# A Performance Comparison of Tree and Ring Topologies in Distributed Systems

Min Huang
min@scl.ameslab.gov

Brett Bode
brett@scl.ameslab.gov

*Scalable Computing Laboratory, Ames Laboratory, Iowa State University, Ames IA 50011*

## Abstract

*A distributed system is a collection of computers that are connected via a communication network. Distributed systems have become commonplace due to the wide availability of low-cost, high performance computers and network devices. However, the management infrastructure often does not scale well when distributed systems get very large. The considerations in building a distributed system are the choice of the network topology and the method used to construct the distributed system so as to optimize the scalability and reliability of the system, lower the cost of linking nodes together and minimize the message delay in transmission, and simplify system resource management.*

*We have developed a new distributed management system that is able to handle the dynamic increase of system size, detect and recover the unexpected failure of system services, and manage system resources. The topologies used in the system are the tree-structured network and the ring-structured network.*

## 1. Introduction

A distributed system is a collection of computers that are connected via a communication network. Usually, in large-scale distributed systems, it is much more difficult to provide software that is fault-tolerant, reliable, manageable and easy to use than in small-scale distributed systems. In order to solve the problem of the lack of software for the effective management and utilization of computational resources, the U.S. Department of Energy has established Scalable Systems Software Center [15]. The goals of the center are to develop an integrated suite of machine independent, scalable systems software components need for the Scientific Discovery through Advanced Computing (SciDAC) [16] initiative and to provide open source solutions that work for both small and large-scale systems. Our work is part of the SciDAC project.

We have designed and developed a distributed management system. In this system, there are one master node and hundreds or thousands of slave nodes. The master node is the manager of the system. It is responsible for computing optimal methods to construct and recover the system. Each slave node is responsible for reporting its working status and resource usage, detecting and reporting its neighboring node's failure, and dynamically adjusting its position according the instruction from the master node. We have tested the construction and recovery performance, and the communication performance in the tree-structured networks and the ring-structured network.

The rest of this paper is organized as follows: The research background is presented in section 2. Section 3 describes the system components, including the different node types and different connection types used in the system. We discuss the system design and implementation in section 4. In section 5, we present the test environment and results. Finally, we conclude with a summary and describe our future work in section 6.

## 2. Research background

### 2.1. Distributed systems

A distributed system is a system consisting of computers that do not share a common memory or a synchronized clock. The computers in a distributed system are connected via a communications network. The computers can access remote resources as well as local resources in the distributed system. A computer accesses remote resources via the communication network. Generally, it is more expensive to access the remote resources than to access the local resources because of the communication delays and the CPU overhead to process communication protocols [1]. Figure 1 shows the

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

architecture of the distributed system.

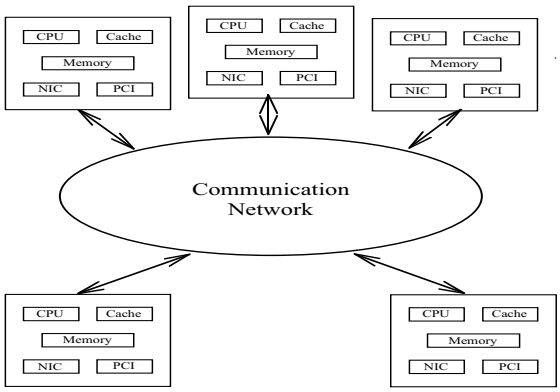


**Figure 1. Architecture of the distributed system**

The motivation behind the development of distributed systems is the availability of the low-cost, high performance computers and network devices. When a few powerful computers are connected and communicate with each other, the total computing power available can be enormous. Such a system can have a higher performance/price ration than a single supercomputer.

### 2.2. Communication networks in the distributed system

We can use fully connected networks or partially connected networks to construct a distributed system.

In a fully connected network, there are direct links between all pairs of computers. The problem with such a system is that adding new nodes to the system results in the increase of each node's degree, which results in opening more file descriptors and more complexity for each node to implement the connections. Thus the scalability of such systems is limited by each node's capacity to open file descriptors and the ability to handle the new connections.

In a partially connected network, direct links exist between some, but not all, pairs of computers. Some of the examples of partially connected networks are star-structured networks, multi-access bus networks, ring-structured networks, and tree-structured networks.

Some of the traditional distributed systems use a star with temporary connections as their network topology. The problems with such a system are that the central node becomes the bottleneck and the establishment and termination of the connections are a significant overhead of the system.

In a multi-access bus network, all the nodes in the system are connected to a single shared bus link. The bus link becomes the system bottleneck.

The topologies used in our system are the tree-structured network and the ring-structured network. We tested the construction and recovery performance, and communication performance in the system.

### 2.3. Definitions

*Construction event:* The construction event happens when a new node wants to join the system. The new node sends registration request to the master node and the master node computes the position for the new node.

*Recovery event*: The recovery event happens when a slave node fails. The neighboring nodes of the failed node report the failure to the master node. The master node computes a method to recovery the system structure.

### 2.4. Research motivation

Our research focuses on the design and implementation of a distributed system that can handle the dynamic increase of the system size and can detect and recover a system failure automatically, simplify the management of the system, and lower the communication delay and system overhead. Our research aims to find an optimal method to construct and recovery the tree-structured and the ring-structured network and to minimize the data transmission latency and network traffic.

## 3. System structure and components

### 3.1. The System Structure

As stated before, our work exams the construction and recovery performance, and the communication performance in the tree-structured network and the ring-structured network.

*Terminologies*

*Complete n-ary tree:* A complete n-ary tree is a tree in which all the nodes have at most *n* child nodes and all the levels are full except for the bottom level and the bottom level is filled from left to right [4]. We call the node with the maximum ID in the tree *the last node*. Figure 2 shows an example of a complete ternary (n=3) tree.

*Ring:* In a ring, all nodes are connected to one another in the shape of a closed loop, so each node is connected directly to two other nodes, one on either side of it [17].

In both the tree-structured network and the ring-structured network, there are 2 types of nodes in the

system, one master node and many slave nodes. In the tree-structured network, the master node of the system is the root of the tree. In the ring-structured network, the master node of the system is the head of the ring.

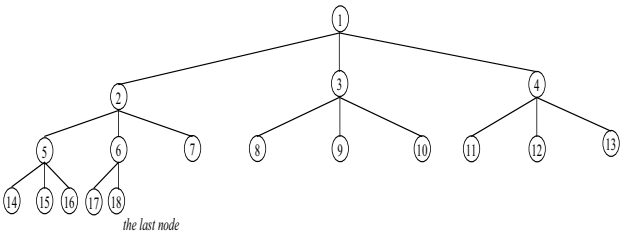


**Figure 2: An example of a complete ternary tree**

### 3.2. Node Responsibilities in the System

To provide reliable services to the end users, both the master node and slave nodes are responsible for resource management and system structure maintenance.

To monitor and manage the system status and the system resources, the master node gathers the working status and resource usage information of all slave nodes and sends instructional messages to slave nodes periodically. Each slave node is responsible for generating messages to report its working status and resource usage. These messages are called the resource management messages. Each slave node is also responsible for forwarding the messages from its parent node to its child nodes and merging the resource management messages from its child nodes and forwarding the merged message to its parent node.

To maintain the system structure, the master node has data storage to store the system structure and it is responsible for keeping the data storage up-to-date. It is also responsible for computing optimal methods to construct and recover the distributed system and give slave nodes instructions to maintain the given system structure. Each slave node is responsible for its registration to be added to the system, detecting and reporting it's neighboring node's failure to the master node, and dynamically adjusting its position in the system according to the instructions from the master node.

### 3.3. Connections in the System

To manage the system resources and maintain the system structure, there are two types of messages transmitted along the network links, *the resource management messages* and the *system structure maintenance messages*. Since both of these messages require reliable delivery services, TCP is used as the transport protocol [5]. To transmit these two kinds of messages, there are two types of connections in the system: *permanent connections* and *temporary connections.*

The permanent connections are used to transmit the resource management messages. All nodes in the system are connected to their neighboring nodes by permanent connections. The reason we use the permanent connections to transmit the resource management messages is the overhead of TCP three-way handshake and four-way termination. The resource management messages are periodically exchanged between the neighboring nodes. Once a connection is established between the neighboring nodes, it is persistent. There is no need to establish a new connection for each resource management message. Using permanent connections to transmit the resource management messages decreases the overhead of the establishing and the closing of the network connection.

The temporary connections are used to maintain the system structure. For both the construction and recovery event, there is a group of system structure maintenance messages transmitted between the master node and a slave node. A temporary connection is for transmitting a given group of messages. After the transmission of the given group of messages, it is not necessary for the temporary connection to exist. Using temporary connections to transfer the system structure maintenance messages can release the load burden of the master node.

## 4. System design and implementation

### 4.1. System Design

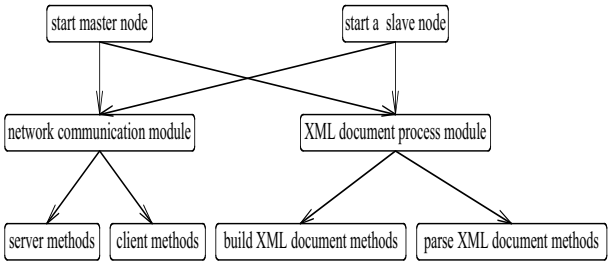


**Figure 3. Software modules**

As discussed in section 3, the system consists of a master node and up to hundreds of slave nodes. Nodes in the system have to communicate with others to accomplish the system goals. The messages transmitted in

the system are in XML format. Network communication and XML document processing modules are needed for the master node and slave nodes to accomplish their goals. The network communication module and XML document processing module were abstracted out and shared by both the master node and slave nodes. Figure 3 shows the software modules and their relationships in this system.

## 4.2. System Implementation

### 4.2.1. The XML Document Processing Module

XML stands for e**X**tensible **M**arkup **L**anguage. XML documents are used in this system as the data storage in the tree-structured network and message formats transmitted in the system. The reason why we use XML documents in our work is that the markup tags in XML are used to describe and store data, and allow the application to store structured data in XML documents and extract data from XML documents [9].

The XML document processing module provides all the methods needed by the master node and slave nodes to process the XML messages. The Xerces-C++ parser is used in our system as the XML parser [14]. There are 2 C++ classes in the XML document processing module, the *BuildMsg* class and the *ParseMsg* class. The *BuildMsg* class provides methods to create and modify XML documents. The *ParseMsg* class provides methods to extract information from XML documents.

### 4.2.2. The Construction and Recovery of the Tree-Structured Network

In the tree-structured network, the network is constructed as a complete *n-ary* tree. Each node in the system has a position ID that is determined by the browse order by breadth first search while the root of the tree has position ID *1*.

*Construction*: When the system starts, there is only the master node that is the root of the tree. New nodes are added to the system dynamically in the order of their registration requests. The master node is the manager of the system. When it starts, the master node initializes the system structure database, and opens a port to listen for requests from slave nodes.

The following illustrates the steps involved in constructing the tree-structured network.

*The new node sends a registration request to the master node. In the registration request, the new node sends its network-based information.*

*The master node queries the structure database, and computes the parent node for the new node.*

*The master node sends a response message that contains the parent information to the requesting node.*

*The new node tries to connect to the parent node specified in the response from the master node.*

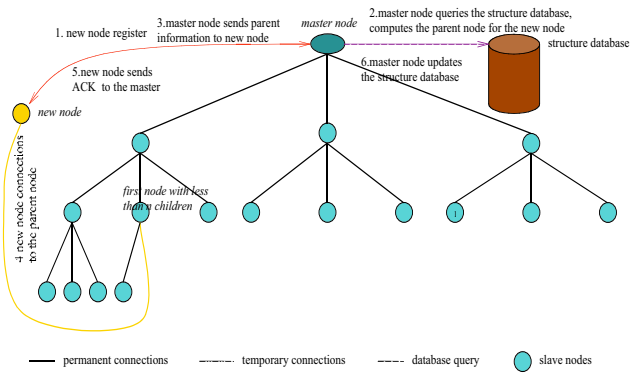


**Figure 4. The process of adding a new node to the system**

*If the new node successfully connects to its parent node, it sends an acknowledgement message to the master node.*

*After receiving the acknowledgement message from the new node, the master node adds the new node to the system structure database.*

*If the new node cannot connect to the assigned parent node, the master node will not add the new node to the system database. The new node will send another registration request to the master node.*

*Recovery:* When a node in the tree-structured network system fails, all its neighbors will report the failure to the master node. To minimize the number of nodes involved in the recovery event , when the master node receives the first report message of the event, it chooses the node with the maximum ID that is alive to replace the position of the failure node (see section3). After the parent node reports the failure, it will close the connection to the master node while after a child node reported the failure, it will expect the new parent information from the master node. The process of recovering a tree-structured network can be illustrated as figure 5.

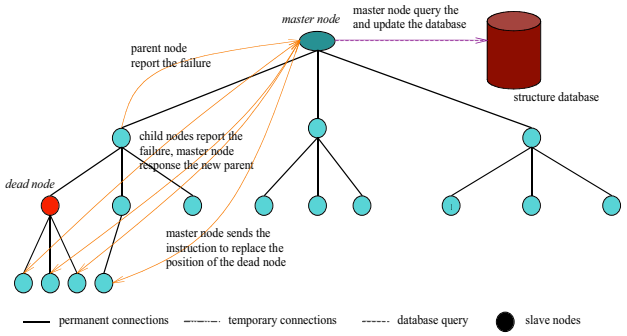


**Figure 5. The process to recover the system**

The reason why all the neighbors instead of only a single neighbor will report this failure can be illustrated as

follows:

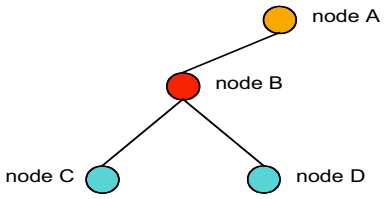


**Figure 6. A segment of a tree when node B fails**

*The neighbors of the failed node do not know each other. There is no connection between any pair of the failed node's neighbors, when a node reports this failure, it cannot notify other nodes. In figure 6, there are no connections between node A and node C, node A and node D, and node C and node D. When node A reports the failure of node B, it cannot notify node C and node D.*

*It is possible that two adjacent nodes fail simultaneously, so one node cannot rely on others to report the failure. In figure 6, if node A and node B fail at the same time, and node C and node D rely on node A to report the failure node B, node B's failure will not be reported to the master node.*

*All the child nodes of the failed node need to get instructions from the master. It is reasonable for a node to get instructions from the master node after it reports the failure. In figure 6, node C and node D get new parent information after they report the failure to the master node.*

### 4.2.3. The Construction and Recovery of the Ring-Structured Network

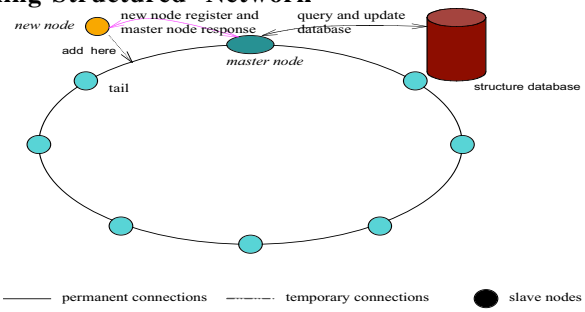


**Figure 7. The construction of the ring-structured network**

*Construction:* As in the tree-structured network, when the system starts, there is only the master node that is the head of the ring. New nodes are added to the system dynamically according to the order of their registration requests. The steps involved in a construction event are the same as in the tree-structured network. In the ring-structured network, the new node is always added as the tail of the ring. Figure 7 shows the construction of the ring-structured network

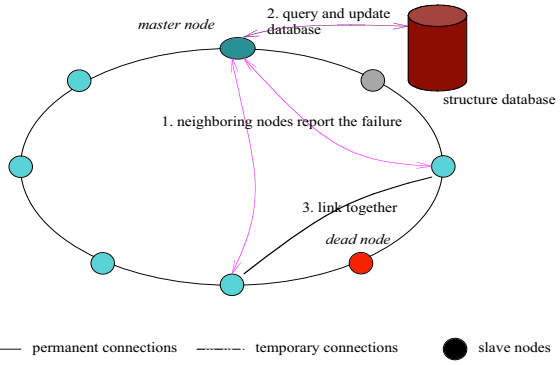


**Figure 8. The recovery of the ring-structured network**

*Recovery:* During the execution of the system, when a node in the system fails, both its previous node and next node will report this failure to the master node. Since it is possible that during the recovery process the failed node's previous node and next node may fail, the master node will find two nearest neighbors of the failed node that are alive and link them together. Figure 8 shows the recovery of the ring-structured network.

### 4.2.4. The Program Design of the Master Node

The master node is a concurrent server that can accomplish multiple tasks simultaneously. The concurrence of the master node is implemented using *POSIX Threads* [11]. There are four types of threads in the master node: the main thread, the resource management thread, the system structure management thread, and the connection handling threads.

*Main thread*: It is the entry point of the program. It is responsible for creating working threads to handle the different functions of the master node.

*Resource management thread*: It is used to accomplish the resource management tasks of the master node and sends the instructional messages to its child nodes.

*System structure management thread:* It is the most important part in the system. It is responsible for computing an optimal method to construct and recover the system so that the system has good scalability and reliability.

Since each node in the system may have more than one neighbor, in case of one node failure, all of its neighbors need to report the failure to the master node. In the process of handling a recovery event, it is possible for new requests, including new registration requests and new failure report requests, to be received. To clearly describe these situations, we define the following terminologies.

*Normal State:* The system is running normally.

*Inprocess State:* The master node is handling a recovery event.

*Process Time:* The beginning time when the master node starts to handle a new recovery event.

*Maximum Delay:* The longest time that the system can be in "Inproess State".

*Timeout State:* The system stays in the Inprocess State longer than the Maximum Delay

For example, here is a description of the system state transformation from the normal state. In the normal state, the master node is expecting both the registration and report request. After handling the registration request, the system stays in the normal state. If the incoming request is a report request and the failed node only has one neighboring node, after handling this request, the system stays in the normal state. If the incoming request is a report request and the failed node has more than one neighboring node, after handling this request, the system changes to the inprocess sate. In the inprocess state, the master node only accepts report requests that report the current failure. If the incoming report request is the last request to report the current failure, after handling this request, the system changes back to the normal state. If the incoming request is not the last report request to report the current failure, the system stays in the inprocess state. If the system stays in the inprocess state longer than the maximum delay and not all the neighbors of the current failed node report the current failure, the system will change to the timeout state. In the time out state, the master node will compute a method to wrap up the current recovery event.

The steps involved in wrapping up the current recovery event are as follows:

*The master node detects if the neighboring node that should have but has not reported the current failure is still active.*

*If it is active, the master node will send an instructional message to tell it to connect the new parent node.*

*Otherwise, a node in the system will be chosen to replace the position of the unreported node. In the tree-structured network, the node with the maximum position ID that is still active will be chosen while in the ring-structured network, the nearest neighbor node that is still active will be chosen to replace the unreported node.*

*Connection handling thread:* The main thread creates a connection handling thread for each directly connected child node. It is responsible for handling the communication with the child node, detecting the status of the child nodes. Since the resource management messages are transmitted along the permanent TCP connections periodically, while the peer is down, the sender will get a failure signal while it is trying to send message to the receiver.

#### 4.2.5. The Program Design of the Slave Node

Similar to the master node, the slave nodes are concurrent servers. The concurrence of the slave node is implemented using POSIX Threads [11]. There are five kinds of threads in the slave node: the main thread, the resource management thread, the client thread, connection handling threads, and the message merge handling thread.

*Main thread:* The main thread is the entry point of the program. It is responsible for creating working threads to handle the different tasks of the slave node

*Resource management thread:* It is responsible for generating and reporting its working status and the resource usage information.

*Client thread:* It is used to handle the communication with the parent node. It is responsible for detecting and reporting the failure of the parent node, and connecting to the new parent node according the instructions from the master node.

*Connection-handling thread:* It is used to handle the communication with the child node. It is responsible for detecting and reporting the failure of the child node.

*Message-merging thread:* The slave node uses this thread to merge the resource management messages from all the child nodes and its own resource management message

## 5. Test environment and results

### 5.1. The Test Environment

Our tests were conducted on the PowerPC G4 cluster in the Scalable Computing Laboratory, Ames Laboratory of U.S. Department of Energy. The G4 Cluster is a 32 node "Beowulf" style cluster computer consisting of 16 single processor G4s with 512 MB RAM and 16 dual processor G4s with 1GB RAM, all running Debian Linux. They use Ethernet and Myrinet for network access.

The master node is the manager of the system. It requires more system resources than a slave node. To test the construction and the recovery performance of the system, each slave node starts at a different time and runs for a random time period. This is implemented in a script file that is submitted to the batch system.

### 5.2. Test Results

We tested two aspects of system performance. First, we tested the time used for a new node to be added to the system and the time used to recover to the given structure in case a node fails in the system. Second, we tested the Round Trip Time (RTT) for messages transmitted in the system. The time unit used in the following results is milliseconds.

Figure 9 shows the average time used to add a new node to the system with different network topologies and different system sizes. We can see that there is no significant difference in the time used to add a node to system. The reason is that the steps involved in adding a new node to the system are fixed. The system structure management thread of the master node is an iterative server; it handles the construction and recovery event in a sequential way. Only after it finishes handling a registration request, will it handle a new registration or report request.

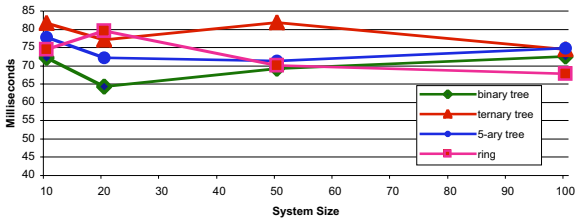


**Figure 9. The construction performance**

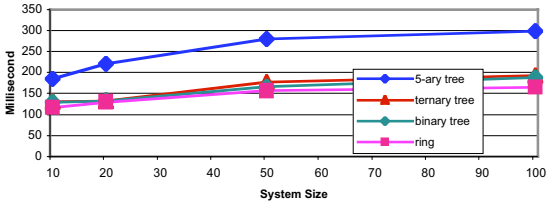


**Figure 10. The recovery performance**

Figure 10 shows that the average time used to recover the system when there is a node failure in the system. We can see that the time used for a recovery event is related to the network topologies. The time used grows with the degree of the node. The reason is that when a node fails, all its neighbors have to report this failure to the master node. The more neighbors one node has, the more report requests the master node has to process. The time used to process a recovery event in a 5-ary tree-structured network is much longer than that used in a ring-structured network. In the tree-structured network, the time used also grows with the number of slave nodes in the system. When a node fails in a tree-structured network, the master node has to find the active node with the maximum position ID to replace the failed node. The more nodes one system has the more complex it is to find a node to replace the failed node.

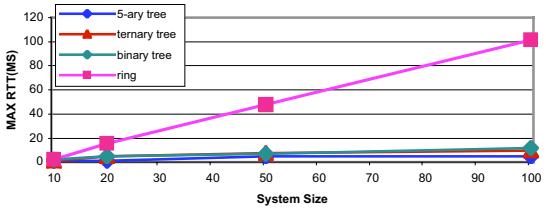


**Figure 11. The maximum RTT with zero payload**

Figure 11 and Figure 12 show the maximum RTT for zero payload and XML payload messages (252 bytes) transmitted with different network topologies and different system sizes. From these figures we can see, the RTT for XML payload (252 bytes) messages is much higher than that of zero payload messages. This is because when receiving an XML message, each node has to process the XML message and processing an XML message is a time consuming task. Another point we can see from these figures is that the maximum RTT in a ring-structured

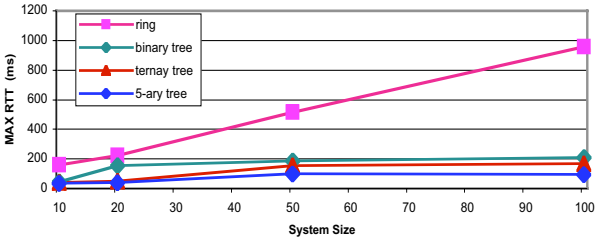


**Figure 12. The maximum RTT with XML payload**

network grows significantly with the system size. The reason is that the diameter of the system grows linearly with the system size. The diameter of an n-ary tree with m nodes is $\log_n m(n \quad 1) + 1 \quad 1$. Thus in an n-ary tree-structured network, the maximum RTT grows much more slowly than in a ring-structured network. In our tests, when the system has 100 nodes, the longest RTT for XML payload (252 bytes) messages in the 5-ary complete tree is 10% of that in the ring-structured network. The growth rate decreases as the degree of the tree increases.

## 6. Conclusion

### 6.1. Conclusions

This paper compared the network topologies used to construct distributed systems, and presented test results for systems using tree-structured networks and ring-structured networks as the network topologies. From the discussions in the previous sections and the test results in section 5, we draw the following conclusions:

It is easier to maintain the system structure in a ring-structured network. However, the longest RTT grows linearly with the system size. For large systems this can be a significant limitation and thus limits the overall scalability of the system. Ring-structured networks are suitable for small to medium sized systems with small messages.

The scalability of a tree-structured network system is related to the node's capacity and the height of the tree. We can carefully choose an appropriate degree n to construct a complete n-ary tree-structured network such that the height of the tree balances the growth in the tree size versus the resource requirements for a node to communicate with additional child nodes. Tree-structured network is superior to the ring-structured network when the system size and the messages transmitted in the system have large payloads.

### 6.2. Future Work

In our system, there is only one master node. It is responsible for managing the system structure. If the master node fails, all the information about the system structure will be lost and there will be no node left to manage the system. One of our future tasks is to construct a backup master node that will backup the system structure information continuously. In case that the master node fails, the backup master node can be used to assume control and manage the system structure.

.

## 7. Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy under contract W-7405-Eng-82 at Ames Laboratory operated by the Iowa State University of Science and Technology. Funding was provided by the Mathematical, Information and Computational Science division of the Office of Advanced Scientific Computing Research.

## 8. References

[1] Mukesh Singhal and Niranjan G. Shivaratri, "Advanced Concepts in Operating Systems", McGraw-Hill, 1994.

[2] Abraham Silberschatz and Peter Baer Galvin, "Operating System Concepts", Fifth Edition, John Wiley & Sons, 1999.

[3] Andrew Warfield, Yvonne Coady, and Norm Hutchinson, "Identifying Open Problems in Distributed System", Proceedings of European Research Seminar on Advances in Distributed Systems (ERSADS), 2001.

[4] Bruno R. Preiss, "Data Structures and Algorithms with Object-Oriented Design Patterns in C++", Wiley, 1998.
URL:http://www.brpreiss.com/books/opus4/html/page356.html (date accessed: November 15, 2004).

[5] W. Richard Stevens, "UNIX Network Programming", Volume 1, Second Edition, Prentice Hall, 1998.

[6] Mark Birbeck, Jon Duckett, Oli Gauti Gudmundsson, Pete Kobak, Evan Lenz, Steve Livingstone, Daniel Marcus, Stephen Mohr, Jonathan Pinnock, Keith Visco, Andrew Watt, Kevin Williams, Zoran Zaev, and Nikola Ozu, "Professional XML", 2nd Edition, Wrox Press, 2001.

[7] XML Tutorial,
URL:http://www.w3schools.com/xml/default.asp
(date accessed: November 15, 2004).

[8] Jerry Emerick, "Managing XML Data Storage", ACM Crossroads archive, Volume 8, Issue 4, Pages: 6 – 11, 2002.

[9] Ronald Bourret, "XML and Databases",
URL:
http://www.rpbourret.com/xml/XMLAndDatabases.htm
(date accessed: November 15, 2004).

[10] Douglas E. Comer and David L. Stevens, "Internetworking with TCP/IP", volume III, Prentice Hall, 1996.

[11] David R. Butenhof, "Programming with POSIX Threads", Addison Wesley, 1997.

[12] Mark G. Sobell, "A Practical Guide to Red Hat Linux 8", Addison Wesley, 2003.

[13] Herbert Schildt, "C: The Complete Reference", Fourth Edition, McGraw-Hill, 2000.

[14] The Apache XML Project, "Xerces C++ Parser",
URL: http://xml.apache.org/xerces-c
(date accessed: November 15, 2004).

[15] Scalable Systems Software for Terascale Computer Centers,
URL: http://www.scidac.org/ScalableSystems
(date accessed: November 15, 2004).

[16] The Project of Scientific Discovery through Advanced Computing,
URL: http://www.scidac.org
(date accessed: November 15, 2004).

[17] Network Topologies,
URL:http://www.webopedia.com/quick_ref/topologies.asp (date accessed: January 19, 2005).