A performance comparison of tree and ring topologies in distributed system

by

Min Huang

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Ricky Kendall, Co-major Professor
Ying Cai, Co-major Professor
Brett Bode

Iowa State University

Ames, Iowa

2004

Graduate College
Iowa State University

This is to certify that the master's thesis of

Min Huang

has met the thesis requirements of Iowa State University

Co-major Professor

Co-major Professor

For the Major Program

TABLE OF CONTENTS

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.

Some of 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.

This thesis presents the research background, system components, design, implementation, experiment results and the conclusions of our work. The thesis is organized as follows: the research background is presented in chapter 1. Chapter 2 describes the system components, including the different node types and different connection types used in the system. In chapter 3, we describe the message types and message formats in the system. We discuss the system design and implementation in chapter 4. In chapter 5, we present the test environment and results. Finally, we conclude with a summary and describe our future work in chapter 6.

1. RESEARCH BACKGROUND

1.1 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. Each computer has its own memory and runs its own operating system. 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 1-1 shows the architecture of the distributed system.

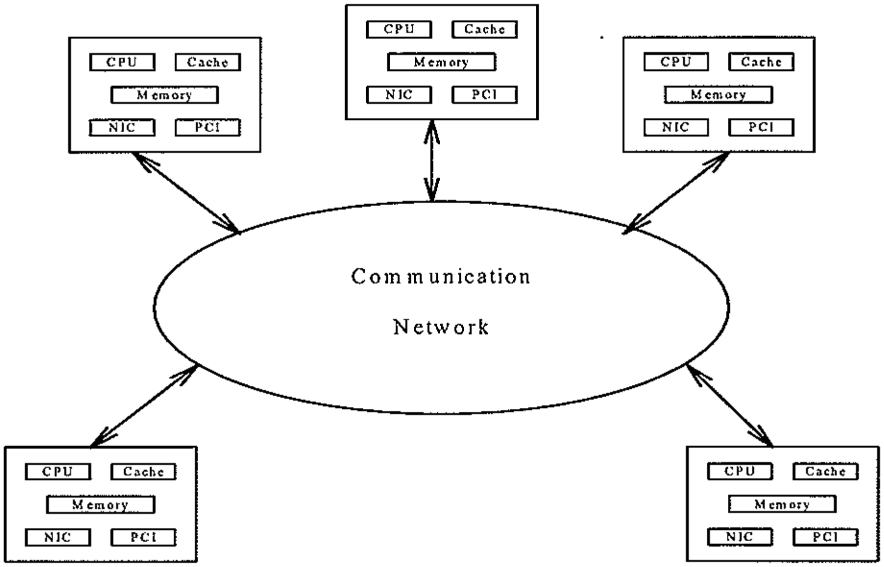

Figure 1-1 Architecture of the distributed system

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 generally costs tens of thousands of dollars. On the other hand, the cost of a single supercomputer with the same computing power can be as high as a few million dollars.

1.2 The Communication Networks in Distributed Systems

In a distributed system, all the computers exchange messages with other computers and access resources in other computers through the communication network. In a computer network, two nodes are *neighbors* if there is a link connecting them. The *degree* of a node is the number of its neighbors. When a computer sends messages to another computer in the system, messages may have to go through intermediate computers to reach their final destinations. The *diameter* of a network is the longest path between any two computers. The computers in a system can be connected physically in variety topologies. Generally, when choosing a topology to build a distributed system, we need to limit the nodes' maximum degree and the diameter of the system. Since when a node has more neighbors, it needs more file descriptors for the connections and it is more complex for it to handle the concurrent communications with different neighbors, and the larger diameter results in the longer message transmission delay. In this section, we describe the most commonly used network topologies in distributed systems and compare them using the following criteria:

- **Basic cost:** the expense to link the computers to build the distributed system [2].
- **Scalability:** the ability to scale the system size without increasing the requirement of each node's capacity and without resulting in significantly longer message transmission delays.
- **Communication cost:** the message delay of sending a message from the source to the destination [2].
- **Reliability:** when one or several computers or links in the system fail, the ability for the remaining computers to communicate with each other [2].

1.2.1 Fully Connected Networks

In a fully connected network, there are direct links between all pairs of computers. When a new computer is to be added to the system, a link has to be added to each existing node. Thus, the basic cost is high and grows as the square of the number of computers. Additionally, 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. However, in this environment, the diameter of the system is fixed to *1*. The communication cost is low because a message sent from one computer to another one only goes through one link. This kind of systems is reliable because when a few computers or links fail, the rest of the computers can still communicate with others. Figure 1-2 shows an example of fully connected networks.

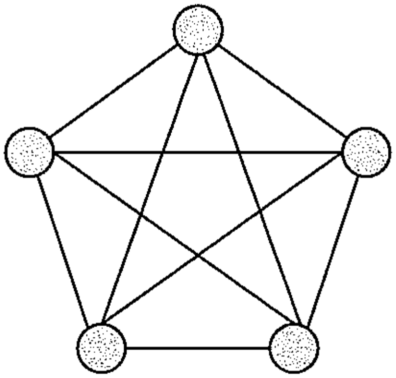

Figure 1-2 Fully connected networks

1.2.2 Partially Connected Networks

In a partially connected network, direct links exist between some, but not all, pairs of computers. The basic cost is lower than that of the fully connected networks and is determined by the topology used. The scalability is not associated so tightly with each node's ability to open new file descriptors and to handle new connections as in a fully connected network, since the new node can be added to be the neighbor of nodes with smaller degrees. However, the diameter of the system needs to be considered when adding new node to the system. The communication cost is higher than that of the fully connected networks, because the message sent from one computer to another computer may go through several intermediate nodes, resulting in a longer message delay. The reliability of a partially connected network is not as good as that of a fully connected network because the failure of one node or link may result in the inability of the rest of the system to communicate. Figure 1-3 shows an example of partially connected networks.

Some of the examples of partially connected networks are *star-structured networks*, *tree-structured networks*, *ring-structured networks*, and *multi-access bus networks*.

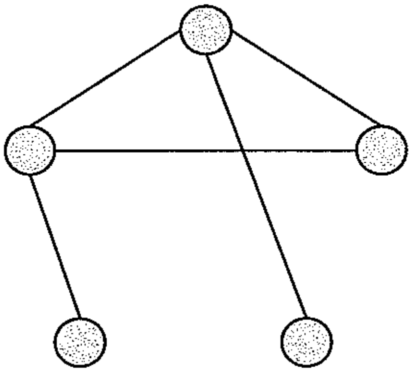

Figure 1-3 Partially connected networks

In a *star-structured network*, one of the nodes (the central node) in the system is connected to all other nodes. None of the other nodes are connected to any other. In such systems, when a new node is added to the system, one link is needed to connect the new node and the central node. The basic cost grows linearly as the number of the nodes in the system and the system scalability is limited by the central node's file descriptors and resources. Even though the diameter of the network remains fixed to *2*, all communications between the non-central nodes must go through the central node, this makes the central node becomes a bottleneck and results in a longer message delay. If the central node fails, none of the other nodes can communicate with each other. Some traditional distributed systems use star as the system topology with the temporary connections. The establishment and termination of the connections are a significant overhead of the system. Figure 1-4 shows an example of a star-structured network.

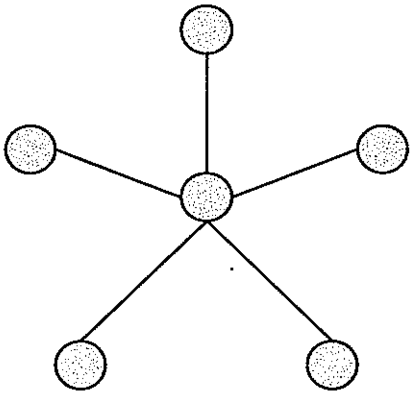

Figure 1-4 Star-structured networks

In a *tree-structured network,* the nodes are connected as a tree. A tree has three different types of node, a *root* node, *interior* nodes, and *leaf* nodes, each with different degrees. In these systems, the basic cost grows linearly with the number of nodes in the tree. The scalability of the tree-structured network is better than that of the fully connected network, since new node can be added as the child node of the leaf nodes or the interior nodes with fewer connections while limiting the height (the diameter) of the tree. However, in such systems, only messages transmitted between a parent node and its child node go though one link, other messages transmitted between two nodes have to go through one or more intermediate nodes. The worst case happens when the two nodes are both on the bottom level of the tree and the root is the only common ancestor of them. In this case, a message from the source has to go upwards until it reaches the root and go downward to reach the destination. If a parent node fails, its children nodes cannot communicate with other nodes in the system. Figure 1-5 shows an example tree-structured network.

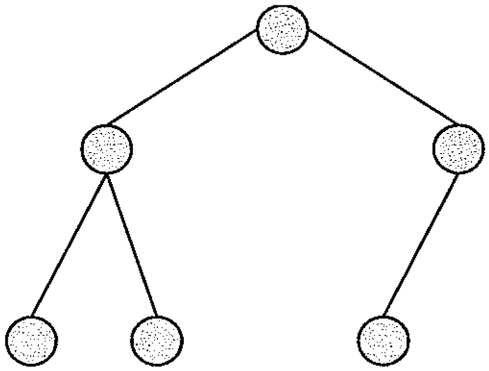

Figure 1-5 Tree-structured networks

In a *ring-structured network,* nodes are connected as a ring. In these systems, the basic cost grows linearly as the number of nodes in the system. Even though the degree of each node remains fixed to 2 as new nodes are added, the diameter of the system grows as the number of nodes in the system, resulting in a longer message transmission delay. The ring can be unidirectional or bidirectional. In the unidirectional architecture, a node can transmit a message to only one of its neighbors. All nodes must send the message in the same direction. In the bidirectional architecture, a node can transmit messages to both of its neighbors. Obviously, the bidirectional ring is more reliable than the unidirectional ring. Figure 1-6 shows an example of ring-structured network.

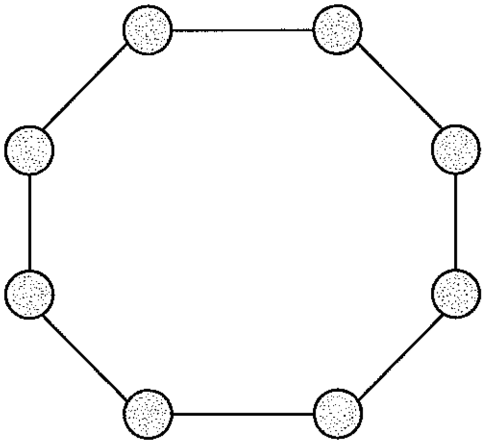

Figure 1-6 Ring-structured networks

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 bottleneck and if it fails, all the nodes in the system cannot connect to each other. Figure 1-7 shows an example of a multi-access bus network.

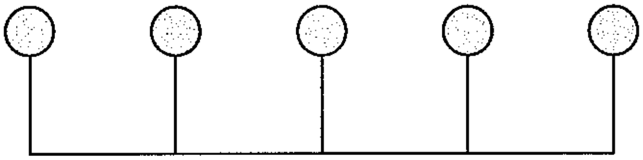

Figure 1-7 Multi-access bus network

1.3 The Advantages of a Distributed System

High Performance/Price Ratio

As stated before, the performance of personal computers and network devices are increasing rapidly while the prices are falling continuously. When a few personal computers are interconnected and communicate with each other, the total computing ability can be as powerful as a single supercomputer. However, the costs of these systems are much lower than those of supercomputers. Hence the main advantage of distributed systems is that they have a higher performance/price advantage over single supercomputers.

Resource Sharing

In the distributed systems, a computer can access a service from another computer by sending a request to it over the communications network, so hardware and software resources can be shared among computers in the distributed system. In general, resource sharing in a distributed system provides mechanisms for sharing files at remote sites, processing information in a distributed database, printing files at remote sites, using remote specialized hardware devices, and performing other operations [1].

Improved Reliability and Availability

In a distributed system, the failure of a few components of the system may not affect the availability of the rest of the system. This gives improved reliability and availability to the system [1].

Modular Expandability

When the existing system cannot meet the requirement of the end user, new hardware and software resources can be easily added to a distributed system without replacing the existing resources [1].

Enhanced Performance

Because many computing tasks can be concurrently executed using different computers and be balanced among the computers in distributed systems, a distributed system is capable of providing rapid response time and higher system throughput [1].

1.4 Considerations in Building Distributed Systems

There are several considerations in building a distributed system. They are listed in this section.

Scalability and Reliability

A system is scalable if it is easy to add new resources to it and provide services to additional users. A system is reliable if it is easy to detect system failures and fix the failures without or with a minimum of human interference. When designing and building a distributed system, the designer should choose a topology that is easy to scale to meet the increasing requirements from the end users. The system must also scale the size of the system and compensate for the failure of some components without losing the performance of the existing system.

Resource Management

In order to provide reliable services, we need to expect the same reliable service from the end nodes and the connective infrastructure in the distributed environment. It is necessary for the manager to monitor the resource usage of each node in the system and adjust the computing task distribution according to the workloads and resource usage of each computational node [3].

Data Exchange Cost

Another consideration with distributed systems is the cost of data exchange. Since the network performance has not improved as fast as the processor performance, as the number of computers in the system increases, the data exchange cost is not negligible when compared to calculations. When the amount of data transfer in the system becomes large, the data transfer cost becomes very large [3].

1.5 Research Motivation

Usually, in large-scale systems, it is much more difficult to provide software that is fault-tolerant, reliable, manageable and easy to use for systems administrators and users than in small-scale 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 System Software Center. 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. The Scalable Systems Software Center will produce an integrated suite of systems software and tools for the effective management and utilization of terascale computational resources particularly those at the DOE facilities [15]. Our research is part of the SciDAC project.

The first motivation of our research is to find an optimal method to build a distributed system so that the infrastructure scales well when distributed memory systems get very large and the distributed system is robust so that in case a node fails in the system, the system is easy to recovery.

The second motivation of our research is to find a way to simplify the management of the system. In order for the manager to monitor the whole system, the master node needs to gather the working status and resource usage from the computing nodes in the system and send instructional messages to the computing nodes to balance the whole system workload.

The third motivation of our research is to find a strategy to exchange information efficiently so that we can minimize the data transmission latency and lower the network transmission load. In the case when the number of nodes in the system becomes very large, we need to minimize the possibility that some nodes might become a bottleneck in the system.

Our research focuses on design and implementation of a network system for building a distributed system that can handle the dynamic increase of the system size and can detect and recover a system failure automatically. The network topologies used in our research are tree-structured networks and ring-structured networks. Our research aims to find an optimal method to build and recovery the tree-structured and the ring-structured network and to minimize the data transmission latency and network traffic.

1.6 Summary

The distributed system is composed of computers that are connected via communication networks. Because of the availability of the low-cost, high performance computers and network devices, distributed systems are widely used in many areas. Distributed systems have the advantages of the high performance/price ratio, resource sharing, improved reliability and availability, modular expandability, and enhanced performance. However, when constructing a distributed system, we need to consider the scalability and reliability, the resource management, and the data exchange cost of the system. The motivation of our research is to find a network topology and an optimal method to construct a distributed system so that the system can handle the dynamic increase of the system size and can detect and recover the system failure.

2. THE SYSTEM STRUCTURE AND COMPONENTS

2.1 The System Structure

2.1.1 Preliminary

Complete n-ary tree: Informally, a complete n-ary tree is a tree in which all the nodes have at most *n* children nodes and all the levels are full except for the bottom level and the bottom level is filled from left to right [4].

If we assign each node in the complete n-ary tree an ID with the order by breadth-first-search while the root of the tree has ID 1, then for a node with ID *i*, its parent node is the node with ID$\lceil (i-1)/n \rceil$, its child nodes are the nodes with IDs:

$n\ (i-1)+2,\ n\ (i-1)+3,\ n\ (i-1)+4,\ \ldots\ ni+1.$

We call the node with the maximum ID in the tree the *last node*. Figure 2-1 shows an example of a complete ternary (*n*=3) tree.

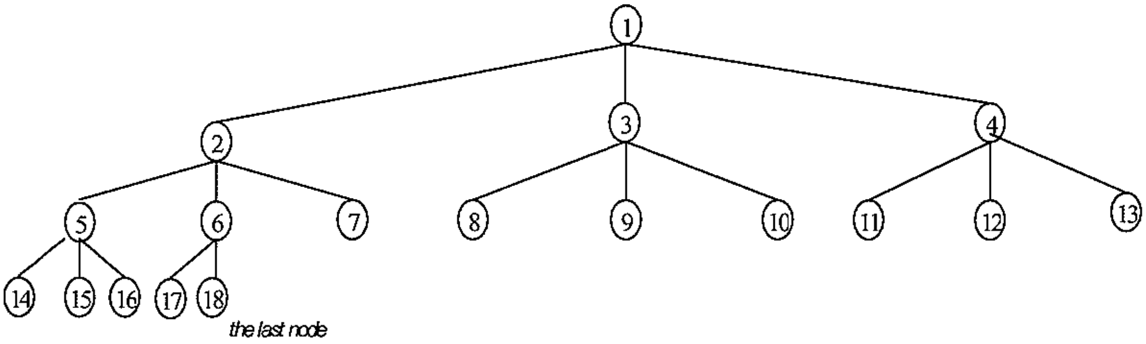

Figure 2-1 An example of a complete ternary (*n*=3) tree

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.

2.1.2 The Master Node and the Slave Nodes

Our work has examined the construction and the recovery of the tree-structured and the ring-structured network.

In both the tree-structured and the ring-structured networks, there are two types of nodes in the system: *the master node* and *the slave nodes*. Each system has only one master node and can have up to hundreds of slave nodes. In the tree-structured network, the master node is the root of the tree while in the ring-structured network the master node is the head of the ring. Other nodes are the slave nodes. There are two major responsibilities for both the master node and slave nodes. One is to maintain the given system structure. The other is to manage the system resources.

To simplify the management of the system, each node in the system is assigned a *position ID* that is determined by its position in the system. In the tree-structured network, the position ID of a node is the browsing order of the node using breadth-first-search where the root of the tree has position ID 1. In the ring-structured network, the position ID of the node is the position index of the node in the ring where the head of the ring has position ID 1.

2.1.3 The Tree-Structured and the Ring-Structured Network

In the tree-structured network, nodes are connected as a *complete n-ary tree*. When the system starts, there is initially only a *master node* that is the root of the tree. When new nodes request to be added to the system, they are added to the bottom level of the tree from left to right to maintain the complete *n-ary* tree structure. If any node in the system fails, a node in the system is used to replace the failure node to keep the given system structure. To minimize the number of nodes involved in this event, we choose a node from *the last node* in the system backwards until we find a node that is still active to replace the failed node. Since it is possible that during the recovery process, the *last node*, its previous node and so on may fail, so that the node with the maximum position ID that is active is used to replace the failed node. All the child nodes of the failed node will connect to the node that is used to replace the failed node. Using the complete n-ary tree as the network topology has two benefits for the network's scalability. The first one is to minimize the height of the tree thus minimize the longest path to send a message to its destination. The second benefit is to fix the maximum degree of the nodes so that when the number of nodes in the system becomes very large there is no need to increase nodes' ability to handle more connections.

In the ring-structured network, nodes are connected as a ring. When the system starts, there is initially only *master node* that is the head of the ring. A new node is added to be the tail of the ring. When a node in the system fails, its previous node and next node are linked together. However, as in the tree-structured network, it is possible that during the recovery process the failed node's previous node and next node may fail, so that we need to find two nearest neighbors of the failed node that are active and link them together. In the ring-structured network, the degree of each node remains to 2 no matter how many nodes are in the system. However the diameter of the ring increases as the number of nodes increases. This results in a longer average message transmission delay.

2.2 Node Responsibilities in the System

2.2.1 Resource Management

To monitor and manage the system status and the resources in the system, in both the tree-structured and the ring-structured networks, the master node gathers the working status and resource usage information of all nodes in the system and sends instructional messages to all the slave nodes periodically. In these systems, messages are transferred in a bidirectional fashion along the network connections. These messages are called *resource management messages*. To minimize the network traffic, each node only exchanges resource management messages with its neighboring nodes. In the case of receiving an instructional resource management messages from the *parent (in the tree-structured network)/previous (in the ring-structured network)* node, each node extracts the message, gets the information it needs, and forwards the messages to its *child (in the tree-structured network)/next (in the ring-structured network)* node(s). In the case of receiving the resource management messages from the *child/next* node(s), each node merges the resource management messages from the child nodes and its own resource management message, and sends the merged message to its *parent/previous* node. In this manner, the master node has the status and resource usage information for all of the nodes in the system and can adjust the work distribution to use the system resources efficiently.

2.2.2 System Structure Maintenance

The master node and slave nodes have their different responsibilities in maintaining the given network structure in the system construction and recovery process.

Master Node's System Structure Maintenance Duties

The master node is the core of the system. It is responsible for maintaining the system structure without or with a minimum of human interference. 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. In the tree-structured network, the system structure is stored in an XML document. In the ring-structured network, the system structure is stored in a double-linked list. The following information of each node is stored in the data storage:

node position ID, node name, node hostname, port number, current client number

In addition to maintaining the system structure data storage, the master node is also responsible for computing optimal methods to construct and recover the distributed system and give the slave nodes instructions to keep the given system structure. In case a new node is to be added to the system, the master node is responsible for computing the optimal method to add the new node to the system. In case an existing node fails, the master node is responsible for computing a method to reconstruct the system to maintain the given structure.

Slave Nodes' System Structure Maintenance Duties

The slave nodes only know the master node and their neighboring nodes i.e. besides the master node, in the tree-structured network, a slave node knows its parent node and its children nodes. In the ring-structured network, a slave node only knows its previous node and its next node. When a new node is to be added to the system, it sends a registration request to the master node, obtains the parent node information from the master node and connects to the parent node. When one of its neighboring nodes fails, the slave node is able to report the failure and can dynamically adjust its position in the system according to the instructions from the master node.

2.3 Connections in the System

In this section, we will describe the transmission protocol used in this project and the different connection types used in the system.

2.3.1 Transmission Control Protocol

The Transmission Control Protocol, TCP, is a connection-oriented protocol that provides a reliable, end-to-end, and full-duplex byte stream over an unreliable internetwork [5].

Services Provided by TCP

Connection-oriented Connections: A TCP client establishes a connection with a given server, exchanges data with that server across the connection, and then terminates the connection [5].

Reliable Services: When TCP sends data to the other side, it requires an acknowledgement. If an acknowledgement is not received, TCP automatically retransmits the data and waits a longer amount of time. After some number of retransmission attempts, TCP will give up [5].

The Ability to Handle Sequencing: The sender associates a sequence number with every segment that it sends. The receiver can reassemble the segments according to their sequence numbers [5].

Flow Control: The receiver always tells the sender exactly how many bytes of data it is willing to accept from the receiver, guaranteeing that the sender cannot overflow the its buffer [5].

Full Duplex Connections: An application can send and receive data in both directions on a given connection at any time [5].

Establish the Connection

The TCP connection is established between two hosts by three-way handshake. Three segments are transmitted before the two hosts are fully synchronized and ready to transmit the data. The steps of the three-way handshake are illustrated as the follows:

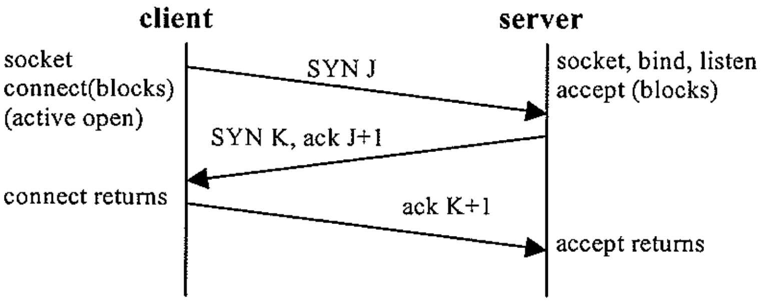

Figure 2-2 TCP three-way handshake

1) The server passively waits for an incoming connection from a client by calling *socket, bind* and *listen*. This is called *passive open.*
2) The client initializes an *active open* by sending a *SYN* segment to tell the server the client's initial sequence number for the segment that the client will send.
3) The server acknowledges the client's *SYN* and sends its own *SYN* containing the initial sequence number for the segment that the server will send. The server sends its *SYN* and the *ACK* in a single segment.

4) The client acknowledges the server's *SYN*.

The number of packets exchanged is three, which is the reason why it is called three-way handshake. Figure 2-2 shows the TCP three-way handshake.

Data Transmission

The transfer of data is dominated by *acknowledgement* of data and the *sequence number* in a segment to allow the receiver to reassemble the segments.

When the sender sends a segment, it expects an acknowledgement from the receiver within a certain time period. If the acknowledgement is not received, the sender retransmits the data.

Sequence numbers allow the receiver to reassemble any out of order packets received from the sender. If a packet arrives that does not have the correct sequence number, the receiver can determine whether it is an old duplicate or it is a packet that has been delayed in the network. The receipt of a duplicate packet allows the receiver to discard it, thus making sure that the receiving process only gets the data once.

Termination of Connection

While it takes three segments to establish a connection, it takes four to terminate a connection. The steps of the connection termination are illustrated as the follows:

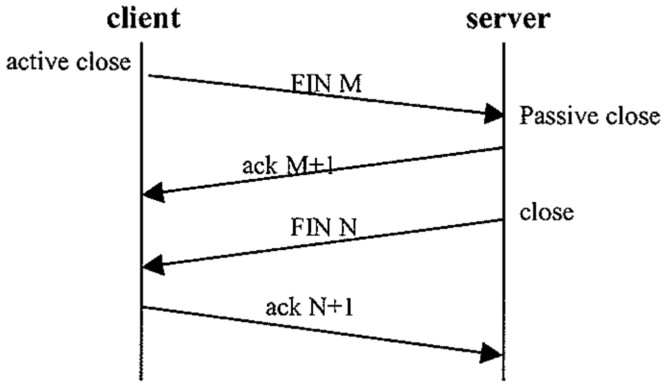

Figure 2-3 TCP termination

1) When one application finished sending data, it sends a *FIN* segment. This is called *active close.*
2) When the other side receives the *FIN*, it acknowledges the *FIN*. This is called *passive close.*

3) Sometime later, the application that received the *FIN* finished sending data; it will send a *FIN* and close the socket.
4) The application that did the active close receives the *FIN* and acknowledges the *FIN*.

Figure 2-3 shows the termination of the TCP connection.

2.3.2 Connection Types in the System

To manage the system resources and maintain the system structure, there are two types of messages transmitted along the network links in the system, 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. To transmit these two kinds of messages, there are two types connections in the system: *permanent connections* and *temporary connections*.

Permanent Connections

The permanent connections constitute the network structure. They 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. In this manner, 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. Permanent connections are the base structure of the system.

Temporary Connections

The temporary connections are used to maintain the system structure. To clearly explain the temporary connections, we define two events that use the temporary connections, the *construction event* and the *recovery event*. The construction event happens when a new node registers to be added to the system. The recovery event happens when a slave node fails. For both the construction and recovery event, there is a group of system structure maintenance messages transmitted along the network connections. For the construction event, a slave node initializes a temporary connection and uses this connection to register to the system; the master node uses the connection to send the instructional information to allow the slave node to be added to the system. For the recovery event, a slave node initializes a temporary connection and reports the failure of its neighboring node; the master node uses the temporary connections to send instructional messages to the slave nodes. 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.

2.4 Summary

This chapter described the structure, components and network connections used in our system.

The network topologies used in this system are the tree-structured network and the ring-structured network. The system has one master node that is the manager of the system and up to hundreds of slave nodes. In the tree-structured network, the master node is the root of the tree. In the ring-structured network, the master node is the head of the tree. Because we need reliable message transmissions, TCP is used as the transmission protocol. There are two types of messages exchanged among the nodes. The resource management messages are transmitted along the permanent connections while the system structure maintenance messages are transmitted along the temporary connections.

3. MESSAGE TYPES AND FORMATS

3.1 XML Introduction

XML stands for eXtensible Markup Language. It is a relatively new markup language and it is a subset of and is based upon a mature markup language called Standard Generalized Markup Language (SGML) [6].

XML was designed to describe and store data. It has a set of rules for designing text formats that let the users structure their data. XML makes it easy for an application to generate, read and search data.

XML is a text-based markup language. Using XML, the user identifies data using tags that are known as "markup". XML tags describe what the data means, rather than how to display it as in HTML. An XML tag acts like a field name in an application program's data structure. It puts a label on a piece of data that identifies it (for example: *<message>...<message>*). XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that reads it. Users are free to add their own tags to an XML document and thus include some structured data within the text. XML provides a basic syntax but does not define the actual tags; the tag set can be extended by the application to accomplish the purpose [7].

Like the field in data structure, in XML, a user can define the field name and is free to use any XML tags that make sense for a given application. The following XML data is an example of XML data used in this system.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Message Type="ControlMessage" Sender="master" Receiver="node9">
  <NewParent ParentName="node1" ParentPortNum="3304" parentHostName="x1">
  </NewParent>
</Message>
```

This example describes a message that the master node sends to a slave node to close the connection to its current parent node and to connect to a new parent node. From it, we can get the following information: this message is a control message sent from the *master* to *node9*, this message notifies the receiver to connect to a new parent. In this message, the master also sends the network-based information of the new parent to the receiver. When the receiver receives this message, it can easily extract the new parent node information and connect to the new parent.

From the above example, we can see, in an XML document, each open tag has a matched end tag. XML tags are case sensitive. For example, the *Message* tag is different from the *message* tag. The data between the open tag and its matched end tag defines an element of XML data. XML documents must have a root element. The root element of the above example is the element *Message*. All XML elements must be properly nested. We can see that the content of *newparent* tag is entirely contained within the scope of *<Message>…</Message>* tag. It is this ability for one tag to contain others that gives XML its ability to represent a hierarchical data structure. XML elements can have attributes in *name/value* pairs just like HTML. In XML, the attribute value must always be quoted [7]. In the above example, the attributes of the element *Message* are *Type*, *Sender*, and *Receiver* while the attributes of the element *NewParent* are *ParentName*, *ParentPortNum*, and *ParentHostName*.

3.2 XML Documents as Data Storage

XML is platform independent and can be shared between different applications. These properties of XML make it possible to use an XML document as data storage in information application systems [8].

The basic idea of using an XML document as data storage is that the tags used in an XML document are for identifying data rather than specifying how the data should be displayed (as in HTML), and the relationships between data elements are provided by simple nesting and references. The application using XML for data storage can easily store information in an XML document and quickly make the information in the XML document available in a simple and usable format. The tags used in the XML document can act as the field names as in a data structure and allow the application to process the XML document easily [9].

3.3 Why Use XML in This Project?

This project is a network management and client-server based application. XML is used as a data structure to store the system structure of the tree-structured network and as the message format transferred between a sender and a receiver. There are number of reasons why we chose XML in this project. This section lists some of them.

Plain Text

Since XML is in plain text format and processor architecture independent, a sender can create an XML message and send the message to the receiver. The receiver can parse the XML message even if it uses a different parser.

Data Identification

XML describes the data. The markup tags identify and structure the data. The element tags and attribute names allow the application to store structured data in an XML document and extract data from an XML document.

Easily Processed

XML is well formed; each open tag has a matched end tag and vise versa. It makes it easy to build a program to process XML data.

Hierarchical

XML documents are hierarchical structured. Hierarchical document structures are faster to access just like searching through a table of contents. They are also easier to rearrange. For example, in the XML document that describes the system structure in a tree-structured network, if a node changes its position in the tree, it is easy to move the node to a new location and drag everything under it along with it instead of moving every piece of data under it individually.

3.4 System Structure XML Documents in the Tree-Structured Network

In the tree-structured network system, the system structure is stored as an XML document in the master node. Each node in the system is an element in the XML document. The following is the information listed in the XML document.

```
<nodename ID="id" Level="level" PortNum="port_number"
            HostName="hostname"  CurrentConnection="current_children_number" >
```

```
  <childnode>*
</nodename>
```

The master node is responsible for creating and maintaining the system structure XML document and it uses this XML document to maintain the structure of the network. When a new node is to be added to the system, the master node queries the XML document and computes the parent node for the new node. When the new node connects to its parent node successfully, the master node updates the XML document to contain the new node. When a node in the system fails, the master node queries the XML document to find a new node to replace the failed node. After this event is handled successfully, the master node updates the XML document.

3.5 Message Format and Types Transmitted in the System

XML documents are also used as the messages transferred between nodes. Because the XML messages are in the plain text format and there are tags used in the XML documents, it is easy for a node to parse the XML messages and extract the useful information.

There are two kinds of messages transmitted along the system: *Resource Management Messages* and *System Structure Maintenance Messages*.

Resource management messages that are transmitted along *permanent connections* carry the working status and resource management information to be exchanged among nodes.

System maintenance messages carry information needed to maintain the system structure. Some of the system maintenance messages are transmitted along the temporary connections while some are transmitted along the permanent connections.

3.5.1 Resource Management Messages

Resource management messages carry the working status and resource management information of each node in the system. They are transmitted along the permanent connections in the system. Resource management messages are merged level-by-level and transmitted to the master node. The master node uses resource management messages to gather the working status information of each node. The following is the format of the resource management message.

```
<Message Type = "ResourceManageMessage ", Time="the message generated time"
```

```
          Wall_Time=”the message generated wall_time”  Sender = sender_name,
          Receiver = receiver_name>
      <CurrentConnection>number of current connection></CurrentConnection>
      <MemoryAvialable>the current free memory</MemoryAvialable>
      <ProcessorAvialable>the current free processor number</ProcessorAvialable>
 </Message>
```

3.5.2 System Structure Maintenance Messages

There are seven types of system structure maintenance messages: *Registration Message, Report Message, Response Message, Control Message, Greeting Message, Bye Message, and Acknowledgement Message.*

Registration Message

When a new node needs to be added to the system, it sends its network-based information to the master node in a registration message. This information is added to the system structure database after the new node connects to the system successfully. The registration messages are transmitted along temporary connections. The following is the format of the registration message.

```
<Message Type= “RegistrationMessage”, Sender = sender_name>
    <NodeInformaiton  PortNum = portnumber_of_the_node,
                      NodeHostName = hostname_of_the_node,
                      NodeName = name_of_the_node >
    </NodeInformaiton>
</Message>
```

Report Message

When a node in the system fails, all its neighboring nodes will send a report message to the master node. The report message includes the failed node's name and the relationship with the sender. The report messages are transmitted along temporary connections. The following is the format of the report message.

```
<Message Type = “ReportMessage”, Sender =sender_name >
    <ReportInformaiton  PeerName = name_of_failrenode,
       ReportType = ”PeerDown”, Relation = “CHILD/PARENT”/>
   </ReportInformation>
</Message>
```

Response Message

When the master node receives a registration message or report message that reports the parent node's failure from a slave node, it computes the new parent information for the slave nodes and sends the network-based information of the parent node to the requested node. The master node does not send a response message to a slave node that reports the failure of the child node. The response messages are transmitted along the temporary connections. The following is the format of the response message.

```
<Message Type= "ResponseMessage"  Sender="master"
         Receiver = receiver_name>
        <ParentInformaiton ParentName=name_of_parent_node
                           ParentPortNum =parent portnumber
                          ParentHostName = parent hostname>
        </ParentInformaiton>
</Message>
```

Control Message

The control message is for the master node to notify a node to connect to a new parent node. To keep the system structure, it is necessary for the master node to initialize a connection to a node and tell it to change its position in the system. In the control message, the master node sends the network-based information of the new parent node to the selected node. The control messages are transmitted along the temporary connections. The following is the format of the control message.

```
<Message Type= "ContrlMessage", Sender="master",
         Receiver = receiver_name >
   <NewParent ParentName=name_of_parent, ParentPortNum = portnumber_of_parnet,
              ParentHostName = hostname_of_parent>
   </NewParent>
</Message>
```

Acknowledgement Message

When a node connects to its parent node, it sends an acknowledgement message to the master. Only after the master node receives the acknowledgement message, does it change the structure of the system. The acknowledgement messages are transmitted along the temporary connections. The following is the format of the resource acknowledgement message.

<***Message*** *Type=“AckMessage ”, Sender =sender_name, Receiver = “master”*>
</***Message***>

Greeting Message

When a node first connects to its parent node, it sends a greeting message to its parent node. Upon receiving the greeting message, the parent node has the name of the new client. The greeting messages are transmitted along the permanent connections. The following is the format of the greeting message.

<***Message*** *Type = “GreetingMessage ”, Sender = sender_name, Receiver = receiver_name*>
</***Message***>

Bye Message

When a node receives a control message from the master node to connect to a new parent node, it sends a bye message to its current parent node to notify its current parent that it will terminate the connection. Otherwise the parent will assume the failure of this node and report it to the master node. The bye messages are transmitted along the permanent connections. The following is the format of the bye message.

<***Message*** *Type=“ByeMessage ”, Sender =sender_name, Receiver = receiver_name*>
</***Message***>

Each node in the system knows the format of different kinds of messages. Each node extracts the information it needs from the message it receives. Because the XML messages are in plain text formats and there are tags used in the XML messages, it is easy for a node to process these messages and gets the useful information. All these different kinds of messages work together to manage the system resources and maintain the given system structure.

3.6 Summary

This chapter gave a brief introduction to XML and defines the XML message formats used in this system. XML is a markup language used to describe data. It is in plain text format and can be easily processed. So XML is used in this project as the data structure to store the system structure and as the message formats exchanged between nodes.

4. SYSTEM DESIGN AND IMPLEMENTATION

4.1 System Design

As discussed in chapter 2, the master node and slave nodes have different roles in the management of the system's resources and the maintenance of system structures. To accomplish their functional goals, they have to communicate with other nodes and to process the messages that they send and receive. The messages transmitted on the network are in XML format. Thus, both the master node and slave nodes have to call network communication methods and XML document processing methods. The network communication methods include server methods and client methods. They are implemented in the network communication module. The XML processing methods include building XML documents and parsing XML documents. They are implemented in the XML document-processing module. The network communication methods and XML document processing methods were abstracted out and shared by both the master node and slave nodes. Figure 4-1 shows the software modules and their relationships in this distributed management system.

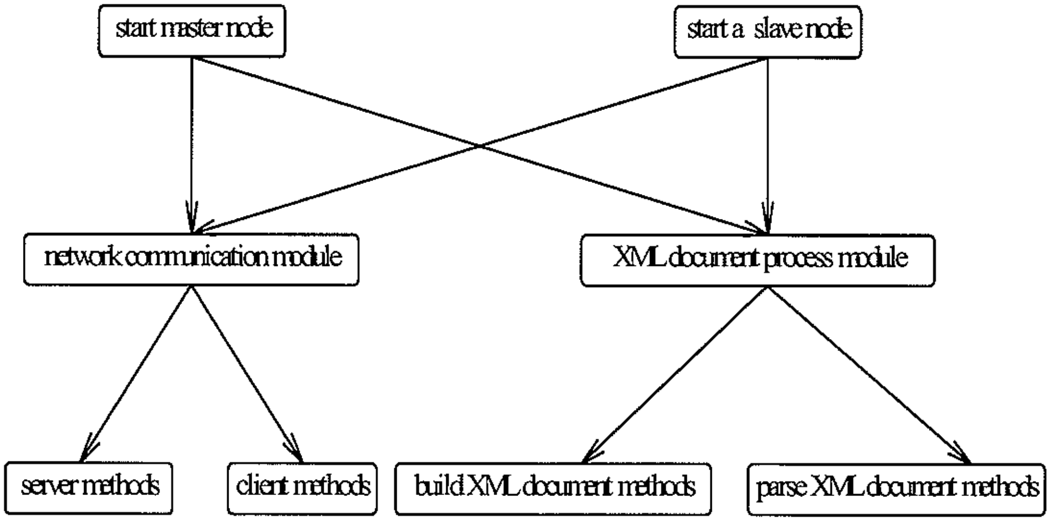
start master node
start a slave node
network communication module
XML document process module
server methods
client methods
build XML document methods
parse XML document methods

Figure 4-1 Software modules

4. 2 System Implementation

4.2.1 The Concurrence of the System

A concurrent server is a server that can handle multiple incoming requests concurrently [10]. A server must be explicitly programmed to handle concurrent requests because multiple clients contact a server using its single, well-known port. Figure 4-2 shows the architecture of the concurrent server.

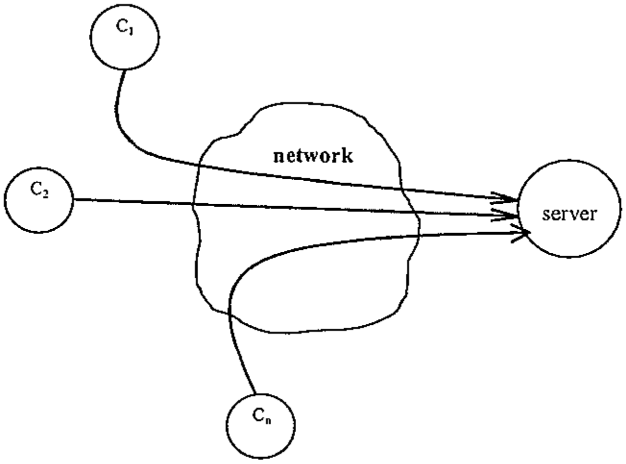

Figure 4-2 The concurrent server architecture

In our system, both the master node and slave nodes are concurrent servers. The concurrence of the systems can be implemented using multiple processes or multiple threads.

A *process* is a running program that has a single address space and a single thread of control with which to execute a program within that address space. To execute a program, a process has to initialize and maintain state information. The state information includes page tables, swap images, file descriptors, outstanding I/O requests, saved register values, etc. The size of this state information makes it expensive to create and maintain processes as well as to switch between them [2].

A *thread* is only part of a running program. Once created, a thread begins to run concurrently with the rest of the program. All threads within a process share the same global memory. This makes the sharing of information easy between the threads [2].

When a new process starts, it gets its own "memory space" - that is, all data needed by the process is its own, and not shared with other processes (except through special mechanisms). In contrast, two threads that are part of the same process can share data in the same way that two functions that are part of the same program can share data. In this way, a thread is sometimes referred to as a "lightweight process". A thread is a finer unit of execution than a process, just as a function is a finer unit of code than an entire program.

There are system calls to create new processes and new threads. Since the child threads share the same global memory as its parent thread, thread creation can be much faster than process creation.

In our system, each node accomplishes its resource management task and system structure maintenance task concurrently. New threads or processes need to be frequently created to accomplish different tasks; information is frequently exchanged among these threads or processes. Due to the lower cost in creating new threads and in the sharing of information between threads, we chose the threaded model to implement the concurrency of the system.

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 the same maximum number of child connections, *n*, which is the maximum number of child nodes one node can have. 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.

The Self-Construction of the Tree-Structured Network

When the system starts, there is only the *master node* that is the root of the tree. The master node is the manager of the system. To accomplish its management tasks, when it starts, the master node initializes the system structure database, and opens a port to listen for requests from slave nodes.

To construct the distributed system, new nodes are added to the system dynamically in the order of their register requests.

The steps involved in constructing the system

1. *The new node sends a registration request to the master node. In the registration request, the new node sends its network-based information.*
2. *The master node queries the structure database, and computes the parent node for the new node.*
3. *The master node sends a response message that contains the parent information to the requesting node.*

4. *The new node tries to connect to the parent node specified in the response from the master node.*
5. *If the new node successfully connects to its parent node, it sends an acknowledgement message to the master node.*
6. *After receiving the acknowledgement message from the new node, the master node adds the new node to the system structure database.*
7. *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.*

Since it is possible that the new node can not connect to its assigned parent node due to the possibility of either the new node or the parent node's failure during the construction event (see chapter 2), only after the master node receives an acknowledgement from the new node, will it update the system structure database. Figure 4-3 shows the process of a new node to be added to the system.

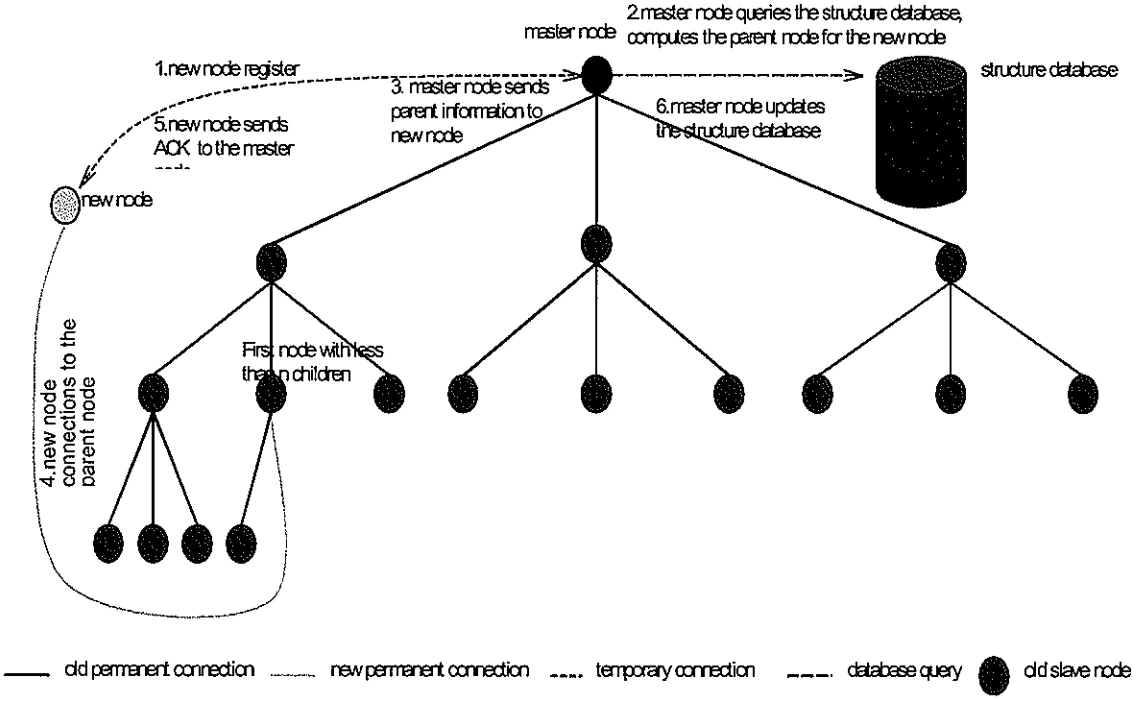
master node
2.master node queries the structure database, computes the parent node for the new node
structure database
1.new node register
3. master node sends parent information to new node
6.master node updates the structure database
5.new node sends ACK to the master
new node
4.new node connections to the parent node
First node with less than n children
old permanent connection
new permanent connection
temporary connection
database query
old slave node

Figure 4-3 The process of a new node to be added to the system

Computing the Parent Node for the New Node

To keep the structure of the complete *n-ary* tree, the new node is added to the bottom level of the tree which is filled from left to right. That is, it is connected to the first node with less than *n* children by the breadth first search.

Theorem: In a complete *n-ary* tree, if the number of nodes in the tree is *m*, the order of the first node with less than *n* children by breath first search is $\left\lfloor \frac{m-1}{n} \right\rfloor + 1$.

Proof:

In a complete n-ary tree with m nodes, each node except the root is the child of some other node. The number of child nodes in the tree is $m-1$. *Let node f be the first node with less than n child nodes and its number of child nodes is c* $(c < n \,\&\, c = (m-1)\%n)$. *We have:*

$$m-1 = (f-1)\times n + c \Rightarrow f = \frac{(m-1)-c}{n} + 1 \Rightarrow f = \left\lfloor \frac{m-1}{n} \right\rfloor + 1$$

End proof

According to the above theorem, when a new node is to be added to the tree-structured network with *m* nodes, the new node with ID *m+1* becomes the child of the node with ID $\left\lfloor \frac{m-1}{n} \right\rfloor + 1$ and is now *the last node* of the system.

The Self-Recovery of the Tree-Structured Network

During the execution of the system, some nodes may fail. To accomplish the system functions in the case of a nodes' failure, the system must recover automatically, that is, it should have self-recovery abilities.

When a node in the tree-structured network system fails, all its neighbors will report the failure to the master node. The reason why all the neighbors instead of only a single neighbor will report this failure can be illustrated as follows:

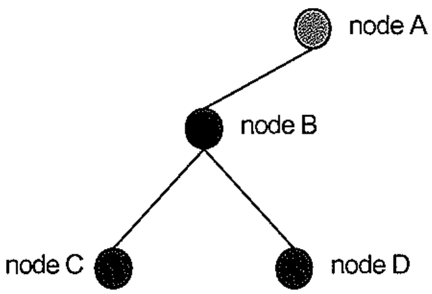

Figure 4-4 A segment of a tree when node B fails

1. *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 4-4, 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.*
2. *It is possible that two adjacent nodes fail simultaneously, so one node cannot rely others to report the failure. In figure 4-4, 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.*
3. *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 4-4, node C and node D get new parent information after they report the failure to the master node.*

To minimize the number of nodes involved in the *recovery event* (see chapter 2), 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 chapter 2).

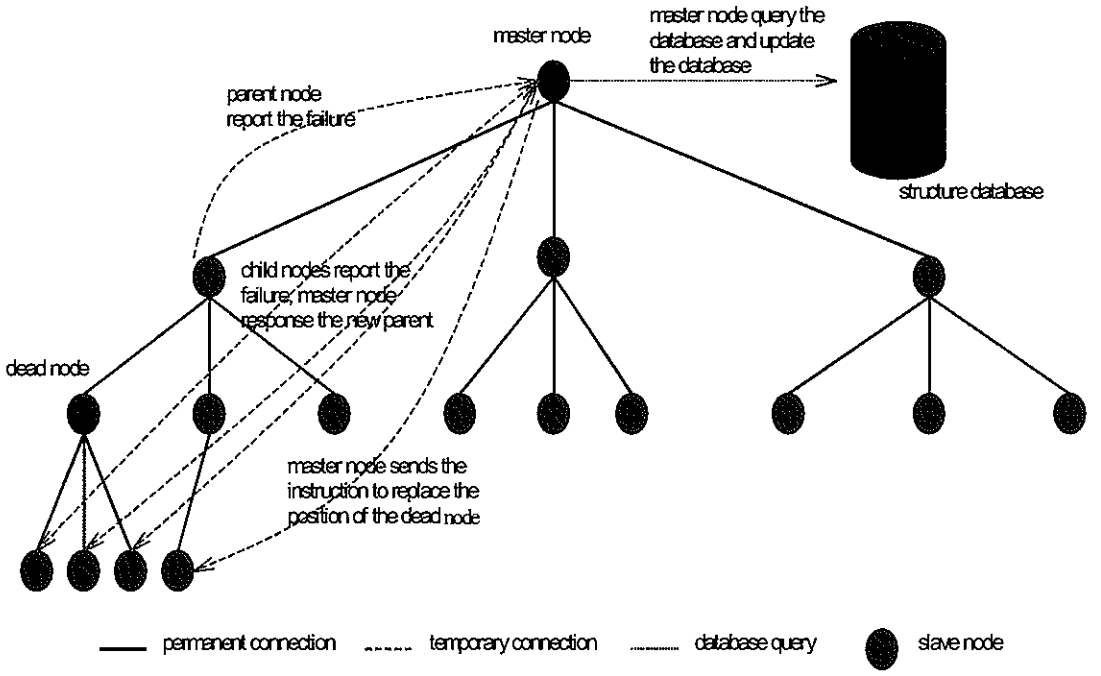
master node
master node query the database and update the database
parent node report the failure
structure database
child nodes report the failure, master node response the new parent
dead node
master node sends the instruction to replace the position of the dead node
permanent connection
temporary connection
database query
slave node

Figure 4-5 The process to recovery the system

For the parent of the failed node, after it reports the failure, it closes the thread that was communicating with the failed node. For example, in figure 4-4, after node A reports node B's failure, it closes the connection with node B. For each of the child nodes of the failed node, after they report their parent node's failure, they will receive a response message that contains the network-based information of the new parent from the master node. If after a certain time period, the master node has not received a report from all of the children of the failed node, it will compute a method to handle all of the nodes that have not reported the current failure. To do this, the master node first detects if each of the children is still active. If it is, the active child will be moved to be the child of the new parent; otherwise, a node will be used to replace its position as the child of the new parent. For example, in figure 4-4, if node C does not report node B's failure, the master node will first detect if node C is still active. If it is, node C will receive a message from the master node and connect to the new parent node. Other wise, a node in the system will replace the position of node C. Figure 4-5 shows the recovery of the tree-structured network.

4.2.3 The Construction and Recovery of the Ring-Structured Network

In the ring-structured network, the ring is stored as a Double Linked List. The master node is the head of the ring. The position ID of a node is the position index of the node in the ring while the master node, the head of the ring, has position ID 1.

The Self-Construction of the Ring-Structured Network

As in the tree-structured network, when the system starts, there is only the *master node* that is the head of the ring in the system. The master node is the manager of the system. When it starts, it initializes the system structure database, and opens a port to listen for requests from slave nodes.

To construct the network, 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, that is, if the number of nodes in the ring is *m*, the new node's parent is the node with position ID *m* and the new node will have position ID *m+1*. Figure 4-6 shows the construction of the ring-structured network

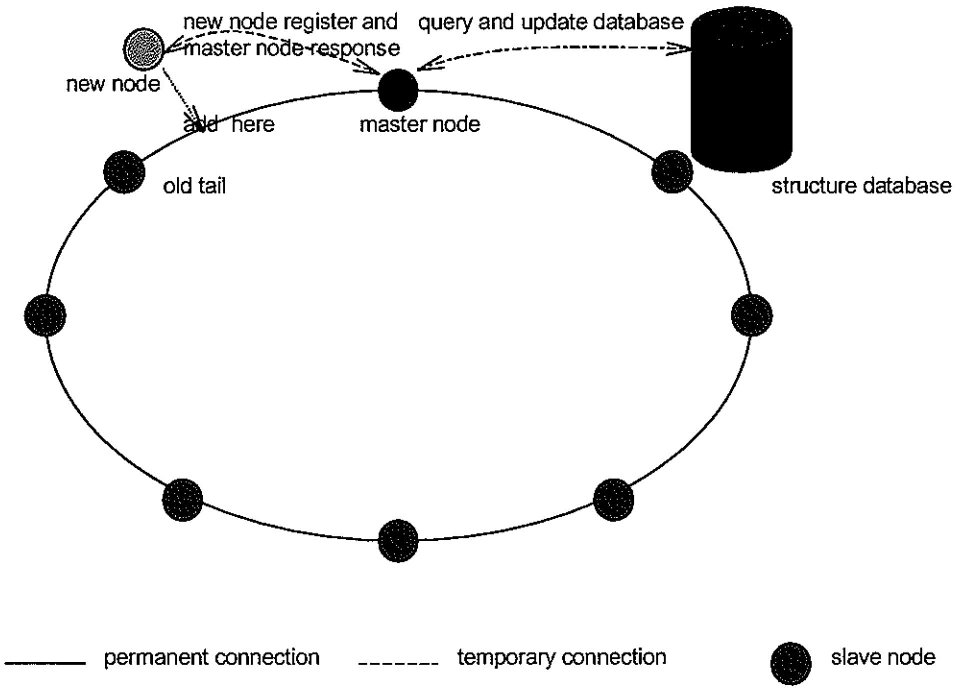

Figure 4-6 the construction of the ring-structured network

The Self-Recovery of the Ring-Structured Network

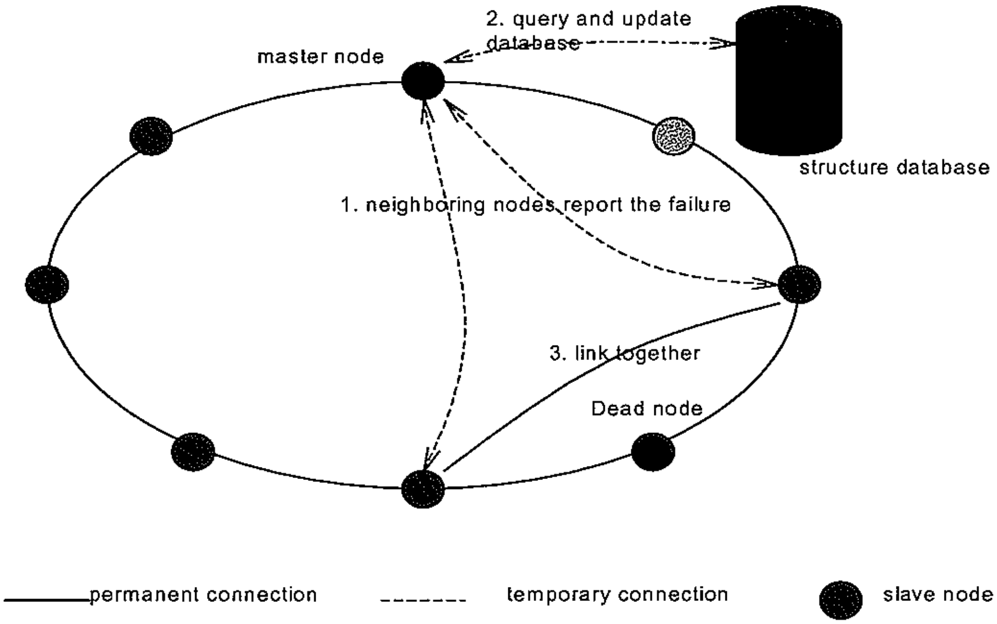

Figure 4-7 The recovery of the ring-structured network

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. Basically, the failed node's previous node and next node will link together. However, it is possible that during the recovery process the failed node's previous node and next node may fail, so that the master node will find two nearest neighbors of the failed node that are alive and link them together. The failed node will be deleted from the system. Figure 4-7 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. This section describes these threads.

Main thread: The main thread is the entry point of the program. It is responsible for creating working threads to handle the different functions of the master node. Its responsibilities are initializing the server, creating the system structure management thread, creating the resource management thread, and creating a new connection handling thread for each new client node. The main thread runs forever.

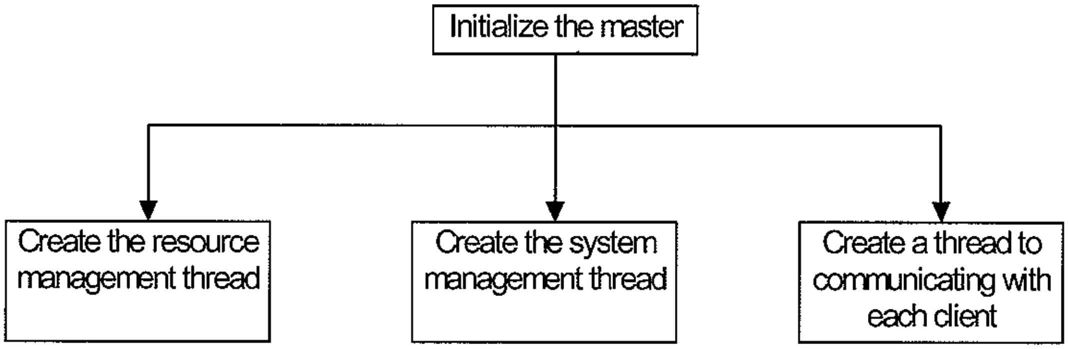
Initialize the master
Create the resource management thread
Create the system management thread
Create a thread to communicating with each client

Figure 4-8 The functions of the main thread

Resource management thread: The resource management thread is used to accomplish the resource management tasks of the master node and sends the instructional messages to its child nodes. At present, the role of the resource management task is to report the current system structure periodically.

System structure management thread: The system structure management thread 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. It has a predefined port number that can be connected to by each slave node. To ensure the integrity of the system structure, the system structure management function is implemented as an iterative server. This thread handles the requests from clients in a sequential way.

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. The master node is expecting any kind of request and can handle either registration or reporting requests.

INPROCESS STATE: The master node is handling a recovery event. Since each node in the system may have more than one neighbor, each is responsible for reporting its neighbor's failure, so that the master node may have to handle more than one report request for a failure event. In the INPROCESS STATE, the master node is only expecting the request that reports the current failure and cannot handle new registration request and report requests that report the failure of nodes other than the current one. If this kind of request comes, the master node sends a response message to the slave node to inform it that the master node cannot handle this request. This request will be sent again.

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 "INPROCESS STATE".

TIMEOUT STATE: If the system stays in the INPROCESS STATE longer than the MAXIMUM DELAY and a new request other than the request that reports the current failure, the system changes to "TIMEOUT STATE". Figure 4-9 shows the transformation of the system states.

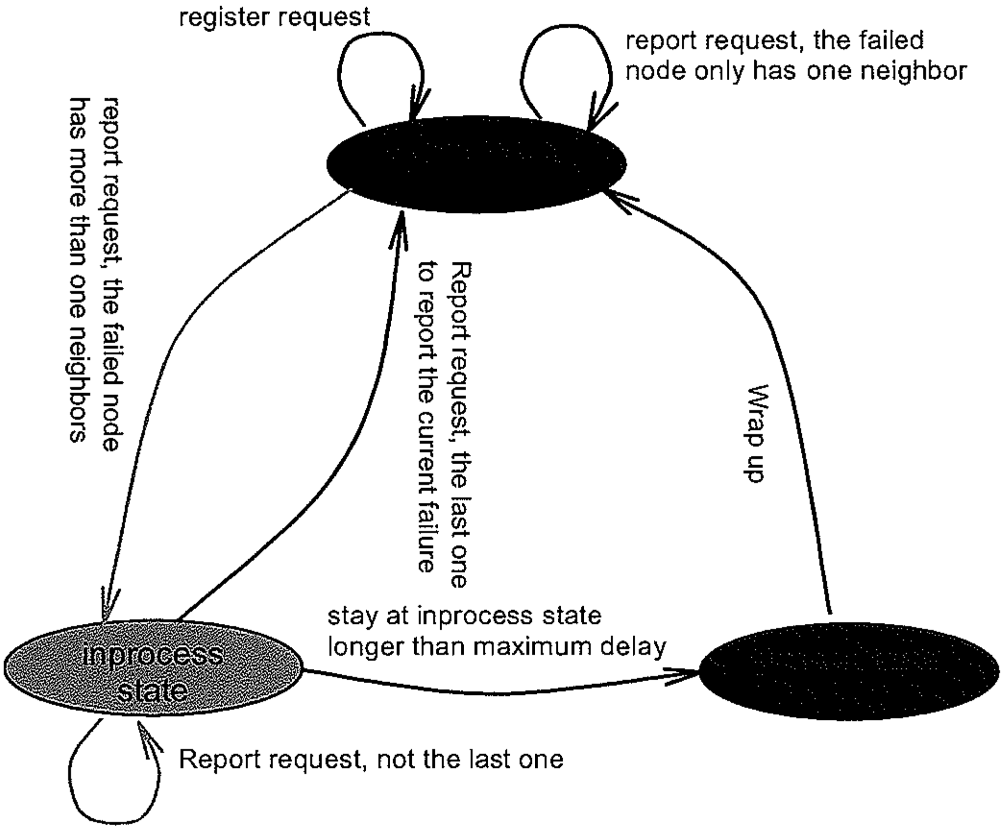
register request
report request, the failed node only has one neighbor
report request, the failed node has more than one neighbors
Report request, the last one to report the current failure
Wrap up
inprocess state
stay at inprocess state longer than maximum delay
Report request, not the last one

Figure 4-9 The system state transformation

The duties of the system structure management thread: The system structure management thread has two major duties: one is to handle the construction event; the other one is to handle the recovery event.

Handle construction event: When a new node registers to join the system, the system structure management thread is responsible for computing the parent node for this node and adding this node to the system. It works in three different cases.

Case1: the system is in the NORMAL STATE: The master node computes the parent node of the new registering node and sends the parent information to the registering node. If the master node receives an acknowledgement from the new slave node, the new node is added to the system and is assigned an ID. Otherwise, the manager just ignores this request. After finishing the handling of this request, the system stays in the NORMAL STATE.

Case 2: The system is in the INPROCESS STATE: The master node is handling a recovery event. The master node cannot handle the new registration request. The new node will send another registration request again. The system stays in the INPROCESS STATE.

Case 3: The system is in the TIMEOUT STATE: The master node is handling a recovery event and stays in the INPROCESS STATE longer than the MAXIMUM DELAY. Upon receiving the new registration request, the master node terminates handling the recovery event by computing a method to wrap up the current recovery event.

Handle recovery event: When a node in the system fails, the system structure management thread is responsible for computing a method to recovery the system structure. This function works in the following different cases:

Case 1: the system is in the NORMAL STATE: The master node receives the first report request that reports the new failure. The master node sets the PROCESS TIME for this event, and handles this report request. If the failed node only has one neighbor, after handling this event, the system stays in the NORMAL STATE, otherwise, the system state changes to the INPROCESS STATE.

Case 2: the system is in the INPROCESS STATE: The master node is handling a recovery event. It only expects report requests for the current failed node. If the request that arrives is this kind of request, the master node handles it otherwise it is ignored and will be sent again. If this report request is the last report for the current recovery event, after handling this request, the system state changes to NORMAL STATE, otherwise the system stays in the INPROCESS STATE.

Case 3: the system is in the TIMEOUT STATE: the system stays in the INPROCESS STATE longer than the MAXIMUM DELAY. Upon receiving a new request that does not report the current failure, the master node terminates handling the recovery event by computing a method to wrap up the current recovery event. We call the node that should have reported but has not reported the current failure the *unreported node.* The steps involved in wrapping up the current recovery event are as follows:

1. *The master node detects if the unreported node is still active.*
2. *If it is active, the master node will send an instructional message to tell it to connect the new parent node.*

3. *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. Its duties are exchanging the resource management messages with the child node and reporting the failure of the child node to the system structure management thread in case that the child node fails.

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. Its responsibilities are initializing the node, creating the *resource management thread* to exchange the working status and resource usage information, creating a *client thread* to handle the communication with its parent node, creating a *message merging thread* to merge messages from all its child nodes, and creating a *connection-handling thread* to handle the communication with each child node. The main thread runs forever. Figure 4-10 shows the functions of the master thread.

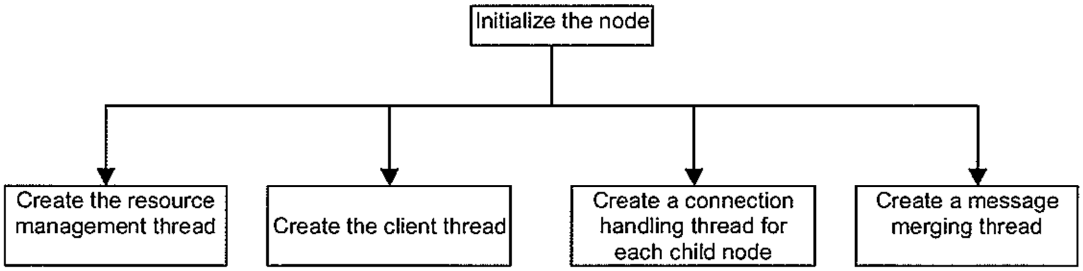
Initialize the node
Create the resource management thread
Create the client thread
Create a connection handling thread for each child node
Create a message merging thread

Figure 4-10 The functions of the main thread

Resource management thread: It is responsible for generating and reporting its working status and the resource usage information. The working status and resource usage information will be merged level by level and sent to the master node.

Client thread: It is used to handle the communication with the parent node. In case its parent node fails, this thread is responsible for reporting its parent node's failure to the master node, and connecting to the new parent node according the instructions from the master node.

Connection-handling thread: When a new connection request comes, the main thread creates a new connection handling thread to communicate with the new client. There are two kinds of connection-handling thread . The first one is to handle the communication with the child node, and in the case of child node fails, to report the failure of the child node to the master node. The second kind of connection handling thread handles the instructions from the master node. In this case, the master node initializes a connection and sends a system structure management message to this node. When this thread receives the instruction message from the master node, it will perform its task according to the message from the master node. After the completion of its task, this thread terminates.

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. In order for the ancestor node to monitor all its offspring, the messages are merged by level and sent to the upper level parent. During the path of message transmission to the master node, each node extracts the information it needs, merges all the messages from its child nodes and its own message, and sends the merged message to its parent node. In the ring-structured network, each slave node has only one child node. The slave node uses the message-merging thread to merge the message from its child node and its resource management message. In the tree-structured network system, each node may have more than one child node. The slave node uses the message-merging thread to merge the message from all its child nodes and its own message. By merging messages rather than simply forwarding all messages from child nodes the network traffic can be decreased and the connection burden of the upper level nodes can be reduced. For example, in an *n-ary* tree, if a node has m levels of offspring, it only needs to handle n different connections using the merging message schema instead of handling $m \times n$ different connections if the offspring send messages individually. This decrease is significant for the master node when the number of nodes in the system scales very large. Figure 4-11 shows the process of monitoring message transmission and merge.

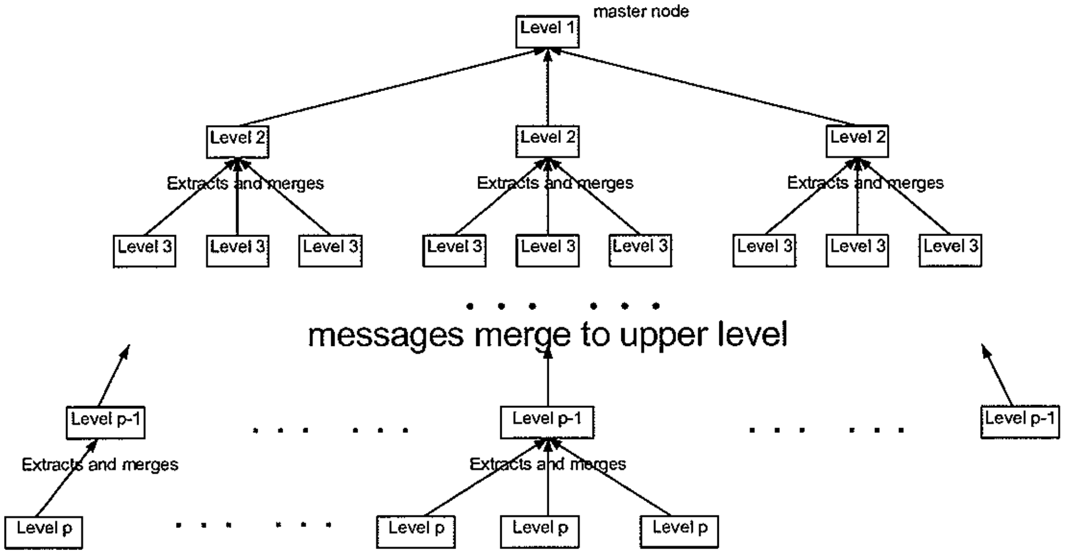

Figure 4-11 The process of monitoring message transmission and merge

4.2.6 The XML Document Processing Module

The master node and slave nodes need to process the messages that they send and receive. The messages are in the XML format. An XML parser is needed to build and parse the XML documents. The Xerces-C++ parser is used in our system as the XML parser [14].

The Xerces-C++ parser is a validating XML parser written in a portable subset of C++. It makes it easy to give the application the ability to read and write XML data. A shared library is provided for parsing, generating, manipulating, and validating XML documents. The parser provides high performance, modularity, and scalability.

The XML document processing module provides all the methods needed by the master node and slave node to process the XML messages. There are 2 classes in the XML document processing module, the *BuildMsg* class and the *ParseMsg* class. The *BuildMsg* class provides methods to build a XML document, add an element to an existing XML document, set attributes to an existing element, move an element to a new position, and merge XML documents that are in the same format. The *ParseMsg* class provides methods to get the element value, get the attribute value of an element. Using these 2 classes, the master node and slave nodes can easily build and parse XML messages.

4.3 Summary

This chapter has described the system design and implementation of this distributed management system.

There are 4 software modules in this system, the master node module, the slave node module, the network communication module and the XML document-processing module. The network communication module and the XML document-processing module provide the basic methods that the master node and slave nodes need to accomplish their functional goals. The master node and slave nodes are concurrent servers that accomplish their system structure management functions and resource management functions. POSIX threads are used to accomplish the concurrency of the system due to its lower overhead for generating and sharing information between threads. The Xerces-C++ parser is used as the XML parser due to its high performance, modularity, and scalability.

5. TEST ENVIROMENT 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 G4 cluster is the first cluster in the SCL and one of the first in the world to run the new Scalable Systems Software Resource Management suite. This software is designed for scalability and fault tolerance and removes many of the limitations of previous batch systems

In our system, there are 2 types of nodes, the master node and slave nodes. The master node is the manager of the system. It requires more system resources than a slave node. The master node is running on a specified node of the cluster and has two predefined ports to listen for the connection requests from clients. One port is for its direct child nodes to connect to it and to exchange the resource management messages; the other one is for all the slave nodes to connect it and to exchange the system structure management messages. To test the self-construction and the self-recovery ability 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 system's ability to maintain the given system structure, that is, 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.

5.2.1 The Test Result of Maintaining the System Structure

To test the system's ability to maintain the given system structure, we tested the construction and recovery performance of the binary tree-structured network, ternary tree-structured network, 5-ary tree structured network, and the ring-structured network with different number of slave nodes in the system. Table 5-1 shows the average time used for a new node to be added to the system and the average time used to recover the system structure when a node in the system fails.

Table 5-1 The construction and recovery performance of the system (ms)

number of slave nodes topology	10		20		50		100	
	Construction	recovery	construction	recovery	construction	recovery	construction	recovery
binary tree	72.2	130.4	64.4	132.3	69.2	165.5	72.6	187.4
ternary tree	81.6	131.0	77.1	133.2	81.8	177.0	74.5	192.8
5-ary tree	77.8	184.5	72.3	220.9	71.4	279.8	74.8	299.0
Ring	74.5	110.7	79.5	130.4	70.1	156.1	67.8	164.3

5.2.2 Round Trip Time for Resource Management Messages

The resource management messages are transmitted along the permanent connections in the system. The messages from the slave nodes are merged upward level by level and sent to the final destination, the master node. The master node sends the resource management messages to its direct child nodes and the messages are forwarded downward level by level to each of the slave nodes in the system. We tested the RTT with zero payload and an XML payload (252 bytes) in the binary tree-structured network, ternary tree-structured network, 5-ary tree-structured network, and the ring-structured networks.

Table 5-2 RTT in a binary tree-structured network (ms)

number of slave nodes topology	10		20		50		100	
	0_payload	xml_payload	0_payload	xml_payload	0_payload	xml_payload	0_payload	xml_payload
1	0.5	32.6	0.6	33.8	0.4	34.2	0.5	35.7
3	2.2	45.8	3.5	50.3	2.1	51.8	1.8	49.0
4	NA	NA	4.8	155.1	6.5	142.7	8.2	170.8
MAX_RTT	2.2	45.8	4.8	155.1	7.1	185.0	11.7	210.7

Table 5-2 through table 5-5 show the RTTs with different types of payload in different systems. The MAX_RTT in these tables means the **MAX**imum **R**ound **T**rip **T**ime in each system.

Table 5-3 RTT in a ternary tree-structured network (ms)

number of slave nodes topology	10		20		50		100	
	0_payload	xml_payload	0_payload	xml_payload	0_payload	xml_payload	0_payload	xml_payload
1	0.7	37.2	0.4	34.5	0.5	35.6	0.6	36.4
3	NA	NA	4.9	50.8	1.3	52.2	2.2	47.3
4	NA	NA	NA	NA	7.3	157.3	6.3	152.1
MAX_RTT	1.1	40.2	4.9	50.8	7.3	157.3	9.5	170.1

Table 5-4 RTT in a 5-ary tree-structured network (ms)

number of slave nodes distance	10		20		50		100	
	0_payload	xml_payload	0_payload	xml_payload	0_payload	xml_payload	0_payload	xml_payload
1	0.5	34.7	0.6	38.7	0.4	38.6	0.5	33.5
2	1.0	38.7	1.1	39.5	1.2	52.1	1.2	41.6
3	NA	NA	NA	NA	4.7	98.5	4.8	97.7
MAX_RTT	1.0	38.7	1.1	39.5	4.7	98.5	4.8	97.7

Table 5-5 RTT in a ring-structured network (ms)

number of slave nodes distance	10		20		50		100	
	0_payload	xml_payload	0_payload	xml_payload	0_payload	xml_payload	0_payload	xml_payload
1	0.5	39.9	0.4	32.3	0.5	37.9	0.4	40.3
5	13.3	52.2	4.9	51.7	5.4	50.9	4.7	46.6
20	NA	NA	15.7	223.8	14.8	213.8	18.2	189.0
MAX_RTT	1.9	161.6	15.7	223.8	47.3	517.5	101.2	957.3

5.3 Results Analysis

Figure 5-1 shows that in the distributed systems with different network topologies, 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.

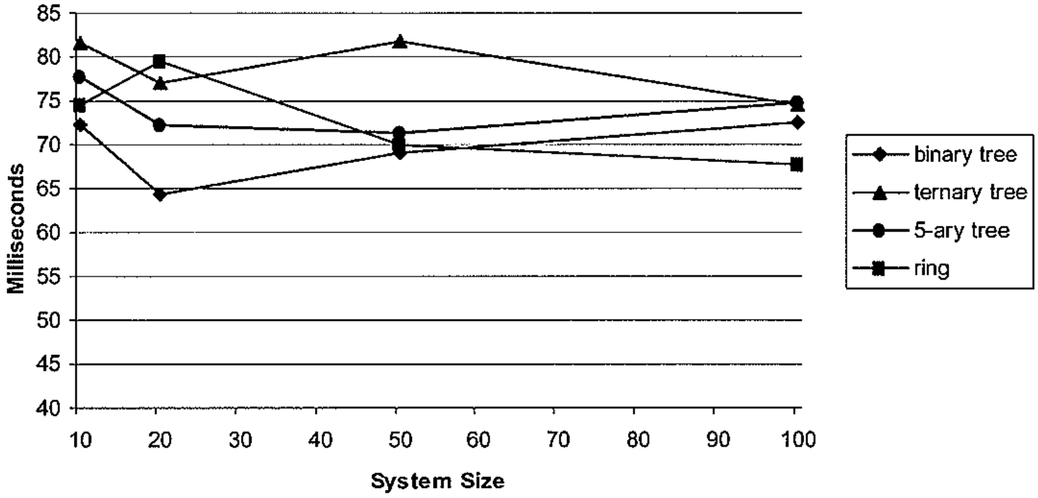

Figure 5-1 The construction performance

Figure 5-2 shows 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. For example, in the ring-structured network, when a node fails, the master node only has to process the report requests from the failed node's previous node and next node while in the 5-ary tree-structured network, when a node fails, the master node may have to process up to 6 report requests. 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 an active node with the maximum position ID to replace the failed node. The more nodes one system has, the more complex to find a node to replace the failed node, and the more time needed to recover the system. In the ring-structured network, this increase is not as significant as in the tree-structured network. This is because in such systems what the master node does is to find two nearest neighbors of the failure node and let them connect together.

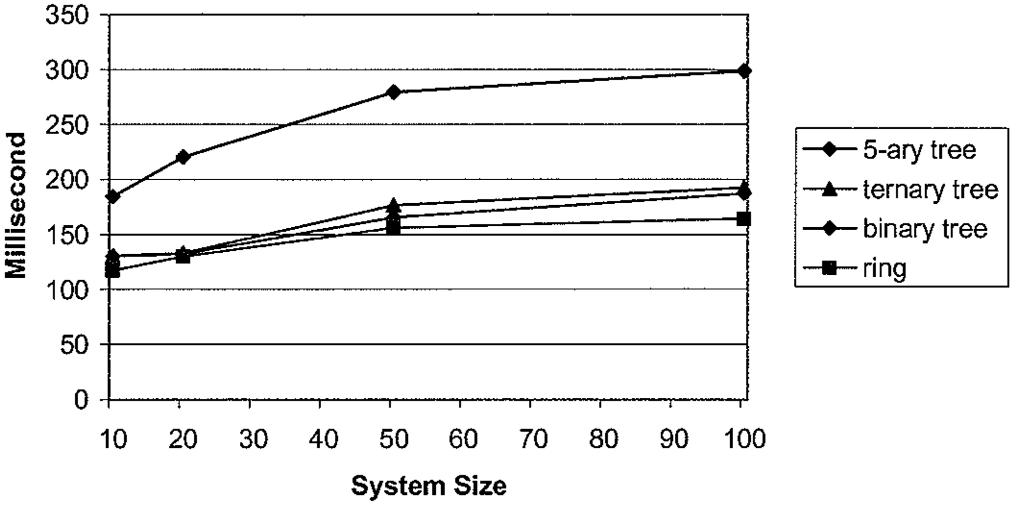

Figure5-2 The recovery performance

Figure 5-3 and figure 5-4 shows the maximum RTT for zero payload and XML payload messages transmitted in the binary tree-structured, ternary tree-structured, 5-ary tree-structured, and the ring-structured network with 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 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$\lceil \log_n m(n-1)+1 \rceil - 1$. Thus in an *n-ary* tree-structured network, the maximum RTT grows much more slowly than in a ring-structured network. The growth rate decreases as the degree of the tree, *n* increases. Figure 5-3 and figure 5-4 show that the maximum RTT in a 5-ary tree-structured network grows slower than that in the binary tree-structured network and the ternary tree-structured network.

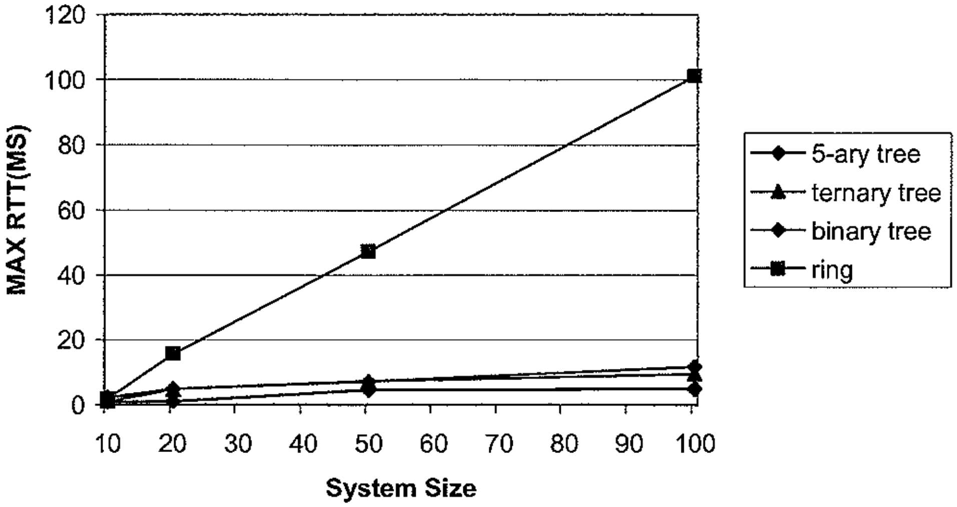

Figure 5-3 The maximum RTT with zero payload in the distributed systems

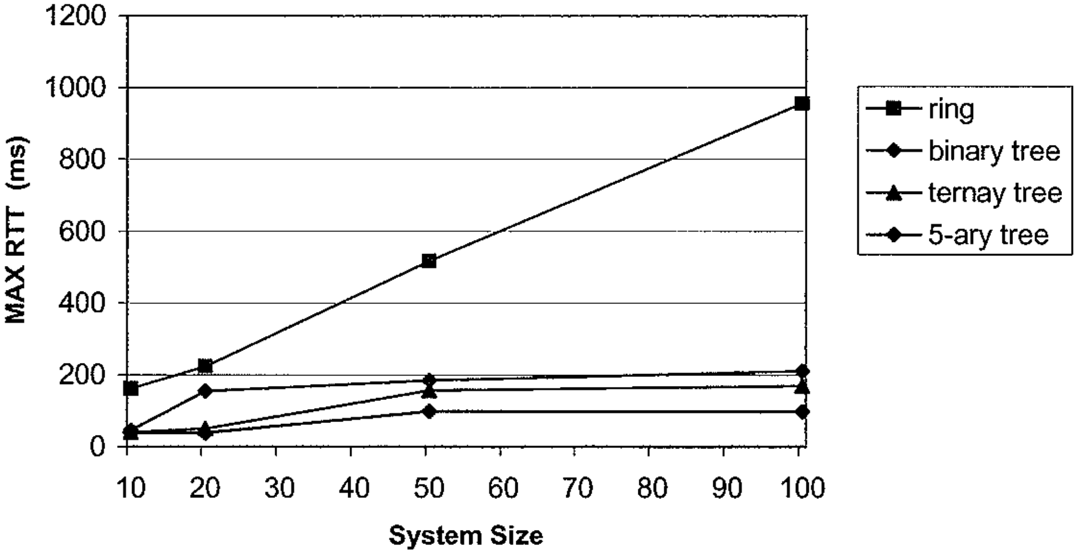

Figure 5-4 The maximum RTT with XML payload in the distributed systems

5.4 Summary

This chapter presents the test environment and results of this system. We tested two aspects of the system performance, the system's ability to maintain the given system structure, and the Round Trip Time (RTT) for messages transmitted in the system. There is no significant difference in the time used to construct different systems. The ring-structured network system is easier to recover than the tree-structured network in the case of any node fails. However the maximum RTT in the ring-structured network grows significantly with the size of the system due to the linear growth in system diameter with the system size.

6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

This thesis 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 chapters and the test results in chapter 5, we draw the following conclusions:

1. It is easier to maintain the system structure in a ring-structured network. When a node in the system fails, the time used to recover the system structure in a ring-structured network is less than that used in a tree-structured network. The average time used to recover a ring-structured network is half of that used to recover a 5-ary tree-structured network when the system has 100 nodes. When a node fails, in the ring-structured network, the master node only needs to find the two nearest neighbors that are active and connect them together while in the tree-structured network, all the child nodes of the failure node have to connect to the new parent node which brings more complexity to recover the system. In the ring-structured network, new nodes can be added to the system without considering the capacities of the nodes. However, the longest RTT grows linearly with the system size. When the messages transmitted in the system have large payloads, the longest RTT can become very large. In our tests, when there are 100 nodes in the system, the longest RTT for XML payload (252 bytes) message in the ring-structured network is 957.3ms, while in the 5-ary structured network it is 97.7 ms. 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.

2. In a tree-structured network system, the complexity of maintaining the given system structure grows as the degree of the tree. The scalability of such system is related to the node's capacity and the height of the tree. We can choose a complete n-ary tree as the system topology to limit the height of the tree. Since for a given system size (the number of nodes in the tree) and a given node's capacity (the ability to handle connection numbers, i.e., the largest degree of the nodes), among all the different types of trees the complete n-ary tree has the minimum height. 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. 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, while the time used to recover the 5-ary complete tree network is less than twice of that used for the ring-structured network. The point we can see here is that the 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.

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).

ACKNOWLEDGEMENTS

I would like to give special thanks to Dr. Brett Bode who gave me much useful advice and help during my doing this project and writing this thesis. Special thanks to Dr. Ricky Kendall and Dr. Ying Cai who helped me to finish this project and gave me advices in writing this thesis. I gained useful knowledge and programming skills for distributed systems from their classes. Also, thanks to the members of the Scalable Computing Laboratory with whom I discussed the ideas for this project. Finally, thanks to my husband, Jun Zhang, for everything he has done for me through years. His encouragement and support have helped me achieve many of my goals.

This research project is supported by United State Department of Energy. The G4 cluster used to develop and test the system is supported by the MICS office of the U.S. Department of Energy.

This work was performed at Ames Laboratory under contact No. W-7405-Eng-82 with the U.S. Department of Energy. The United States government has assigned the DOE Report number IS-T2370 to this thesis.