# The File System Assistant: A File System Distribution Technique

*Jim Harrell*, Cray Research, Inc., 655-F Lone Oak Drive, Eagan, Minnesota 55121

**ABSTRACT:** *The MPP architecture used on many of today's computer systems provides some interesting challenges to maintaining high-performance I/O. This paper describes some of the reasons for I/O performance problems, the need for I/O distribution, and the mechanisms used on the new Cray UNICOS/mk operating system to support high-performance I/O on its future supercomputer mainframes.*

## 1 Introduction

This work has been done in UNICOS/mk for the Cray T3E. This paper is divided into two parts. The first is a general description of the UNICOS/mk project goals and the I/O problems inherent in a distributed architecture system. The second part describes the specifics of the File System Assistant (FSA) implementation. The FSA is an unique means of providing scalable I/O for a distributed system.

## 2 Background

In 1993, Cray Research shipped its first Massively Parallel Processor (MPP) system, called the CRAY T3D system. The CRAY T3D system is tightly coupled to the Cray Research UNIX operating system UNICOS on a CRAY Y-MP or CRAY C90 system across dedicated channels. It is the UNICOS operating system that provides all the operating system services for the CRAY T3D system. The operating system on the CRAY T3D system provides machine-dependent support on the local processor element (PE) and forwards all system calls to a UNICOS system on the CRAY Y-MP or CRAY C90 system.

The CRAY T3D system provides good support for the applications that are run on it. It also provides a good rationale for moving the UNICOS system to the Cray MPP platform to provide all of the operating system services on a few PEs. This system was targeted for the next generation of Cray MPP providing support for large high-performance applications.

Cray Research is creating a microkernel-based version of the UNICOS operating system, called the UNICOS/mk operating system. The initial purpose of the UNICOS/mk system is to support the next generation Cray MPP system, called the CRAY T3E system. The CRAY T3E system needs a scalable, distributed operating system, which concentrates vital system services in a few PEs and provides minimal services on the PEs where applications are run.

A couple of important hardware features played a significant role in how the new operating system development was directed. The first feature is universal access to all I/O "ports" from any PE. There are no peripherals local to PEs; all I/O is done to very high speed I/O rings called the CRAY SCX channel. The devices are connected to the SCX channels.

The second feature is the MPP interconnect. All the PEs are connected with a 3D Torus. The interconnect performance supports high bandwidth/low latency communication. This was used as the vehicle to make universal access work on a distributed system, where forwarding system call requests and replies would be possible.

The port of the UNICOS system from a traditional vector processor machine to the new UNICOS/mk organization was made possible by rebasing basic system services and machine-dependent functions in a microkernel. The rest of the system services were reorganized and divided into a dozen or so servers. This allows distribution and scalability of servers.

In addition to this fine-grain serverization, the process manager (PM) server provides the UNICOS interface to applications. The PM is distributed to allow some system calls to be processed locally on a PE, while the more "global" calls, or parts of calls, are forwarded to other servers.

Using this organization, vital operating system functions are located on a configurable number of PEs (referred to as the OS PEs), thus allowing the other PEs (that is, PEs that run the applications, which are called compute PEs) to forward many system calls to these OS PEs. The UNICOS/mk system provides a single system image across all the PEs. In this new system organization, all system services are equally accessible from any PE. This system is now being used as a development vehicle on CRAY T3D systems.

Copyright © Cray Research Inc. All rights reserved.

## 3 The need for I/O distribution

The modeling of this system done by Cray Research shows that while the distribution of services works well in most services, the I/O intensive applications require less latency in request processing, and some mechanism to avoid having the file servers flooded by the request rates of the compute PEs. In order to meet the needs of I/O intensive applications, a small support server called the File System Assistant (FSA) has been created. The FSA resides on each of the compute PEs.

The FSA provides simple distribution of the basic I/O functions. The remainder of this paper discusses the background and implementation of the FSA server.

## 4 Choices on MPP Systems

I/O support for applications on MPP systems is based, in part, on the organization of the operating system. The following paragraphs describe the different choices from our perspective.

### *4.1 Complete/replicated systems*

MPP operating systems that have complete operating system services on every PE provide local I/O when files are local and remote I/O via NFS, DFS, or some other remote file service when the files are not local to the PE. This works well for some kinds of applications and is used by several vendors of MPP systems.

The obvious advantage of a complete local operating system on a MPP is that when I/O is local, it goes faster than sending the request off to be processed somewhere else. However, when data has to be loaded and unloaded from local PE systems in order to run applications, that time needs to be factored into the cost of I/O because it requires the use of the local processor, keeping it from running the applications during any load/unload operation.

It is worth mentioning that while there were no local devices on the CRAY T3E system, and therefore the necessity of providing a complete local system was not present, that replicated/complete systems would probably not have been our consensus choice for the following reasons:

- The difficulties in providing a single system image on top of multiple systems
- The difficulties in organizing files to be available on the PEs where applications need them.
- The amount of memory required for complete operating system services
- The processing and interrupt time for local daemons makes it difficult to schedule multi-PE applications without having some processes in the multi-PE application waiting for other processes to be scheduled.

### *4.2 Distributed systems*

Several MPP systems, including Cray Research's, use operating system service nodes instead of complete system support on all PEs. These systems usually have microkernels and enough operating system support to manage the PEs hardware, and handle application system calls by forwarding the requests and passing the replies back to the application.

A system that concentrates all the operating system services on a few nodes and forwards all application requests must bear the latency and potential for bottle necks at the PEs that process the system call requests.

Systems that use forwarding can limit the kinds of applications that are supported. Another course that has been taken before is to initially provide a system that forwards all system calls, and then later to support some number of system calls locally on all PEs, and forward the remainder to OS PEs.

## 5 The I/O bottleneck

There are many system calls that do not occur often, and this is one rationale used to justify not providing complete system support on all PEs. The simplest alternative is to simply forward all system calls to OS PEs for processing, which is easy. However, most system call reports for Cray machines, show read, write, and seek as the number one, two, and three system calls by usage. In most systems, system call usage and system time are related. Generally, these calls together claim better than 75% of overall system call usage. Usually the top 10 system calls account for between 90 and 95% of total system calls processed during any reporting period.

Because of this, and a general desire to make I/O perform well, some simple models were generated to determine how I/O would work in the new system. This work indicated two areas that needed attention. The first was IPC overhead and the second was I/O scalability.

### *5.1 IPC Overhead*

The IPC overhead had two categories of problems. The first was that the IPC implementation was inefficient. IPC is a large concern because historically message passing systems get scrapped because of, among other things, the overhead of the messages. However, this is a general problem, and is not addressed in this paper. The second overhead problem was that the protocol between the process manager and the file server had too many IPC messages.

There are a number of solutions that can and are being used to reduce the IPC usage, and thereby the overhead. First, the protocol between the process manager and file server was changed to reduce the number of messages. A different approach was taken when the servers were on the same PE. In this case, direct jumps replace the messages between the servers. The protocols between other I/O-related servers, like the file server and disk server were also changed so that the sleep waiting for I/O completion would take place in the file server.

Changes were made to avoid the context switch that is part of a remote IPC from one PE to another. In this case, the interrupt thread wakes up a thread to process the incoming IPC. The whole path can involve multiple threads and several sleep/wake-up pairs on each PE. Context switches are extremely expensive and these are avoided whenever possible. One solution was to provide interrupt threads that were guaranteed not to sleep.

### 5.2 Scalability (User Controlled)

There are different ways of approaching solutions to I/O scalability problems. The very best way is to reduce the number of requests from user applications, and ensure that all I/O is done directly to and from user buffers.

Another change is to make sure that the use of all positioning functions is reduced. For some reason, positioning functions precede many I/O requests even when the I/O is strictly sequential. There are a series of POSIX position-independent I/O functions that provide the capability to pass position and the I/O request in the same system call.

At first glance this would appear to violate the UNIX philosophy of simple, single function system calls, but the positioning addition does help, especially in reducing request load. Other important tools are interfaces that provide parallel request processing, like listio, or interfaces that take advantage of hardware capabilities, like allowing distribution of I/O to a number of PEs. The Cray MPP hardware supports an I/O centrifuge feature, which allows I/O to be read and written to multiple PEs at the same time. Software support for this feature was added to the listio(2) system call.

### 5.3 Supporting the I/O request load by distribution

The strategies that require changes in user programs, or expect particular user behavior are not going to solve all the problems. Some heavy users can be expected to seek optimized solutions, but others may not. Even though the MPP system is a relatively new architecture, there are going to be applications that will not be able to be structured in a particular fashion. It is a better strategy to expect to support a broader variety of application I/O profiles.

The UNICOS/mk system is aimed at supporting large multi-PE applications, but the system will also provide a development environment for interactive users, and single PE applications. Because there is a mix of requirements, and the single PE and interactive will consist of mostly small I/O operations, it is believed that small I/O should be optimized. Small I/O here is defined less than 8KB. I/O at device speeds is also provided.

The last goal involved predicting the amount of I/O required. This is always difficult in the absence of hard numbers. Previous experience with large systems suggests a few thousand per second, but this has not been studied carefully. Cray Research has set the goal at 400 to 800 I/O operations per second per PE. Given this goal and a large number of PEs it was rapidly proven that a single file server would become a bottleneck.

There are a couple of ways of spreading the I/O load. Three slightly different mechanisms are as follows:

- A remote mount, multiple file server mechanism
- Multiple file servers using shared tables; this is based on the Shared File System concept already in use at Cray Research.
- A slave or assistant file server on every compute PE

The following sections describe these mechanisms in more detail.

#### 5.3.1 Remote mount mechanism

The remote mount mechanism uses multiple file servers, each mounting some number of different file systems, which are connected by remote mount to form a single tree of file systems, as is normally found on a UNIX system. This is a simple solution because remote mount is a relatively simple concept to implement. However this solution depends on I/O requests naturally spreading evenly across different file systems. Unfortunately this rarely happens.

#### 5.3.2 Multiple file servers sharing tables

Another way of spreading the I/O is to use multiple file servers sharing the file system tables and using a locking protocol to protect access to the tables. Variations on this mechanism include partitioning the vnode or file tables among the file servers and passing requests between them, depending on which file server held the tables that would be used by to satisfy the I/O request.

This method would be useful where high speed locking primitives existed in the hardware. In this case, an extended version of a multi-threading locking mechanism would be a good place to start. If a table partitioning scheme is used, then the locking and request passing might support file systems spread across a network.

There are several difficulties with these slightly different Shared File system mechanisms. The first is that the scaling will be determined by the number of file servers sharing the tables. The second is that there is probably an upper bound to the number of file servers that could be involved in the sharing, which is assumed, without benefit of data, to be small.

#### 5.3.3 File system assistant (FSA)

The last mechanism is the file system assistant (FSA). This is a special server that resides on every compute PE. The purpose of a FSA is to off-load read, write, and seek operations from the file servers that are on remote OS PEs to the local PE. The open, close, and other less used operations are forwarded to the file server.

This allows reads and writes to be processed locally, independent of the file server and removes the latency and overhead of the IPC to the file server for request and reply processing. Moreover, because the process manager (PM) and FSA are local, the communication is done directly via jumps instead of requiring IPC to marshal arguments, copy messages, and switch thread contexts.

The FSA is composed of a server main function and message management routines, a copy of much of the Cray local file-system code relevant to read and write operations, and stubs that forward and respond to other file system requests. The FSA server is much smaller than a complete file server.

The FSA does not do any caching of data. The current implementation requires an open(2) flag to enable the use of the FSA by the file server. Other restrictions are to use regular files and well-formed I/O (that is, I/O done on sector boundaries so as not to require caching). Note that currently, the default mode of I/O is direct to the user buffer. This was inherited from the vector version of the UNICOS system. Coherency problems are avoided by not caching data. Data is cached in the file server and/or the disk server.

The FSA can be used or bypassed. The remote file server can be used at the same time as the FSA. The different servers can be used by the same application on different files without interference or confusion.

The flow of control on open(2) and close(2) is similar to each other. The PM on the local compute PE receives a request from a user program. The request is forwarded to the remote file server. The file server sends an IPC to the FSA which contains vnode and inode data, disk block allocation data, the location of the disk server to be used, flags, and so on. The FSA replies to the local PM. All future communication concerning this open file for this application will be sent directly to the FSA.

A close operation goes from the user application to the local PM to the file server. The file server closes the file and forwards the request to the FSA which completes the close.

The disk block allocation will eventually run out or be invalidated. In this case, the FSA makes a special request to the file server for another allocation. The reply is the updated allocation, which is handled by the FSA. A truncate request to the file server generates a broadcast invalidate to all FSAs that have the file open.

The read(2) and write(2) flow of control is from the user application to the local PM and then to the FSA, which processes the request locally. Assuming the blocks are all within the local map, then the I/O request is turned into a request to the disk server and that request is forwarded via an IPC call.

## 6 Summary

The File System Assistant is but one attempt to address I/O scalability and I/O distribution problems on distributed architecture systems. The problems described in this paper are not specific to the Cray T3E, but is inherent in all distributed or client-server systems. Results to date indicate that the FSA has greatly reduced the latency of the dominate I/O system calls.