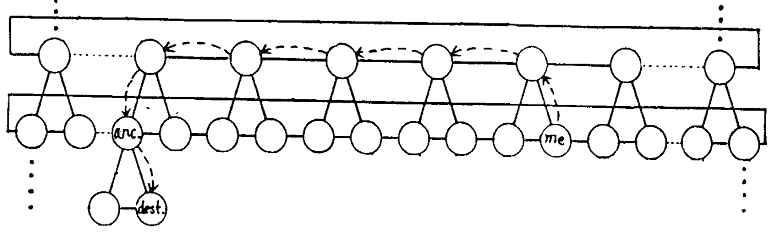

Fig.5. Destination is not my descendant

3.3. A Simulation of X-tree Communication

Simulation is a technique for representing a dynamic system by a model in order to gain insight into the operation of the underlying system. The distributed simulation program for X-tree consists of three parts: a *menu events process* , a *manager process* and several *node processes*, as shown in Fig.6.

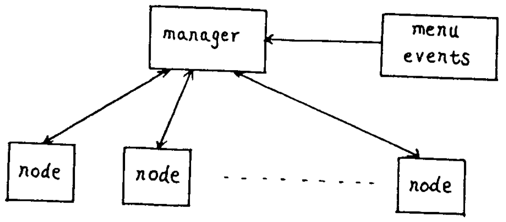

Fig.6. the program structure

The *menu events process* recognizes the event item selected by the mouse, such as kill a node, resume a node, send a message, send messages randomly, etc., and notifies the manager process.

The *manager process* is responsible for collecting simulation information, monitoring the change of the network, and animating X-tree topology on the screen. For example, when a node is killed, the manager process tells all that node's neighbours so they can change their connection tables. The icon of the killed node is then erased from the screen.

The *node processes* execute the same algorithm. They send or transfer messages, animate the communication routing on the screen, or modify their status and tables according to the commands from the manager.

3.4. X-tree Performance Results

To examine the behaviour of X-tree topology, dozens of simulation were performed. Fig.7 presents three typical statistics reports of the traffic load drawn from a 4-layer X-tree network when the nodes send messages randomly among each other. The term traffic load means the number of messages received by or passed through a node. We can now summarize X-tree performance in terms of the desirable properties defined in section 1.

(1). $\lim_{N \to \infty} \frac{2 * \log_2 (N+1) - 4}{N} = 0$;

(2). $\forall$ node $\in$ X-tree degree(node) ≤ 5;

(3). The shortest routing algorithm is simple and easy to implement;

(4). The connectivity of X-tree is much better than star, tree, ring topologies;

(5). The traffic loads of X-tree are not exactly uniform. Generally, the heavier traffic loads are located on the second last layer of X-tree. We can justify this statement by referring Fig.7.

node	first		second		third	
	load	weight	load	weight	load	weight
1	67	0.05	84	0.05	250	0.05
2	113	0.08	135	0.08	129	0.08
3	105	0.07	156	0.09	411	0.08
4	133	0.09	150	0.08	451	0.09
5	139	0.10	159	0.09	506	0.10
6	142	0.10	173	0.10	471	0.09
7	108	0.07	151	0.08	406	0.08
8	74	0.05	108	0.06	286	0.05
9	83	0.06	93	0.05	290	0.06
10	86	0.06	92	0.05	297	0.06
11	70	0.05	77	0.04	287	0.05
12	86	0.06	103	0.06	297	0.06
13	87	0.06	112	0.06	282	0.05
14	78	0.05	89	0.05	263	0.05
15	72	0.05	101	0.06	312	0.06
total	1113	100%	1783	100%	5061	100%

Fig.7. Traffic loads of 15 nodes

4. CONCLUSION

In this paper, we presented an overview of the Jade environment and a simulation of X-tree. Novel features of Jade are that it supports the modelling and simulation of target distributed systems including the animation of such simulations. The goals of Jade are to provide a cost effective software development environment that is both easy and comfortable to use, and which can produce more reliable, maintainable programs. Using the Jade environment, the X-tree network topology and one routing algorithm has been simulated. These preliminary simulation results suggest that this kind of interconnection topology may be suitable to build megamicro computers which consist of hundreds or thousands of identical small computers. For example, we can build some standard units, each unit consists of, say, fifteen transputers (computer plus communication mechanism), as Fig.8 below:

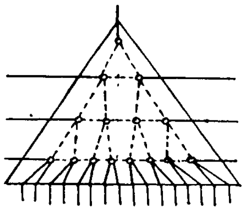

Fig.8. X-tree computer unit

Each of these units could be implemented in VLSI enabling a very large X-tree to be constructed. Possible configurations which could be constructed using these units are illustrated in Fig.9 and Fig.10.

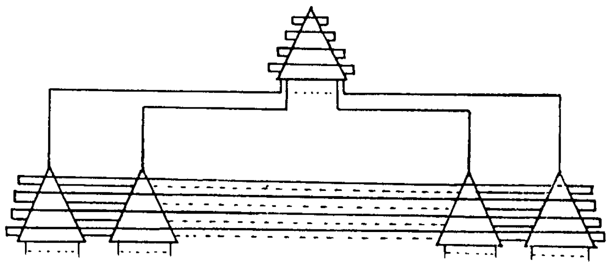

Fig.9. X-tree computer

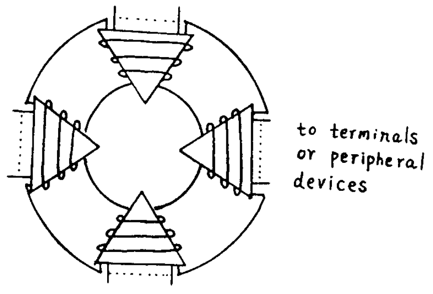

Fig.10. X-tree-ring computer

ACKNOWLEDGEMENTS

We would like to express our thanks to the Natural Sciences and Engineering Research Council of Canada for its support of Project Jade, and our Jade colleagues for their contributions.

REFERENCES

[1]. C. H. Sequin, A. M. Despain, and D. A. Patterson (1979) "Communication in X-Tree, A Modular Multiprocessor System " ACM 78 Proceeding, Washington, D.C.

[2]. Neal, R., Lomow, G.a., Peterson, M., Unger, B.W., and Witten, I.H. (1984) "Experience with an inter-process communication protocol in a distributed programming environment" CIPS Session 84 Conference, Calgary, Alberta

[3]. Cheriton, D.R., Malcolm, M.A., Melen, L.S. and Sager, G.R., (1979) "Thoth : a portable real_time operating system." CACM, 22 (2), 105-115, February

[4]. Unger, B.W., Birtwistle, G., Cleary, J., Hill, D., Lomow, G.A., Neal, R., Peterson, M., Witten, I.H., and Wyvill, B. (1984) "Jade : a simulation and software prototyping environment." Proc SCS Conference on Simulation in Strongly Typed Languages, San Diego, Colifornia, February

[5]. Bonham, M. and Witten, I.H. (1984) "Towards distributed document preparation with interactive and noninteractive Viewing." CIPS Session 84 Conference, Calgary, Alberta

[6]. Dewar, A. and Unger, B.W. (1984) "Graphical tracing and debugging of simulations." Proc SCS Conference on Simulation in Strongly Typed Languages. San Diego, Colifornia, February

[7]. Birtwistle, g., Wyvill, B., Levinson, D., and Neal, R. (1984) "Visualizing a simulation using animated pictures." Proc SCS Conference on Simulation in Strongly Typed Languages. San Diego, Colifornia, February