AN ABSTRACT OF THE THESIS OF

Krishna S. Garimella for the degree of Master of Science in Electrical and Computer Engineering presented on October 26, 1989.

Title: Performance Analysis of Network Based File Systems.

Abstract approved : Redacted for Privacy

Roy Rathja

The performance of multi-user workstations that access files over a local area network is studied. Such file systems, called distributed file systems, provide access of remote file systems with a great degree of transparency. The main objectives are

- To study the principles of distributed file systems, specifically NFS from Sun Microsystems and RFS from AT&T.
- To investigate a methodology to evaluate the performance of such file systems.
- To evaluate the performance of these network based file systems and study the effect of various implementation issues on their performance.

The major conclusions drawn are

- Allowing larger block transfers from a remote resource to clients over the network improves performance. The benefits,

however, are limited to disk block sizes of 12-16 Kbytes data. Performance can also be improved by proper layout structure on the disk.

- Enhancements of the server prove to be more cost effective than those of the client.
- Network contention is not the major bottleneck when a high speed network is being used.
- A configuration of computers sharing file systems over a LAN can have satisfactory performance in a software development environment.

# Performance Analysis of Network Based File Systems

by

Krishna S. Garimella

A THESIS

submitted to

Oregon State University

In partial fulfillment of
the requirements for the
degree of
Master of Science

Completed October 20, 1989

Commencement June 1990

APPROVED

Redacted for Privacy

Associate Professor of Electrical and Computer Engineering in charge of major

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented October 26th, 1989

Typed by Krishna Garimella for Krishna S. Garimella

# Table of Contents

**List of Figures**

# List of Tables

# Performance Analysis of Network Based File Systems

# CHAPTER1

## 1. Introduction

### 1.0 Motivation

The performance of multi-user workstations that access files over a Local Area Network (LAN) is studied. Examples of such systems are the Sun Microsystems' Network File System (NFS) **[1]**, AT&T's Remote File Sharing (RFS) **[2]** and the DOMAIN File System from Apollo Computers **[3]**. The advent of low-cost personal computers and workstations has brought about a need for information and peripheral sharing across machine boundaries. In response, researchers have developed a number of network disk servers and subsequently high speed local area network based file systems. Several configurations with a variety of performance and cost trade-offs have emerged. These include configurations in which workstations with little or no local disk storage are connected to a powerful file server with fast disks, and ones in which personal desktop computers are connected to hosts for backup and archival storage.

Network or distributed file systems typically allow multiple machines to transparently access file system resources across machine boundaries. Distributed operating systems extend the network transparency to resources other than files, such as processes and virtual memory. To effectively design, configure and administer network based file systems, one must be able to understand and quantify their performance. The objective is, thus, to assess the penalty on performance due to remote file accesses and explore a number of design alternatives that can minimize the penalty.

## 1.1 Objectives

The objectives of this dissertation can be summarized as:

- To study the underlying issues of distributed file systems.
- To investigate a methodology to evaluate the performance of such file systems.
- To evaluate the performance of popular network based file systems and investigate the effect of various implementation issues on their performance.

## 1.2 Overview of the dissertation

In Chapter 2 an introduction to distributed file systems from different vendors is given and the underlying differences in design and operation are outlined. The approach taken for the performance analysis builds queuing network performance models from the measurement data. This is explained in detail in Chapter 3. A summary of results, comparisons and conclusions are given in Chapter 4, explained in light of earlier chapters.

Chapter 5 lists the major conclusions that can be drawn from the performance measurements. It also describes some future work that can be done in the area of performance analysis.

Appendix A gives a description of the Business Benchmark and the Multi Vendor Test Suite (MVTS) used to analyze RFS and NFS respectively. Also included are the "raw" test results for RFS and NFS. Appendix B looks at an analysis of the performance of different vendor's Remote Procedure Calls (RPCs), which are the building blocks of distributed file systems. The results are a summary of tests done by Joshua Levy of Atherton Corporation [4] .

## 1.3 Conclusions

The major conclusions of this study can be summarized as follows

- Allowing volume transfers from the remote resource to the client over the network improves performance. However these benefits are limited to moderate disk block sizes of 12-16 Kbytes. Volume transfers can be made more efficient by having a better page layout strategy on disks, such that fast and contiguous access of pages is possible.
- Enhancements to the server prove to be more cost effective as improvements on the client end pertain to only one client.
- Network contention is not the major bottleneck when a high speed physical link ( e.g. Ethernet) is used.

- Workstations that share file systems over a local area network can have satisfactory performance. A fairly good performance can be achieved even if the file systems of hosts are entirely remote and all the file accesses are over the network (as in the case of diskless systems).

# CHAPTER 2

# 2. On Networks and File Systems

## 2.0 Introduction

This chapter provides a brief overview of underlying networking principles and an outline of different distributed file systems, their implementations and fundamental differences.

## 2.1 Review of underlying network concepts

The implementation of modern computer networks is done in a highly structured way. The following sections briefly describe the structure.

### 2.1.1 Protocol Hierarchies

To reduce design complexity, most networks are organized as a series of *layers* or levels, each built upon its predecessor. The number, the name, and the function of each layer differs from network to network. However, in all networks, the purpose of each layer is to offer certain services to the higher layer, shielding those layers from the details of how the offered services are actually implemented. Layer *n* on one machine carries on a conversation with layer *n* on another machine. The entities comprising the corresponding layers on different machines are called *peer processes.* In reality no data is transferred directly between two *peer processes* (except the lowest layers). Instead, each layer passes data and control information to the layer immediately below it, until the lowest layer is reached. At the lowest layer there is physical communication with the other machine as opposed to the virtual communication used by higher layers.

### 2.1.2 The ISO-OSI Model

The International Standards Organization (ISO) proposed a seven layer networking model as the first step toward standardization of various protocols. The reference model of the Open Systems Interconnection (OSI) was described by Zimmermann in 1980 [5]. The major components of the ISO-OSI and ARPANET models are outlined in Figure 2.1.

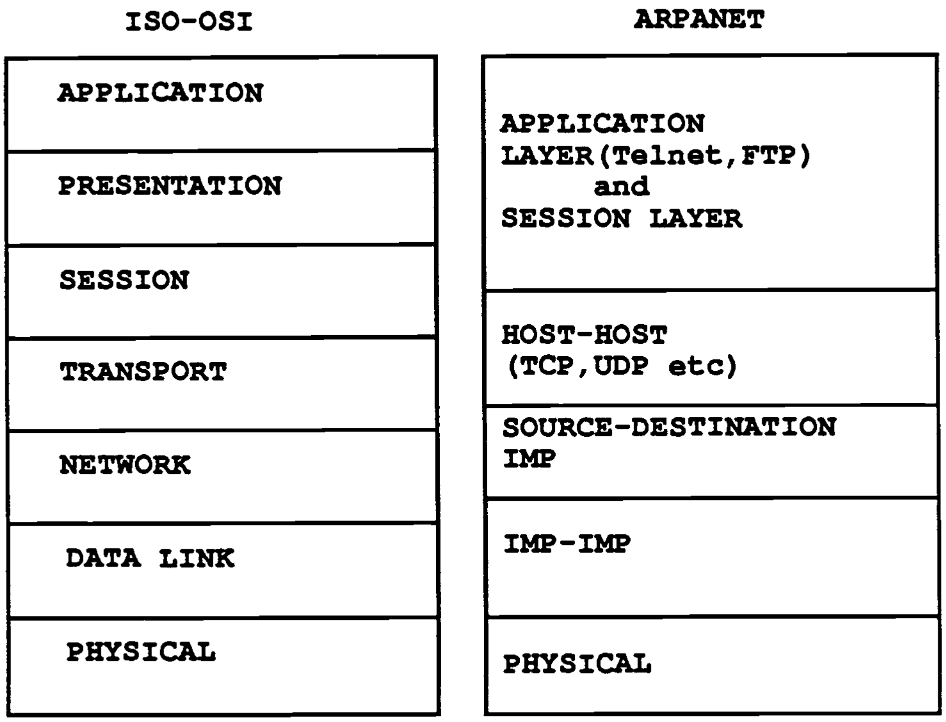

FIGURE 2.1: OSI Layering and Correspondence With ARPANET

The following paragraphs briefly describe the seven layers of the ISO-OSI model.

### The Physical Layer

The physical layer is concerned with transmitting raw bits over a communication channel. This layer is concerned with transmission of an unstructured bit stream over the physical link. It also deals with such parameters as signal voltage swing and bit duration. The design issues here largely deal with mechanical, electrical and procedural interfacing to the subnet.

### The Data Link Layer

The data link layer does the accounting and traffic control chores needed to transfer information on an electrical link. It puts every piece of information into the right place and checks it before putting it on the physical link. The data link was originally thought of as being a function of the software in a device. It is increasingly being done by special-purpose integrated circuits requiring little external programming.

### The Network Layer

The network layer sets up a logical transmission path through a switched network. This is sometimes called the *communication subnet layer*. In local networks the transmission path may only be theoretical, since the individual units are almost always electrically connected to the network. But in large systems, several transmission paths (or *routes*) and even alternative media (dial-up telephone lines, satellite links etc.) may exist. The key design issue is how the route is determined. It could be based on tables that are static and are rarely changed. It could also be based on dynamic routing tables, where the dynamism is determined by different routing algorithms.

## The Transport Layer

The basic function of the transport layer is to accept data from the session layer, split it into smaller units, if need be, and pass it on to the network layer. It translates whatever unique requirements the other higher layers might have, into something the network can understand. There are error detection and correction facilities and also provisions for expedited delivery of high priority messages. This layer also attempts to re-establish lost connections due to network failure. This layer is a true source-to-destination or **end-to-end** layer. The transport layer establishes connection between the "peer" processes at each end of the communication system. This part is often implemented by the host operating system.

## The Session Layer

The session layer is a coordinating function. It establishes a communications link between units and gradually feeds or buffers information to the devices or program performing the presentation function. Ignoring the presentation layer, which merely performs certain transformations, the session layer is the user's interface to the network. In some networks, the session and the transport layers are merged into a single layer, or the session layer is absent altogether, if all that the user wants is raw communication service.

### The Presentation Layer

The next layer in the model is the presentation layer. This layer prepares the information for the application. Services that this layer would typically provide include data translation, formatting and syntax selection. Transformations like text compression and encryption are also done in this layer.

### The Application Layer

The content of the application layer is dictated by the user. This includes applications that are to be run in a distributed environments, vendor provided utilities like electronic mail, and file transfer utilities. Human user interfaces, like the X-Window system, also are a part of this layer.

## 2.2 File Systems

Most applications of computers use files for storage of information. The file is an abstraction of a storage construct that is based on magnetic disks or other secondary storage media. Conventional operating systems include a file system that is concerned with organization, storing, retrieval, naming, sharing and protection of files. File systems are designed to allow programs to use a set of operations that characterize the file abstraction and free them from concerns about details of disk storage and layout. At their simplest, files are defined as sequences of similar data items and file systems provide functions to read and write sub-sequences of data beginning at any point in a file.

A distributed file system extends the file system beyond the machine boundaries. A distributed file system is an essential component of a distributed system; it is needed to support shared information, to enable users to access files without copying them to their local workstation disk and if diskless workstations are used it is needed for all permanent storage. In the following sections we look, in detail, at two popular vendor supplied distributed file systems; NFS from Sun Microsystems and RFS from AT&T

## 2.3 NFS: Network File System

### 2.3.0 Introduction

The Sun Network File System (NFS) protocol provides transparent remote access to shared files across networks. The NFS protocol is designed to be portable across different machines, operating systems, network architectures, and transport protocols. This portability is achieved through the use of Remote Procedure Call (RPC) primitives built on top of an eXternal Data Representation(XDR).

implementations already exist for a variety of machines, from personal computers to supercomputers.

### 2.3.1 Remote Procedure Call

Sun's Remote Procedure Call (RPC) specification provides a procedure-oriented interface to remote services. Each server supplies a "program" that is a set of procedures. NFS is one such program. The combination of host address, program number, and procedure number specifies one remote procedure. A goal of NFS was to not require any specific level of reliability from its lower levels, so it could potentially be used on many underlying transport protocols, or even another remote procedure call implementation. For ease of discussion, the rest of the description will assume that NFS is implemented on top of Sun RPC [7].

### 2.3.2 External Data Representation

The eXternal Data Representation (XDR) standard [8] provides a common way of representing a set of data types over a network. Although automated RPC/XDR [9] compilers exist to generate server and client "stubs", NFS does not require their use. Any software that provides equivalent functionality can be used, and if the encoding is exactly the same it can interoperate with other implementations of NFS.

### 2.3.3 The RPC, XDR and the ISO-OSI Model:

The RPC and XDR reside in the ISO-OSI model as shown in Figure 2.2. The actual layering is complex, as RPC does not fit into the model especially well. It has been designed to be fast, and therefore does not contain a multi-layer structure. It can logically be considered to be equivalent to the session layer of the OSI layer.

| | |
|---|---|
| Application | Mount |
| Presentation | XDR |
| Session | RPC |
| Transport | TCP, UDP |
| Network | IP |
| Data-Link | ETHERNET |
| Physical | |

**FIGURE 2.2: RPC, XDR and the ISO-OSI model**

### 2.3.4 Stateless Servers

The NFS protocol was intended to be as stateless as possible. That is, a server should not need to maintain any protocol state information about any of its clients in order to function correctly.

Stateless servers have a distinct advantage over servers that maintain state, also known as "stateful" servers, in the event of a failure. With stateless servers, a client need only retry a request until the server responds; it does not even need to know that the server has crashed, or the network temporarily went down. The client of a stateful server, on the other hand, needs to either detect a server failure and rebuild the server's state when it comes back up, or cause client operations to fail.

This may not sound like an important issue, but it affects the protocol in some unexpected ways. Note that even if a reliable transport protocol such as TCP is used, the client must still be able to handle interruptions of service by re-opening connections when they time out. Thus, a stateless protocol may actually simplify the implementation.

On the other hand, NFS deals with objects such as files and directories that inherently have state. The file will be of no use if its contents are not kept intact. Inherently stateful operations such as file or record locking, and remote execution, were implemented as separate services.

### 2.3.5 File System Model

NFS assumes a hierarchical file system similar to UNIX. Each entry in a directory (file, directory, device, etc.) has a string name.

Different operating systems have different ways of naming the files or different syntax to represent the "pathname". A "file system" is a tree on a single server (usually a single disk or physical partition) with a specified "root". Some operating systems provide a "mount" operation to make all file systems appear as a single tree, while others maintain a "forest" of file systems. Version 3 of NFS uses a slightly more general file system model **[1]**.

Although files and directories are similar objects in many ways, different procedures are used to read directories and files. This requires a network standard format for representing directories. The same argument as above could have been used to justify a procedure that returns only one directory entry per call. The problem is efficiency. Directories can contain many entries, and a remote call to return each would be too slow.

### 2.3.6 NFS Implementation Issues

The NFS protocol was designed to allow different operating systems to share files. However, since it was designed in a UNIX environment, many operations have semantics similar to the operations of the UNIX file system. This section discusses some of the implementation specific details and semantic issues.

Server/Client Relationship

The Remote Procedure Call is implemented using the client-server Model. The client-server model involves a set of processes called *clients* which communicate with a process called the *server.* The server process can reside either on the local host or on a remote host accessible through the network. The clients send a request to the server and the reply is a "service" to the client.

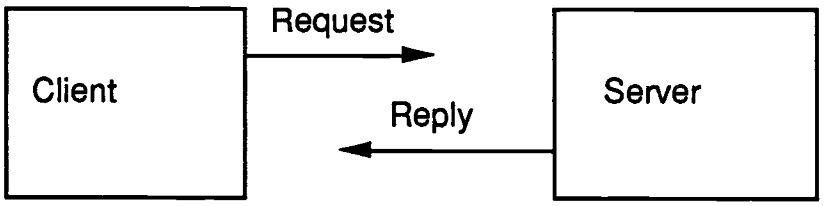

**FIGURE 2.3: The Client - Server Model**

In the RPC school of thought, a client sending a message to the server and getting a reply is like a program calling a procedure and getting a result. As in the case of the client-server model the caller initiates an action and waits until it is completed and the results are available. The procedure call to a remote machine is transparent to the user. Thus the caller need not be aware of the difference between a local procedure call and a RPC. To help hide the differences between local and remote calls, a set of library calls can be provided to programmers.

## **2.3.7 Implementation of RPC**

As we have seen, a set of "library" calls can be provided on the client to hide the details of the network from the user. These library procedures are called *stubs.* Stub procedures not only transfer data between the client and the server but also send messages. The following example gives an idea about the steps of a remote call. Referring to Figure 2.4, in Step 1 the client program calls the stub procedure. Parameters are passed in the usual way as between a calling program and a procedure. The client does not notice any difference between local and remote calls. In Step 2, the client stub collects the parameters and encapsulates them to be sent over to the server end. This is known as parameter marshalling. After this the message is handed over to the transport layer. Step 3 consists of the transport layer passing over the packet to the server end by transmission. Next, in Step 4, the transport layer of the server collects the messages and passes it on to the server stub. The server stub "un-marshalls" the packet in Step 5, and passes it on to the server. The server then executes the instruction and sends back the reply. The server stub then marshalls the reply(Step 7) and passes it on to the server's transport layer to be transmitted back to the client(Step 8). The purpose of the whole operation is to give the client procedure, transparency of the underlying details of getting a reply from a remote server.

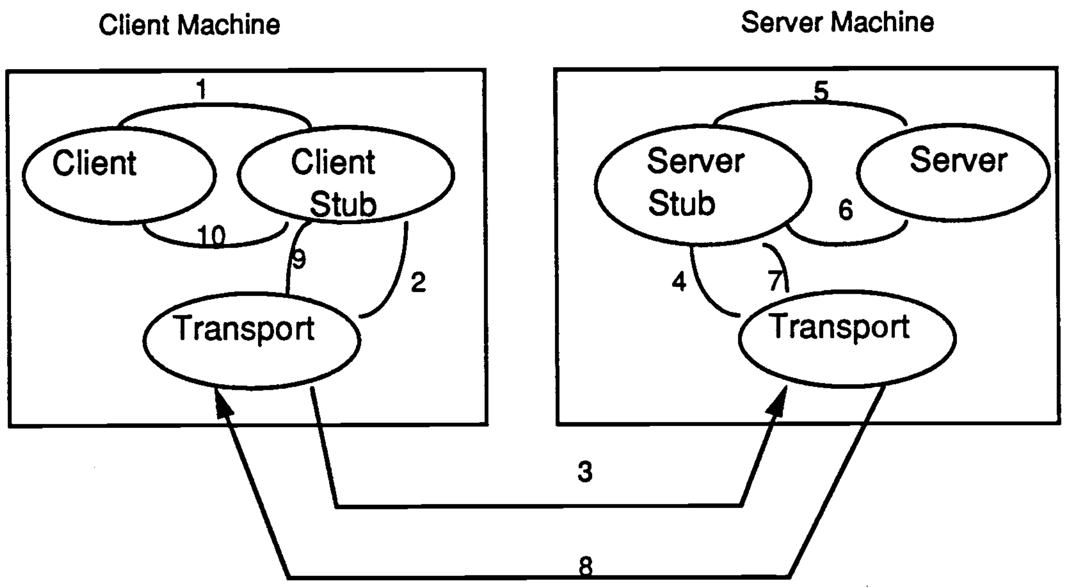

**FIGURE 2.4 Ten steps to execute an RPC**

One may wonder how a client comes to know which server to call for a particular protocol. A scheme due to Birrel and Nelson [9] includes not only the client and the server but also a specialized database system. In their method, when a server is booted, it registers with the data base system by sending a message containing its name, its network address and a unique identifier. When the client makes its first call to the server, its stub is faced with the problem of locating the server. To do this, the server stub sends its name to the database system. The database returns a unique identifier and the network address of the server. This is called *binding* . The unique identifier distinguishes the server from other servers on the network.

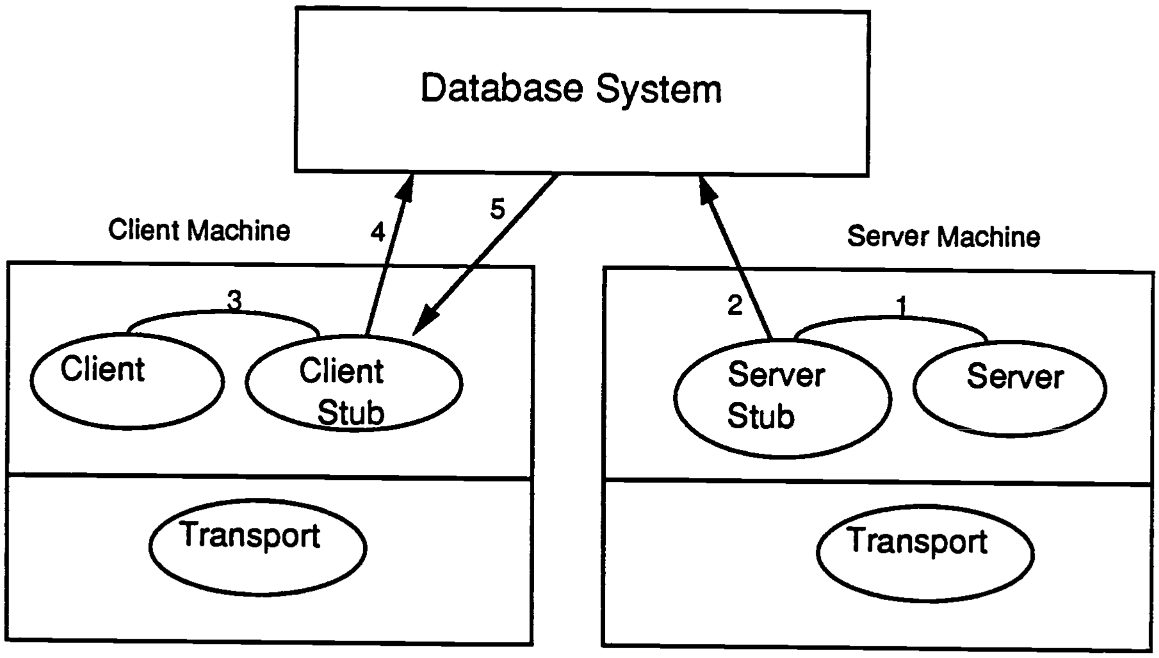

**FIGURE 2.5 Client - Server binding done via a database**

Thus we see that RPC provides the following services.

- It allows users to initiate a procedure on a remote machine so that the procedure appears to execute on the local machine in the user's address space. RPC does this by interfacing with the network layers below it.
- RPC provides a message format that interacts with and specifies a high level protocol. RPC allows users to obtain necessary data on the remote server without the user being aware of it.
- RPC provides a library of function calls in a high level language for local and inter process communications.

The generality and simplicity incorporated in implementing NFS servers is not without drawbacks. It is impossible to implement some complicated file system semantics, like removal of open files, on a stateless system. These problems are solved by doing some tricks with the file semantics [1].

## 2.4 RFS: Remote File Sharing

### 2.4.0 Introduction

This section describes the implementation of AT&T's Remote File Sharing (RFS). RFS is a distributed file system provided with UNIX System V, Release 3 that allows a network of machines to transparently share files [2]. A process running on one machine can access a remote file in the same manner as if it were local. To accomplish this transparency RFS meshes with the existing UNIX file naming semantics. Unlike other distributed file systems RFS preserves the full UNIX file system facilities and allows access to all types of files including special devices and named pipes. In addition RFS extends file and record locking to remote files. The following sections describe the implementation issues on which RFS is based and the architecture used to achieve the goals.

### 2.4.1 RFS Goals

The RFS was implemented with the following goals in mind, as outlined by Andrew Rifkin et al [2]:

- Transparent Access: To preserve the standard UNIX interface and make the access of remote and local files identical, except for a difference in performance.
- Semantics: To preserve the UNIX semantics by providing access to special devices and special files regardless of them being local or remote.
- Binary Compatibility: To provide binary compatibility for existing applications with a new environment with network resources.
- Network Independence: To provide compatibility with different technologies, RFS should be implemented with network independence.
- Performance: High performance should be achieved, by minimizing network access.

## 2.4.2 RFS Architecture

The RFS architecture is based on the client-server model which has been described in Section 2.3.6. Once connected the machines communicate using a message protocol based on the UNIX system call interface. The *streams* mechanism [11] is used in conjunction with the transport interface defined in System V to separate RFS from the underlying network, making RFS network independent. By defining an RFS file system type, RFS is cleanly integrated into the UNIX kernel using the File System Switch (FSS) mechanism in System V Release 3 (SVR3). Also, to ensure security, the normal UNIX file protection is extended to remote files and a mechanism is provided to map user id's.

### Client-Server

A file sharing relationship consists of two machines. Like in NFS, the file physically resides on the server machine, while the client machine remotely accesses the file. The client accesses the file by sending a request message to the server. The server's response is to provide the requested resource.

### Connection Establishment

The RFS connection establishment involves locating and identifying a remote resource followed by remotely mounting it. The location and identification of resources is done using the RFS name server. The remote mount adds the remote file system to the local file system.

### RFS Message Protocol

The RFS message protocol is based on UNIX system calls, which are well defined and accepted. This protocol is used to communicate remote resource requests between client and server machines. For each system call there exists a request and response message. The request message formats all pertinent information necessary to execute the system call, while the response message formats all possible results.

### Network Independence

By separating RFS from the underlying transport service, RFS is able to run over a variety of protocols and networks without modification. To accomplish this, two problems had to be solved. First it was necessary to choose the right type of transport service. Second, a standard interface between RFS and the transport service had to be provided.

Unlike NFS, RFS was designed to work over large concatenated networks and since the overhead of retransmission and error detection is high for datagram service over large networks, reliable virtual circuit service was chosen. To compensate for virtual circuit setup costs, a single virtual circuit is maintained between a client and a server machine. This circuit is established during the first remote mount, and all subsequent mounts are multiplexed over this circuit. The circuit is held open for the duration of the mounts.

The second problem was solved using the transport interface (based on the ISO Transport Service Definition) defined within the System V networking framework and the *streams* mechanism. By adhering to the transport interface, a connection between RFS and a transport service is provided through the *streams* interface as illustrated in Figure 2.6.

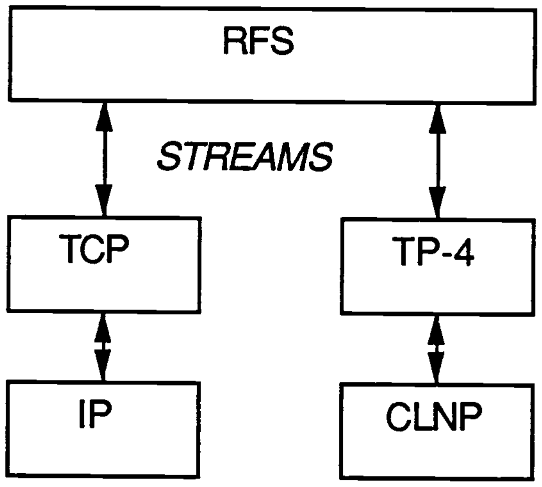

**FIGURE 2.6: STREAMS Based Communication**

## 2.5 RFS Implementation and Differences with NFS

At the application level, RFS functionality is quite similar to that provided by NFS. However, the underlying design and implementation of RFS and NFS are quite different.

### 2.5.1 System Call Vs File System

One of the principal differences between RFS and NFS is in the abstractions used as models for their design and implementation. RFS uses a remote system call abstraction whereas NFS uses a remote file system abstraction.

## The System Call Abstraction

The system call abstraction uses system calls as the interface by which one manipulates file system objects. There are a number of important points to note about this abstraction.

- One argument to the system call is always a reference or a handle to a file. This reference takes the form of either a pathname or a file descriptor.
- A pathname relies on the uniform file system name space presented by UNIX file system semantics. The resolution of the pathname across and within the file systems is below the level of abstraction presented by the system call interface to the user. Also the relation between pathnames and files themselves is "many to one", i.e., there can be several names for the same file.
- A file descriptor is returned by a system call and represents a reference to a file which is guaranteed by UNIX semantics to exist until the descriptor is closed.

In contrast to the System Call Abstraction, the file system abstraction typically provides a somewhat lower level of operations on files and file systems.

## 2.5.2 Stateless Vs Stateful Systems

The second notable point concerning the differences between RFS and NFS is that, whereas NFS does not maintain server state on behalf of a client, RFS does.

The NFS server is designed to retain no memory of its clients activities; it is "stateless". The primary reason for statelessness is to make recovery after a server crash easy; if the server has no state, then there is no state to restore. As far as the client is concerned the server has slowed down infinitely and will eventually respond. Thus the client keeps sending requests.

The RFS server, however keeps a record of the client state information. Thus a special recovery mechanism has been designed to erase this state in the event of a crash on the client side. The main purpose of the recovery mechanism is to restore state on the server machine in the event of a client crash, and to cleanup a client machine in the event of a server crash. The statefulness assures that data integrity and consistency is maintained in case of system crashes and network failures.

# CHAPTER 3

## 3. Performance Models and Configurations

### 3.0 Introduction

This chapter introduces a Queuing Network Performance Model which can be be used to analyze performance measurement data. No mathematical analysis of the model is attempted in this dissertation. The purpose of presenting this model is to get familiar with service centers and customer demands in a distributed file system. These models can be used as a basis for developing simulation tools for distributed environments as described in Chapter 5. Later in this chapter the configuration used for gathering the measurement data is described.

### 3.1 Queuing Network Models

The input requirements of models dictate the quantities that must be measured. The queuing model can be used for analysis because it posses an attractive combination of efficiency and accuracy. In recent years queuing models have become a tool of choice in computer system design and analysis of applications. There are three components to the specification of a queuing network model: the *service center description*, the *customer description*, and the *service demands*.

- The service center description identifies the resources of the system that will be represented in the model: disks, CPUs, communication networks, etc.
- The customer description indicates the workload intensity: the average number of requests in the system, or the average rate at which requests arrive to the system, or the number of users and the average time that a user pauses between receipt of a response and initiation of another request.
- The service demands describe the average amount of service that each request requires at each service center.

Once these inputs have been specified, the model can be evaluated using efficient numerical algorithms, yielding performance measures such as utilization, residence times, queue lengths and throughputs. The model can also be used to project the performance of the system under various modifications.

### 3.1.1 The Canonical System

Figure 3.1 represents the "canonical system" studied in this paper. The tests had been run on a configuration similar to the one shown in Figure 3.1. The configuration consisted of 4 Intel - 80386 based workstations, running Unix System V. They were connected by a 10 Mbps ethernet. Figure 3.2 illustrates the canonical model used in our study. The file server is represented by two service centers, corresponding to CPU and the disk. The network is represented by a single load dependent service centre, since the efficiency of an Ethernet depends on the number of workstations simultaneously attempting to assert data. Each client is represented by a single service station corresponding to its CPU. The local disk is not a service station when remote files are accessed.

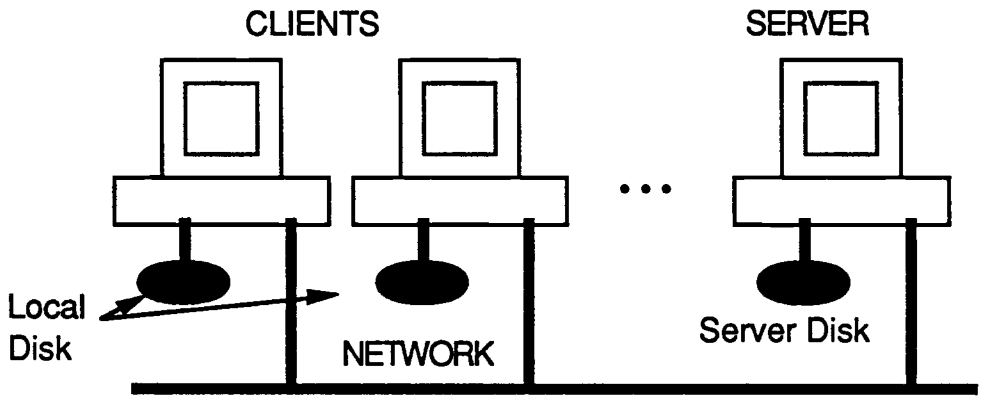

**FIGURE 3.1: Canonical System**

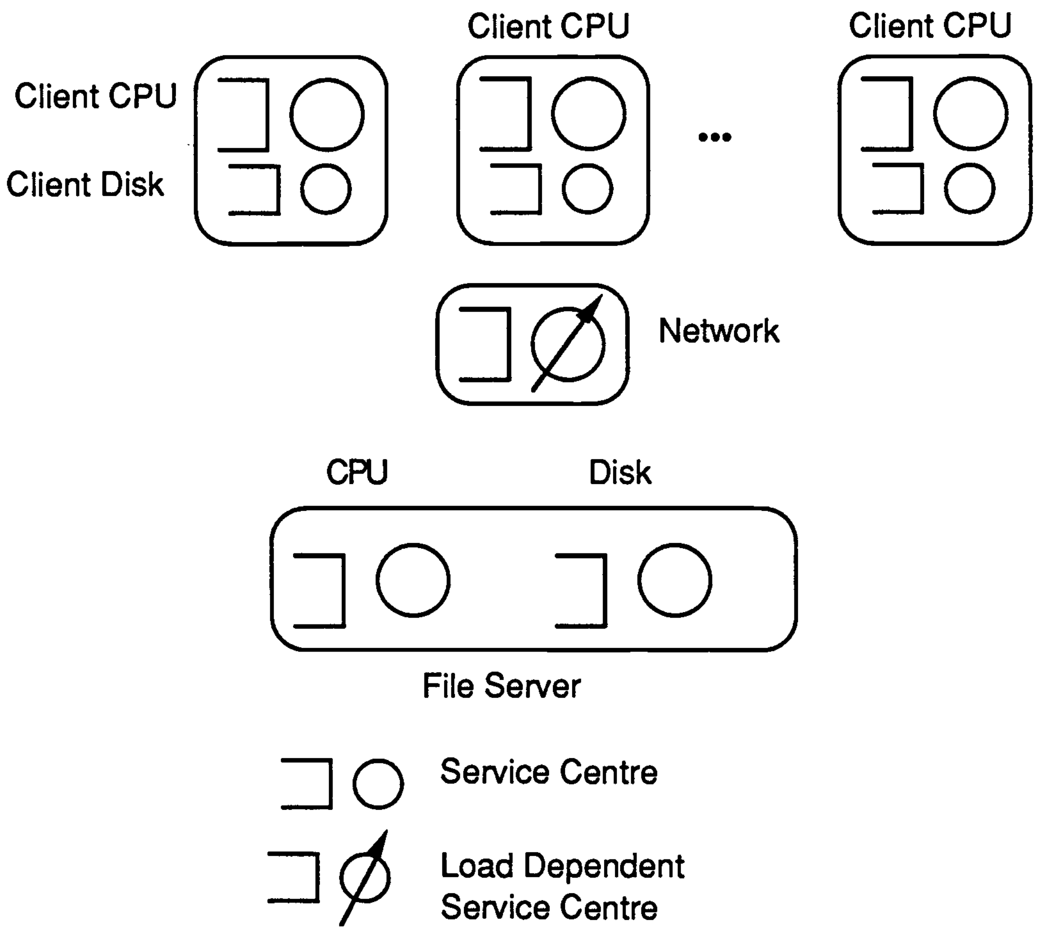

**FIGURE 3.2: The Canonical Model**

## 3.2 Benchmarks and Configuration

### 3.2.1 Benchmarking RFS

The Business Benchmark from Neal-Nelson Associates was used to study RFS. The benchmark is described in Appendix A. This is an industry standard benchmark program provided for performance analysis of different system configurations. The configuration consisted of 4 - Intel 80386 based workstations running Unix System

V. Each had a main memory of 8 Megabytes and were running at a clock speed of 20 MHz. They were also running TCP/IP and were connected to a Ethernet based LAN. One of the machines was configured as a server of RFS. The Business Benchmark was located on the server's file system. The server's directory, containing the benchmark, was remotely mounted on the client and the program was run over the network. The tests were also run on the stand alone systems and the local and remote test results were compared. The results are presented in Appendix A.

### 3.2.2 Benchmarking NFS

For NFS the Neal-Nelson's tests could not be used for benchmarking, because NFS does not allow remote mounting of special files, specifically remote disks. The setup of Neal-Nelson tests required such a mount and hence they could not be used. Instead, a special test suite, called the Multi Vendor Test Suite (MVTS) was used to determine the times for remote and local read/writes. To find the effect of block size on the performance, the program "ioperf" run on NFS. The raw results are presented in Appendix A.

### 3.2.3 Results Obtained from Literature

Because of access to better resources, E. D. Lazowska, et al, [12] have been able to run exhaustive tests involving measurements to determine the effect of a large number of workstations on the network. The results also show the effect of different parameters like network packet size and file cache size on performance. Their results in conjunction with the benchmark results are used to analyze the performance of the file systems. Lazowska et al [12] ran tests on configurations based on similar models on Remote File Systems by SUN Microsystems (NFS), Apollo (DOMAIN) and V - Kernel systems from Stanford University.

RPCs from different vendors, due to the inherent implementation differences, influence the performance of the file systems implemented on top of them. A study of performance of different RPCs has been done by Joshua Levy [4] as outlined in Appendix B.

# CHAPTER 4

# 4. Analysis of Results

## 4.0 Introduction

This chapter presents results obtained from various measurements and from previous work done in this area. The results are presented along with some design alternatives which might improve the performance of distributed file systems. The effect of issues governing performance, like congestion due to increased network usage, is also studied. Implementation issues like disk block size, network packet size and file block caching are also studied.

## 4.1 The Effect of Congestion

Unlike a dedicated disk on a single workstation, the file server and local area network are shared facilities, and access to them is subject to delay owing to competition from other workstations. In other words, the relative response time perceived by a workstation user in remote file system environment can be thought of as having three components:

- User mode processing. This is the time spent in computation by the application program.
- Basic I/O time. This is lower for the local than for remote access in a configuration where identical workstations are used on the network.

- Queuing Delay. This is due to congestion in remote access and this grows with number of workstations sharing the file server. This delay increases the penalty of remote access.

The queuing delays evident in Figure 4.1 are due to the congestion at the file server. Figure 4.2 shows the utilization of the server CPU, the disk and the network. Several points can be noted from Figure 4.1 and Figure 4.2.

- At light loads (i.e. small number of workstations), a file server design that minimizes file access latency in the absence of congestion is of some value.
- The performance at higher loads is governed by the most heavily used resource, which limits the throughput. This would be the file server CPU in this case.
- Ethernet can accommodate a large number of file servers because of its high bandwidth and network contention is not much of a problem.

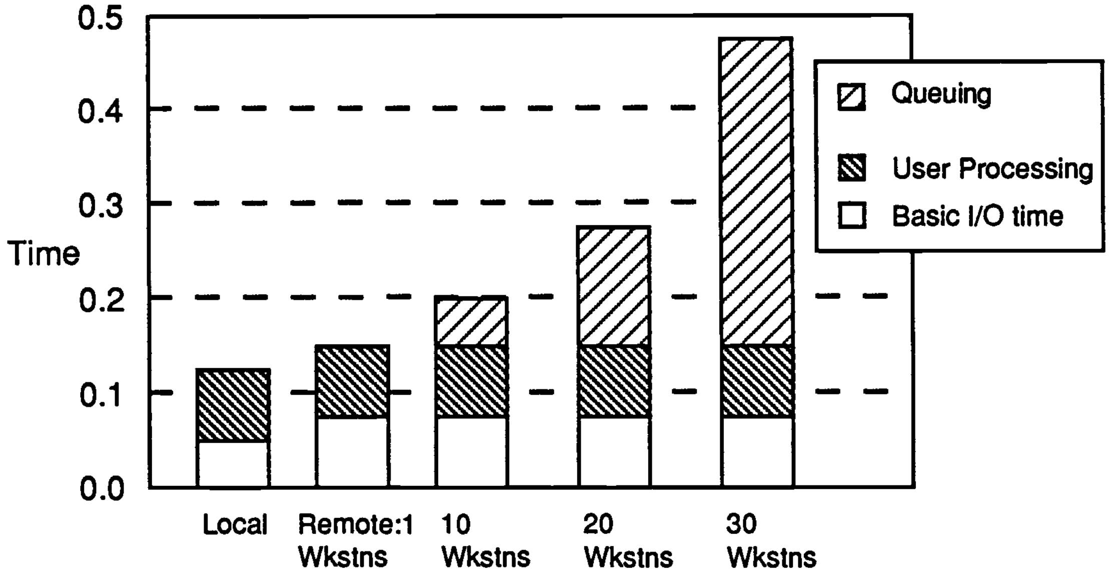

**FIGURE 4.1: The Effect of Congestion (Source: [12] )**

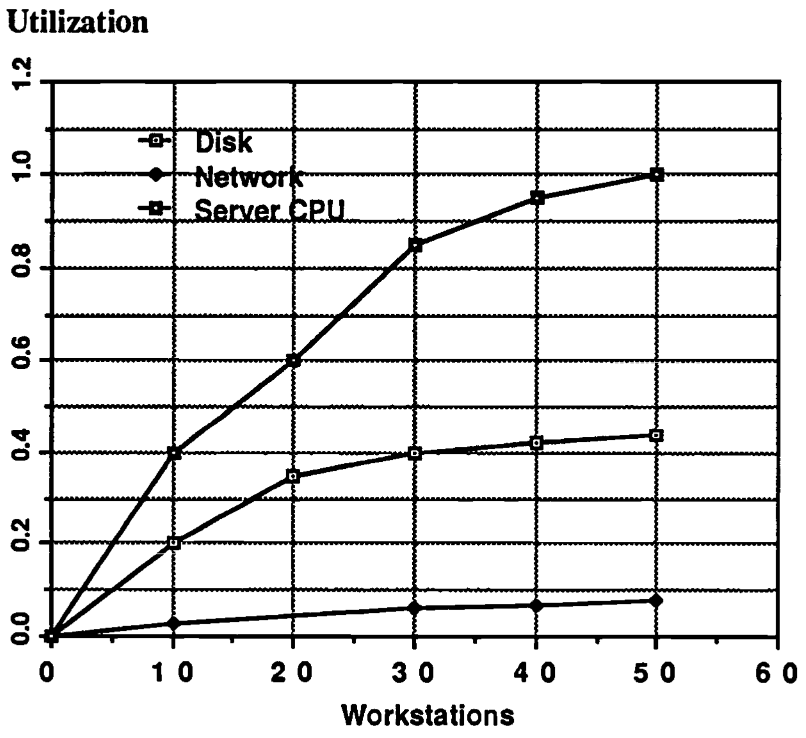

**FIGURE 4.2: CPU, Disk and Network Utilization (Source: [12])**

## 4.2 Design Alternatives

In this section, measurements are used to investigate a number of design alternatives intended to improve the performance. As noted above light-load performance can be improved by reducing the time spent for the service at any resource (e.g. A reduced file access latency) or by improving the concurrency of processing the requests. However at high loads the increase in concurrency is difficult, so the performance is most improved only by reducing the service demand at the most heavily utilized device. As can be seen from Figures 4.1 & 4.2 the bottleneck seems to be the server CPU. The design alternatives must, therefore, aim at reducing the service demands on the server CPU.

### 4.2.1 Increased Disk Block Size

The program "ioperf" was written to conduct experiments on the effect of different disk block sizes. This was a part of the MVTS test described in Appendix A. For this test a SUN Sparcstation was configured as an NFS server and an Altos 1000 (Based on Intel-80386) as a client. There was no other network activity and no more user load on the machines. The "time" command from the "csh" was used to get the timings. A 100 Kbytes file was used and the times per kilobyte of transfer were calculated. The raw results are presented in Table A.1.

The disk access time per kilobyte of data transferred is presented in Figure 4.3. The reduction in access times can be explained as follows. File access in UNIX is accomplished in blocks of 4 K Bytes. Consider an increase in the disk block size from the default value of 4 Kbytes to 8 Kbytes. This change will have two primary effects: The effective disk access time is reduced since seek and latency times are "amortized" over 8 Kbytes rather than over 4 Kbytes. The server CPU time is also reduced since only one disk I/O operation and one user request packet are required per 8k transferred as opposed to two of each operation for a block size of 4 Kbytes.

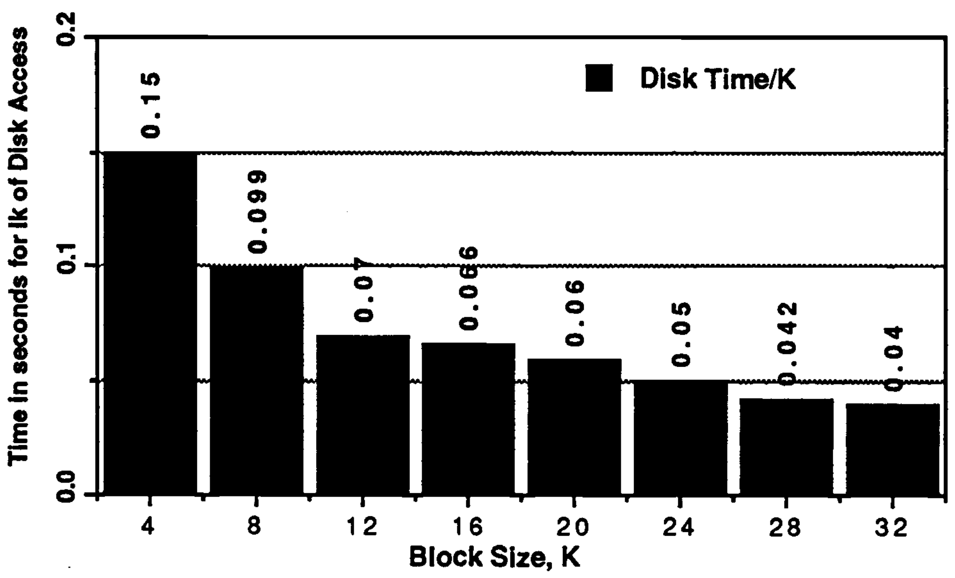

**FIGURE 4.3: Effect Of Block Size on The Remote Disk Access time**

## 4.2.2 Increased LAN Packet Size

It is clear from Figure 4.3 that increasing the block size can be extremely effective in improving performance. However, the increase in the block size actually increases the number of data carrying messages passed between the server and client. If the amount of data that is carried in one message is increased, then the fragmentation required will be less and so will be the overhead for such an operation. Lazowska et al [12] investigated the effect of increasing the packet size to 4 Kbytes from the standard 1 Kbytes. The modification improved the response times as seen in Figure 4.4. The baseline system is nothing but a system with standard network packet size of 1 Kbytes and 4 Kbytes block size.

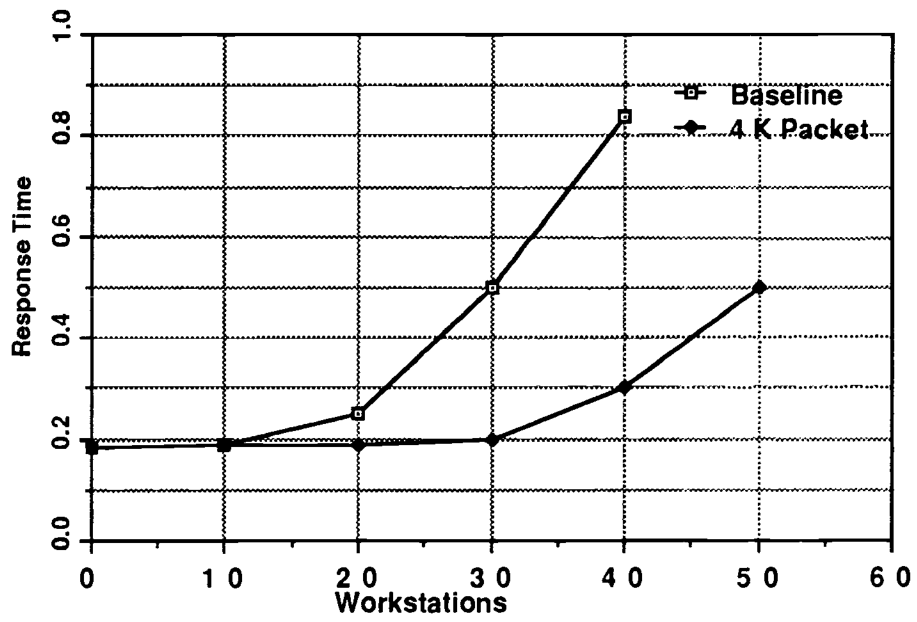

**FIGURE 4.4: Effect of Increased Network Packet Size. (Source: [12])**

**4.2.3 Limitations of Amortization**

The advantages of amortization of overheads like the disk access and network overheads, over large volumes of data, are quite conclusive. There are, however, several limitations of amortization.

- If the blocks are very large, then the network will be busy for long "bursts" of data which will be a limitation when many clients are on the network.
- For applications where the processing cost depends on the amount of data processed, the marginal benefit of amortization decreases.
- The access time will no longer decrease, when the block size exceeds the file size. The average file size in a software development environment can be assumed to be approximately 10-15 KBytes. This figure, however, varies depending on the type of application that's being run on the systems.

**4.2.4 File Block Caching**

In a caching scheme, file blocks are saved in main memory so that subsequent requests for these blocks can be satisfied without a disk operation. The success of caching depends on the principle of locality, or on the cache hit ratio. There are two possible configurations for caching in network based file systems **[12]**. Caching on the server end can reduce the demand on server CPU but client end caching will also reduce the network performance overhead.

However additional main memory is generally required for client caching and thus would require more resources as each client will need to have an independent cache. There were no measurements done to study the effect of caching on the file systems. File cache performance is considered in considerably great detail in [12].

## 4.3 ENHANCEMENT OF THE CONFIGURATION

This section briefly describes the possibility of improving performance by enhancement of the configuration. Instead of reducing the amount of work done, the work can be spread among more devices.

The most obvious solution is to acquire a second file server to share the load imposed by the client workstations. Because the effect of this is to halve the service demand at the bottleneck, performance at high loads is similar to that of client caching, as about 50% cache hit ratio can be expected in the cache scheme [12]. The other alternative is to replace the file server CPU with a much faster CPU and have a lower service demand. Lazowska et al claim to have gotten the expected performance gains after making the above two enhancements [12].

# CHAPTER 5

# Conclusions

## 5.0 Introduction

Economic and administrative considerations have pushed the development of technologies for distributed file systems quite rapidly. The measurement data presented in this dissertation is used to study the factors governing the performance of such systems. In the following section, major conclusions are described. Work that can be done in this area in the future will also be discussed.

## 5.1 Conclusions of the Dissertation

The following conclusions can be drawn from the observations made so far.

- Allowing volume transfers, or "amortization" of overhead over a large number of bytes is a definite performance improver. The improvements tend to be smaller for higher block sizes. This is because large block sizes result in increased network contention, particularly in a CSMA/CD network. Furthermore the performance benefit decreases as the amortized volume of data exceeds the actual file size. The benefits are thus limited to moderate sizes of 12-16 Kbytes of data. Better efficiency can also be gained by having file layouts which ease the access. Large transfer units might require large buffering capability. This can be avoided by either having large page sizes or highly contiguous allocation of small pages. The fragmentation due to large pages can be avoided by using the latter method, which is quite efficient due to the layout strategy.

- Enhancements on the server side may prove to be more cost-effective as the improvements on the client side pertain to only one client. A faster server means a faster service to all the clients. The enhancements on the client end will, however, reduce the network overhead. The client will use the data stored in the local cache without actually needing to fetch the data over the network and thus will save on the network overhead.

- A system of workstations sharing file systems over a local area network can have satisfactory performance. As with any shared facility, a good design is necessary to minimize queuing delays under high loads. As the load increases, queuing delays cause performance to degrade. With a well designed file server and client/server interface, the cost of remote file access is reasonable even for a substantial number of client workstations. Even if the entire file system is located on the server and there is no local secondary storage, as in case of diskless workstations, the performance remains satisfactory [12]. In fact diskless workstations provide an economical alternative to expensive local disk storage capacity at the cost of tolerable performance degradation.

- Network contention is not a major bottleneck when a high speed network (e.g. Ethernet) is used as even achieving an average utilization of 50% is an incredibly large amount of data being transferred.

## 5.2 Future Research

Future work in performance analysis should investigate the possibilities of creating simulation tools for distributed file systems. These simulation tools should assume the models presented in this dissertation. The tools should provide good user interface for ease of specifying service centers, customer description and the service demands such that the researcher can reconfigure the system under study. This will expedite the process of performance analysis and also provide an economical way of doing so. The models can be generalized for distributed environments for computer resource management. This would extend the ease of analysis to distributed operating systems. An excellent treatment of performance models for distributed environments is given in [14]

In the network based file systems the possibility of increasing the concurrency of operations should be investigated. This is likely to improve the performance at high user loads. Further, the studies should aim at bettering the performance for diskless configurations, as these really prove to be cost effective and easy to administer in a multiuser software development environment.

## References

1. RFC 1094, "NFS: Network File System Protocol Specification" Network Working Group, Sun Microsystems, Inc., March 1989.
2. Andrew P. Rifkin, "Remote File Sharing: Architectural Overview", *USENIX Conference Proceedings*, Summer 1986, pp 248 -259.
3. The DOMAIN System User's Guide, Revision 01, Apollo Computer Inc. 1987.
4. Joshua Levy, "A Comparison of Commercial RPC Systems", Report to Atherton Corporation, Sept. 1989.
5. H. Zimmermann. , "OSI Reference Model-The ISO Model of Architecture for Open Systems Interconnection", *IEEE Trans. Communications*, vol. COM-28, April 1980, pp 425-432.
6. Andrew Tannenbaum, Computer Networks, Prentice Hall Inc., 1981.
7. 1057, "RPC: Remote Procedure Call Protocol Specification", Sun Microsystems, Inc., June 1988
8. RFC1014, "XDR: External Data Representation", Sun Microsystems, Inc., June 1987.
9. George F. Couloris, et al., Distributed Systems: Concepts and Design, Adison-Wesley, 1988.
10. Andrew D. Birrel, and Bruce Jaynelson, "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, Vol 2 No. 1, Feb 1984, pp 39-59.

11. Howard Chartock, "RFS in SunOS", *USENIX Conference Proceedings*, Winter 1987.

12. Edward D. Lazowska, et al., "File Access Performance of Diskless Workstations", *ACM Transactions on Computer Systems*, Vol 4, No. 3, Aug 1986, pp 238-268.

13. John H. Howard, et al., "Scale and Performance in a Distributed File System" *ACM Transactions on Computer Systems*, Vol 6 No. 1, Feb '88, pp 51-81.

14. Cheriton D.R. and Zwaenpoel W., "Distributed V Kernel and its Performance for Diskless Workstations", *Proc 9th ACM Symposium on Operating Systems Principles* , Oct 1983, pp 128-140.

15. Wilbur H. Leyman, Performance Analysis of Transaction Processing Systems, Prentice Hall, Inc., 1989.

16. Charles Hedrick, "Introduction to Internet Protocols" CSFG, July 1987, Rutgers, The State University of New Jersy.

17. Douglas E. Comer, Internetworking with TCP/IP, Prentice Hall Inc., 1987.

18. A.S.Melamed, "Performance Analysis of UNIX-based Network File Systems", *IEEE MICRO*, Feb 1987, pp 25-38.

19. John Quaterman, et al., "The UNIX System: 4.2 BSD" Report to the Department of Computer Sciences, University of Texas, Austin, May 1985.

# APPENDICES

# APPENDIX A

## The Benchmarks and Results

### 1. The Neal-Nelson's Business Benchmark

The Business Benchmark from Neal-Nelson Associates profiles a machine's operation under a given configuration by running a series of standard programs. There are eighteen different categories under which the program tests a range of simultaneous tasks.

The program measures the time required for each copy to complete each test under various load conditions. When the benchmark has collected the information, a report is generated about how the machine responds under increasing levels of activity. The two tests pertinent to distributed file systems were run for this study. The results of these tests are outlined in Tables A.1 and A.2 and Figures A.1 and A.2.

**Test 1:** This test consists of a 500 cycle loop with one read of 512 bytes inside each loop. It is intended to measure the speed of block disk I/O and data transfer while reading.

Read Times

| Local | Remote |
|---|---|
| 3 | 7 |
| 1.5 | 11 |
| 1.5 | 22.5 |

| | |
|---|---|
| 2.8 | 25.75 |
| 6.20 | 32 |
| 10.8 | 41.5 |
| 12.3 | 48.5 |
| 14.14 | 52.0 |

**TABLE A.1: Remote and Local Reads in RFS**

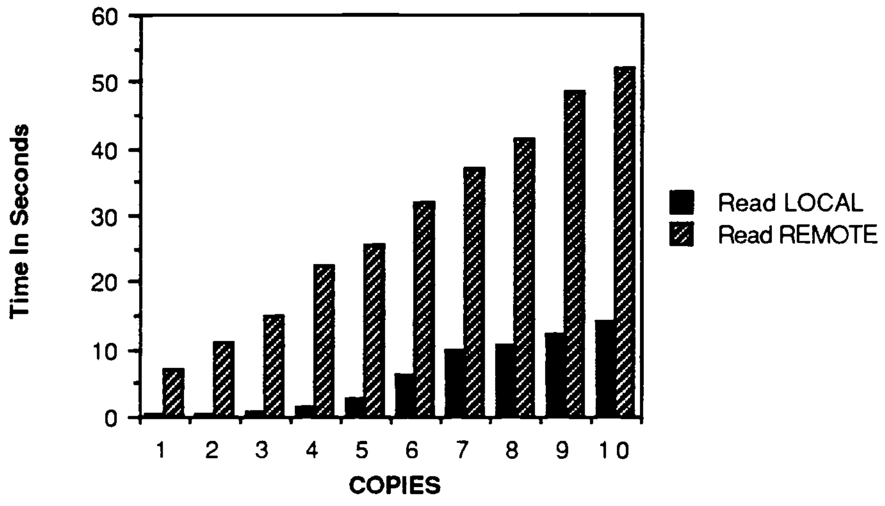

**FIGURE A.1: Comparison of Remote and Local reads**

**Test 2:** This test consists of a 500 cycle loop with one write of 512 bytes inside each loop. It is intended to measure the speed of block disk I/O and data transfer while writing.

Write Times

| Local | Remote |
|---|---|
| .3 | 20 |
| .4 | 41 |
| .6 | 32 |
| 6.25 | 112 |
| 16.4 | 115 |
| 20.3 | 121 |
| 24.1 | 127 |
| 28.6 | 129 |

**TABLE A.2: Remote and Local Writes in RFS**

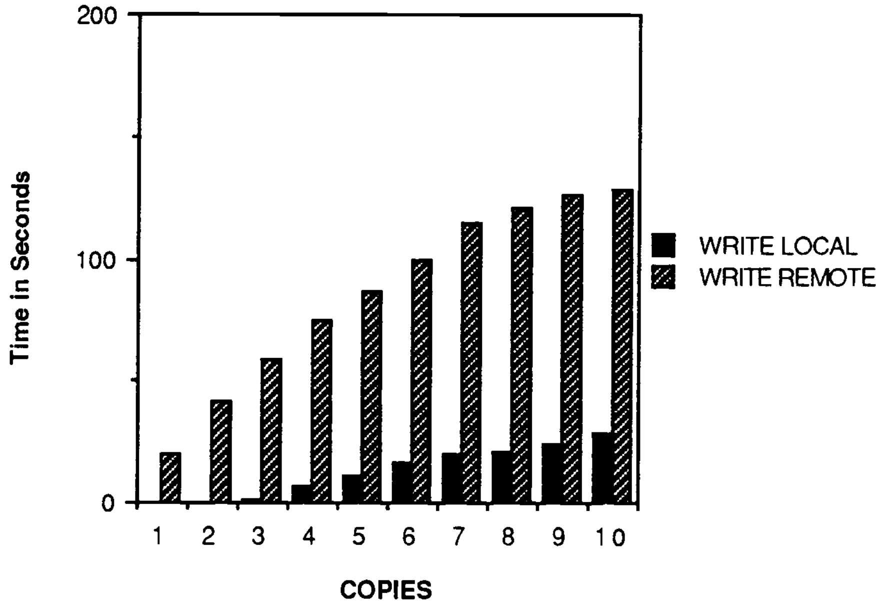

**FIGURE A.2: Comparison of Remote and Local Writes**

## 2. The Multivendor Test Suite (MVTS) FOR NFS.

The multivendor test suite is a set of programs from Sun Microsystems Inc. to test the performance of various implementations of NFS. It contains 2 tests.

Test1: Read and Write tests:

These tests measure the time elapsed for remote and local writing and reading of a 10 Megabyte file 10 times.

Test2: IOPERF:

This program measures the elapsed times for remote reads with increasing block size. Actually IOPERF just tests the functionality of the NFS implementation at different block sizes by using a 100 Kbyte file. The times were measured using the "time" function of the csh.

The raw results obtained for the NFS MVTS are as follows.

**Test1:** **LOCAL:**

**Read:** 126.0 Seconds (83220 Bytes/Sec)

**Write:** 124.0 Seconds (84562 Bytes/Sec)

**REMOTE**

**Read:** 183.0 Seconds (57299 Bytes/Sec)

**Write:** 264.0 Seconds (39718 Bytes/Sec)

**Test 2:**

| BLOCK SIZE | ACCESS TIME |
|---|---|
| 4 | 15.0 |
| 8 | 9.9 |
| 12 | 7.0 |
| 16 | 6.6 |
| 20 | 6.0 |
| 24 | 5.0 |
| 28 | 4.2 |
| 32 | 4.0 |

**TABLE A.3: Raw Results of Test2**

# APPENDIX B

## Comparison of RPCs from Current Vendors

### 1. Introduction

RPCs (Remote Procedure Calls) represent a convenient and increasingly common method for distributing systems among networked computers. This Appendix presents a comparison of RPC systems from Apollo and Sun for speed and dependability.

Speed tests showed Sun's and Apollo's RPC products performed comparably for small packets, but Apollo's system degraded for increasingly large packets. Dependability tests showed both Sun (using UDP) and Apollo performed dependably at machine loads under 5, but Sun lost dependability at higher loads. All data was gathered using the commercially available versions of each vendor's software as of November 1988. *The work presented in this appendix is entirely due to Joshua Levy of Atherton Corporation.*

### 2. Network Environment

Network software is implemented in layers, each layer being built at the one below with several different protocols available on each layer. A common configuration has (from lowest to highest) ethernet, IP, UDP and TCP, and finally, an RPC protocol.

UDP is an unreliable packet oriented protocol, while TCP is a reliable connection oriented protocol. "Unreliable" means that a data packet may not arrive, may arrive after a packet sent before it (i.e. in any order), or may arrive more than once; the only guarantee is that every time a packet arrives, the data in it is uncorrupted. "Reliable" means data arrives exactly once, in the right order, and uncorrupted. "Packet oriented" communications are similar to a letter since each piece of mail is addressed and routed separately. "Connection oriented" communications are similar to a phone call, because a connection is established, messages are sent back and forth, and the connection is broken.

UDP packets have a maximum size, which varies from machine to machine, but TCP connections can handle any amount of data. However, in UNIX, each TCP connection requires a file descriptor, therefore, an application can have only a limited number of TCP connections open at any one time. Conversely, all UDP communications are sent through one file descriptor, allowing an unlimited number of UDP "connections".

## 3 Analysis

The results obtained by Joshua Levy were as follows

(1) Sun (udp) is the fastest RPC, when it is usable.

(2) Sun (tcp) is the fastest RPC, when Sun (udp) is not usable.

These differences in performance are understandable when a few facts are known about (1) which transport layer each RPC system uses, and (2) how it is used. Apollo uses UDP, making sure each packet is 1K or less. Sun (udp) uses UDP, using the maximum packet size, while Sun (tcp) uses TCP.

First, consider the two UDP based systems: Apollo and Sun(udp). Since both use the same communications layer they should be the same speed. An explanation for their speed discrepancy is the UDP packet size they use. Since UDP is unreliable, an RPC system using it must wait for an acknowledgement that its packet arrived. This means sending 2 1K byte packets take longer than sending 1 2K byte packet. The more data which must be sent (up to the maximum UDP packet size) the worse things will get for the Apollo implementation.

Second, compare UDP and TCP based RPC systems. Setting up and tearing down a TCP connection requires 6 small packets. Once the connection is set up, however, TCP can move data more economically than UDP. The result: UDP will always be faster for RPC calls which require a small number of UDP packets to move the arguments. TCP will become faster as more UDP packets are required. The exact number of UDP packets which must be sent to make TCP more economical depends on each protocol's implementation and the type of network being used. Empirical tests show for 8 UDP packets worth of data (8Kbytes), Apollo is slower than Sun (tcp).

## 4 Conclusions

The proper conclusion for an analysis of this sort is an algorithm, which, given an application and its implementation constraints, returns the "best" RPC system to use in implementing that application. Unfortunately, generalizing such an algorithm is difficult. Here are some general rules to keep in mind while evaluating an RPC product:

(1) Determine what is important to your application: speed, dependability, ease of use, a particular feature, etc. Each RPC system has its own strengths and weaknesses. They are not all the same.

(2) Distributed programs can be modified to use any RPC system very quickly, in as little as a week. It may be worthwhile, therefore, to experiment with several systems before deciding on one.

(3) For small packets (below 1K bytes), Sun (udp) and Apollo are the same speed. As the amount of data increases, however, Sun outperforms Apollo.

(4) Once an application is using RPCs, the overhead of having the service routine(s) on a foreign machine (as compared to the local machine) is small.

(5) For lightly (under 5) loaded machines, Sun (udp) and Apollo are equally dependable, but as the load increases, Sun becomes less reliable.

(6) If an application has a relatively small number of servers and clients, it may be better to open a permanent TCP connection between them.