PARALLEL I/O SYSTEM FOR A CLUSTERED COMPUTING ENVIRONMENT

LIU MING

(B.Eng, Harbin Institute of Technology)

A Thesis Submitted

For the Degree of Master of Science

School of Computing

National University of Singapore

Jan 2006

ACKNOLEDAGEMENTS

First, I'd like to thank A/P Wong Weng Fai, my supervisor for all his guidance, support and direction.

Thank my thesis examiners, A/P Teo Yong Meng and A/P Yap Hock Chuan Roland

Thank the PVFS staff for the assistance.

Thank the Myrinet team for the help.

Thanks all my friends for helps and cares.

TABLE OF CONTENTS

Summary

Clustered computer systems become most vigorous star in this high level computing game due to its high performance and low cost. In this environment, a parallel file system is well adopted to obtain higher performance. Most parallel file systems are trying to pursue the speed, the performance. For high-level computing, system availability is also a big issue that should be considered.

To evaluate the influence coming with the system availability, we should experiment a parallel file system and a revised system with the availability based on the former then compare performance differences. Journaling and redundancy are two main techniques in this domain. In this thesis, we choose a popular parallel file system, the Parallel Virtual File System, as the prototype to primarily evaluate the effects on systems after bringing in the availability. We mount 2 PVFS systems on a client to build up a Twin-PVFS and use our own API functions to implement the RAID-1 Level redundancy evaluate its influences. First, a series of tests in different situations, such as the data file size, network and the number of I/O node, is designed to totally measure the performance of PVFS. Then we choose some data that are proper to be compared and test our Twin-PVFS and the original PVFS on the same circumstances and parameters. For the comparability, a parallel access mode with PVFS API also has been tested. The journaling mode was presented also. The test result shows that this availability reduces the system performance a lot but this influence differs in the specific situations, i.e. the network bandwidth and the file data size.

LIST OF FIGURES

Chapter 1. Introduction

Modern science and commerce require high computation capability and large storage capacity more and more. Therefore, they are always one of the driving forces to accelerate the computer development. Fortunately developing processing techniques make them possible. The computation speed of order of magnitude of GFlops and the storage devices of order of magnitude of TBytes introduce the computers into some extreme complicated computation, such as the Earth simulation [1], weather forecasting [2], and the nuclear weapon simulation [3].

Nowadays, a personal computer (PC) with a powerful single chip containing about 1 hundred million transistors (or more) can exceed several time-shared behemoths of dozens years ago. To achieve the increasing needs for high performance, clustered PCs become a cheap and proper solution.

The first PC-based cluster was born in the Earth and Space Sciences project at NASA, 1994[4]. A parallelized computer system, especially a cluster in this thesis, means that this super power computer locally consists of many parallelized workstations, even PC's connected by a local high-speed network switch. These computers have their own independent CPU's, memories, I/O systems. Each CPU also does only a part of job parallel, exchanges the data with their own memory and saves the results in their own disks or later moves them to other devices. The fastest Linux cluster in the world is Thunderbird in Sandia National Laboratory of U.S.A. with Dell PowerEdge Cluster. The list of cluster systems in the TOP10 of 2005 Nov, see the Appendices, shows that IBM eServer has succeeded in the highest computing. But the number of clusters in the full TOP500 grew also again strongly, 360 of 500 use cluster architecture. These systems are built with workstations or PCs as building blocks and often connected by special high-speed internal networks. This makes clustered systems the most common computer architecture seen in the TOP500. The importance of this market can also be seen by the fact that most manufacturers are now active in this market segment [6]. This trend is more apparent because building a super cluster will extremely reduces the cost and time on the design of supercomputer hardware architecture and dedicated software. Thus a cluster can be a poor men's supercomputer. Hence the total amount of its disks capacities can reach the order of magnitude of Tbytes. It may be enough to store the necessary data by a parallel file system. In this paper the term, "the Parallel file system" refers in particular to "the Cluster file system".

High performance computing ordinarily processes a large amount of data. New modern multimedia applications also require I/O devices with high bandwidth. Unfortunately after improvements of decades, the data transfer rate of a single hard disk is still lower than we expect. A single local file server can not satisfy large numbers of high bandwidth applications, such as Video-on-demand (VoD) with MPEG-1/2/4 quality, the earth geography science. The disk speed becomes the bottleneck after solving the low network bandwidth. Borrowed the concept from RAID, a cluster can use stripe technique to obtain higher data throughput.

This clustered file systems obtain great bandwidth, balance those computers' load, but it also brings us a big problem, the reliability. Unlike distributed file system

[5], fault tolerance is not the first aim of parallel file systems. If one of those nodes breaks down the files on this cluster will be fragmentary. If the data can't be recovered, it'll be an unacceptable disaster.

1.1 Contributions

The Parallel Virtual File System (PVFS) [28] is a popular file system for clusters. It has some common features of a typical parallel file system, i.e. high bandwidth, segmented files, balanced loading and faster access speed. But it also has some disadvantages we will recount later.

Perhaps one of the most insufferable weaknesses is none-redundancy. In high performance computing, data processing costs a lot of time. Any data loss in processing, such as processes hang up, operating system breaks down, bad sectors on disks, other hardware fail, might ruin the whole work. In a single computer, it might happen rarely; in a huge cluster with many relatively independent computers, the risk of data loss will be consumedly increased.

A failure of an I/O node will cause the failure of the whole cluster because the data is distributed over the cluster. The objective of this dissertation is to evaluate the performance effects after adding RAID-1 mode to PVFS to obtain higher availability, reliability and redundancy. At the beginning we build up a cluster and install PVFS on it with different nodes respectively and do a series of tests. We simply mount 2 PVFS on a client machine and use our own API to access PVFS in parallel to simulate RAID-1 mode, called Twin-PVFS, with the same environments above and do the same tests. It's assumed that the newcome availability will take more I/O operations on the system and the system might be slowed down but it is not clear in our mind how the influence will be in detail. We will not wonder if its performance is not better than the prototype. After analyze the results of tests, we can evaluate the effects on PVFS this new feature takes. Because the PVFS stripes file data across multiple servers like RAID-0, after introducing this additional RAID-1 mode, this Twin-PVFS becomes RAID-01 mode. That means one of I/O node is down, its twin can still work. This ensures that the whole file system is still working. To simplify this project, the main structure of the PVFS is still adopted in this revision.

The main difference with CEFT-PVFS [36][37] briefed in the next chapter, is that our system is based on the mount function of PVFS which the latter two systems have thrown away. This approach migh be slow, a parallel access mode with PVFS API is used to contrast.

1.2 Outline

The rest of this thesis is organized in the following way. In Chapter 2 we explore the background, the history and the current state of arts of file system. Topics include file system, disk storage system, distributed file system and parallel file system. Chapter 3 focuses on the presentation of the prototype of our system, i.e. PVFS and our revision. The system performance and comparison with the PVFS are measured and analyzed in Chapter 4. The concluding chapter summarizes our results and forecast the future works of PVFS.

Chapter 2. Background and Related works

In this Chapter we provide background, basic concepts and some related works about the file systems, especially the Parallel File System.

2.1 File System

Usually a file is used to store data on the storage devices by application programs. A file system is the software that creates some abstractions including not only files and directories but also access permissions, file pointers, file descriptors, and so on. File systems have other duties as well [7]:

- Moving data efficiently between memory and storage devices
- Coordinating concurrent access by multiple processes to the same file
- Allocating data blocks on storage devices to specific files and reclaiming those block s when files are deleted
- Recovering as much data as possible if the file system becomes corrupted

The file system isolates the applications from the low-level management of the storage medium and ensures that concurrent applications do not interfere with another. Applications refer to the files by their names which are textual strings.

2.1.1 Hierarchical Name Space

A file system is built as a tree with a single root node, the root / ; each node in this tree is either a file or a directory, every non-leaf node is a directory and every leaf node can be one of directories, regular files or special device files. A file name is given by a path name that describes how to locate this file in the file system hierarchy. Thus, a full path name includes a path name and a file name.

The file system treats file data as an unformatted stream of bytes; directories are also considered as regular files in the low-level respect, the system treats a directory as a byte stream, but these directory data contain the file names in the directory in a special format so that the programs can find the files in a directory.

2.1.2 Access Model

Which file can be accessed is controlled by Access Permissions mechanism. There are three classes of users to implement read, write and execute permissions: the file owner, a file group and the others.

When a program opens a file, the file system assigns a unique pointer and a unique file descriptor to it [8]. This pointer is an integer that points a position at which the next byte will be read or written. A file descriptor is an integer which the program uses for subsequent references to the files. In Unix-like file system, each inode (the index node) contains the information of the file data layout on the disk and other information about the file owner, access permissions, access time and so on.

When a program accesses a file by its file name, the file system parses the file name and checks the permission to access the file and retrieve the file data. After an application creates a new file, the kernel assigns it an unused inode. Inodes are stored in the file system but the kernel reads them into an inode table when it manages files.

There are two other tables that are maintained by the kernel also, the file table and the user file descriptor table [8]. These 3 tables control the file state and access permission.

2.2 RAID

RAID, short for Redundant Arrays of Inexpensive Disks or Redundant Arrays of Independent Disks, was proposed at the University of California, Berkeley in 1988[9]. This invention was to address the disk system performance and reliability since the data transfer rate of a single disk can not suit the necessity of modern computing. In the original paper, there are five RAID levels differing on the performance characteristics and the ways to replicate data, RAID-0, RAID-1, RAID-2, RAID-3, RAID-4 and RAID-5. For some special applications, the combination of some of those existing levels is introduced, such as RAID-10, RAID-53. In recent years, RAID-6 with 2 parity disks and RAID-7 with the combination of hardware and build-in software appear.

2.2.1 RAID-0

In RAID-0 mode, the data is striped across the disk array without any redundant information. The loss of a single disk will corrupt the whole data. This simple design doesn’t supply a good availability but it supplies the good performance because it doesn’t need to do some extra disk read or write to implement more availability and compute some extra information. Recently benefiting from the hardware price’s fall, even some PCs are equipped with these once-expensive devices.

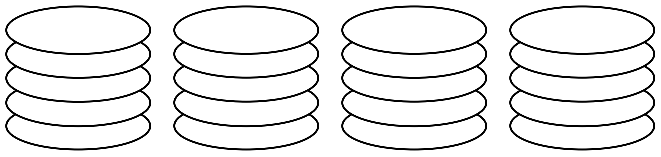

Figure 2-1 RAID-0

2.2.2 RAID-1

RAID-1 uses a simple manner, disk mirroring, to implement the redundancy. When data is written to a disk, the same info is written to its twin disk. Usually the writing operation can be operated in parallel; the writing time for this RAID node is just a little longer than one for a single disk. When data is read, it can be retrieved from the disk with shorter queue, seek and rotational delays [10], because the read transfer rate by retrieving alternate blocks from both disks in parallel. If a disk fails, the another copy will take over the responsibility to finish the job. But this improvement on the availability wastes too much because the whole disk array has two identical parts. Mirroring is frequently used in database applications where availability and transaction rate are more important than storage efficiency [11].

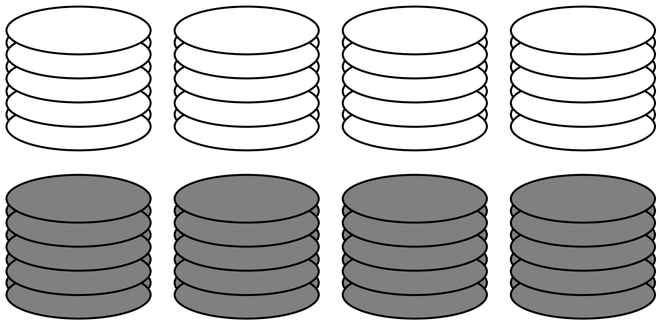

Figure 2-2 RAID-1

2.2.3 RAID-2

RAID-2 uses Hamming codes containing parity information to provide higher availability. Once a disk fails, the rest disks will find out which disk fails and give the correct answer because Hamming codes can find the errors that happened to the file data and correct it. This approach requires some additional disks to implement the parity calculation and this calculation costs some system computing capacity.

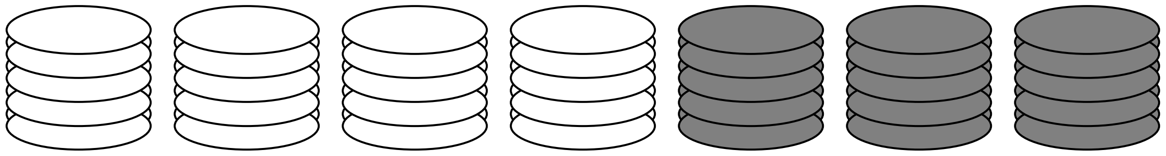

Figure 2-3 RAID-2

2.2.4 RAID-3

RAID-3 is a simplified version of RAID-2. Instead of multiple ECC bits applied in RAID-2, bit parity is used in RAID-3. This scheme consists of an array of disks for data and one unit for parity exclusively. The system XOR data bit by bit in these sub-stripes to write an additional parity sub-stripe to the parity disk.

When each write request is operated, the whole stripe with parity is written to the disk array in parallel. For read operation, only the data disks involve in it. If any of disks fails, it restores the original data by an XOR between the redundant bits on other disks and the parity disk. With RAID 3, all disks operate exactly simultaneously. It requires that all of disks must have identical specifications to maximum the performance. This is not a very effective method for accessing small amount of data, but RAID 3 is rather suitable for specialized use where large block of data need to be processed at high speed, as in supercomputers, multimedia warehouse.

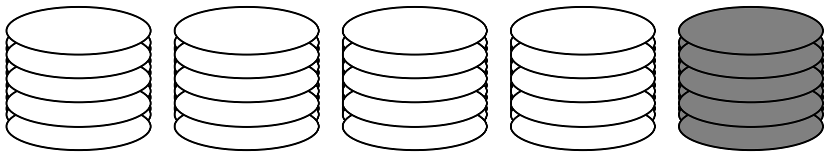

Figure 2-4 RAID-3

2.4.5 RAID-4

RAID-4 adopts the block-interleaved parity disk array in which the data is interleaved across disks in blocks of arbitrary size. Like RAID-3, it has a disk array to store the file data and puts the parity data on a separate parity disk. Unlike RAID-3 with parallel read/write per operation, RAID-4 accesses some of disks individually.

For read operations, firstly it determines which disk the requested block resides on and then accesses them only, even only one disk is accessed for small files. Write operations cause some overheads because of its individual access mode. To write data to the disks, RAID-4 only updates those related disks and the parity disk. It requires a series of operations:

1. read the old data from the sector being overwritten and the old parity from the parity disk;

2. extract the old parity data using the XOR operation;

3. XOR the new file data and obtain the new parity data;

4. write the new data and the new parity data to the respective disks.

The main drawback of RAID-4 is that it stores all parity data on a single disk. Write operations must read and then write the parity disk every time. Obviously, the parity disk might be the bottleneck of the system easily. Thus, RAID-4 is not well accepted in real systems.

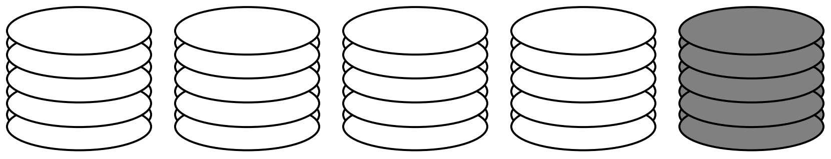

Figure 2-5 RAID-4

2.4.6 RAID-5

RAID-5 appears as an improved RAID-4 with fully striped disk array. The parity data is not stored on a single disk; it is distributed over the entire disk array. This means each disk has the parity data of file data on other disks in interlace.

Read operations only access those disks that have the required data. Write operations have the same drawbacks with RAID-4; the process of read-modify-write still affects the system performance for those applications that require high transfer rates for write operations.

A good method, called left-symmetric parity distribution, was invented in [13], has the best performance. The advantage of this method is that whenever we traverse the striping units sequentially, we will access each disk once before accessing any disk twice. This property reduces disk conflicts when servicing large requests [14].

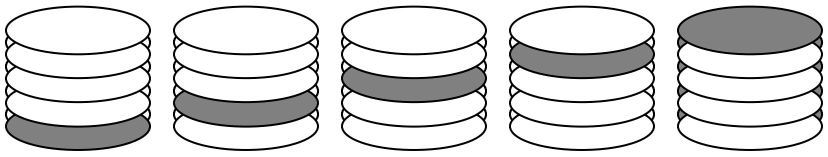

Figure 2-6 RAID-5 Left-Symmetric Parity

2.3 Distributed File Systems

The file system we discussed above runs on a single machine. The concurrent accesses to the same file are allowed after ensuring the sequential consistency.

A distributed file system makes it possible that many computers have a common view on a file set or a file system. The first famous distributed file system should be Network File System (NFS) developed by Sun Microsystems in 1985 [15]. NFS allows computers connected each other by the network to share files. In NFS, the computer sharing its files is a server, and a computer that accesses these files remotely is a client. In other words, a computer can synchronously be a server for some files on its own machine and a client for some files that reside on other machines. After mounting a directory and its subdirectories on the server in their own directory hierarchy, the client accepts these remote files as a part of its directory hierarchy and the programs on the client can access them as local data. When a client is going to access the remote files, the file system on this client sends a request to the server and gets the return. How these remote directories are located is transparent for the user level.

2.4 Parallel File System

A parallel file system is a tightly coupled networked file system. It stripes file data across many computers by a local network. After the stripe technique brings higher data transfer rate, the disk transfer rate is no longer the system bottleneck if the network is fast enough. However it also makes the file system more complex.

2.4.1 Striping

The fastest SCSI 320 can provide the maximum data transfer rate at over 100MB/s per disk [35] but a modern switch can provide the high bandwidth with order of magnitude of Gbps or even Tbps. To achieve the supercomputer capacity, the data transfer rate must be much faster.

The striping technology is the key to achieve high performance for a parallel file system. The term striping is from the RAID prototype. It means that a collection of data is allocated on several computers and each computer only stores a portion of file data. This striped data usually is split into a string of fixed size blocks that are assigned cyclically to the nodes.

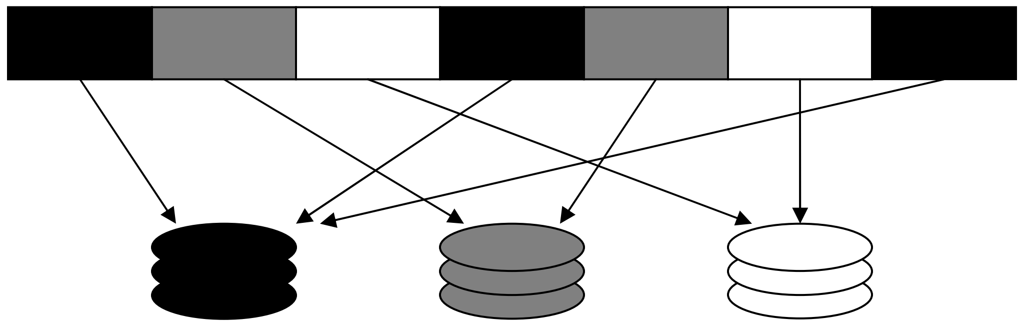

Figure 2-7 Disk striping

Two main parameters decide how this data will be distributed in the striping scheme [7]:

- **Stripe factor.** This term means the number of disks in striping. It determines the striping degree and further determines the parallelism degree, the data transfer date.
- **Stripe size.** It is used to define those striped blocks size. For different tasks, the requests may differ, some needed data are many small records and some data are huge files. For the former, a small striping size will get a better performance; for the latter, a big striping size will reduce the frequency of sending read/write request. Therefore, some parallel file systems set striping size as a variable to match different requirements.

2.4.2 Structure

Unlike the server-less structure in some distributed file systems, most parallel file systems use the client-server model. Compared with the Network File System, this model can save network communications, reduce the system complexity but also reduce the reliability because the server may crash down.

A server has the responsibility to manage the striped data info, i.e. the metadata. Metadata, in a file system, refers to information describing the characteristics of a file, such as permissions, the owner and group, and the physical distribution of the file data [16]. In the case of a parallel file system, the file distribution has more info, i.e. the file locations and the disk/node locations. Some of nodes in the system are called I/O nodes or I/O processors. They are the warehouse to store those file data. The rest of nodes, we call them the compute nodes/processors, are designated to run the users' applications.

2.4.3 File access

In a parallel file system, each I/O node only maintains a subset of a file [7]. Accordingly every file has an inode on every I/O node. To access a file in a parallel file system, a process will get every inode of this file. There are two ways to achieve this: the first one is to duplicate all of the directory information on each I/O node; the second one is to set an across-nodes name server to solve the naming space. In the former solution the data change in any I/O node is also done on other I/O nodes. It causes frequent internal communications between I/O nodes. In the latter solution the name server takes the obligation to direct the processes. The processes only contact the server to locate the required data and the changes of striped data on those I/O nodes are recorded on this server. However, the whole file system relies on server states. The reliability of this server is depressed.

Parallelized access has more complicated issues than the local access on the sequential consistency. But basically it also uses the similar techniques in the local file systems, such as lock, token ring. Since the file data is striped, the consistency will cover two layers: the file layer and the striped file data. In Client/Server node, the consistency on the file layer can be handled by the server and the consistency on the striped data can be handles by the local file system. In Peer-Peer mode, the consistency of these two layers is the responsibility of the manager daemon in the file system.

2.4.4 Buffer

To improve the efficiency and accelerate the system, the buffer technique is well adopted: firstly the data will be put into the buffer, a space in main memory, when trying to read from or write to the disk. How to keep the consistency of the data in the buffer and on the disk is a big issue in a local file system. We can imagine how to keep the consistency between those nodes is a bigger problem. The striped and shared data requires more complicated approach to solve this two levels puzzle. For the well-used Client/Server mode, there are two types of buffer: the buffer on the compute nodes and the buffer on the I/O nodes [12].

2.4.5 System Reliability

In a local system, if a disk fails or the system crashes or the power supply is lost while an application is in progress of writing, the file data and the inode info may be in an inconsistent state, and when the system is rebooted the data in the memory will be lost.

A parallel file system is a complicated and cooperated system with many components. To finish huge amounts of works, usually it runs day to day. During its running, any fault, from message timeout to broke-down I/O nodes or server, may cause a string of troubles. Those striped data more complicate this problem, i.e. the crash of one or more I/O nodes may cause the data incoherence and further the entire file system may hang up.

To address this tragedy, each parallel file system chooses its own strategies to maintain the data files and inodes in a coherent state since the avoidance/recovery approaches are related to another issues of the file system.

- **Redundancy**

  One machine may die sometimes; the probability of two machines crash simultaneously must be much less. The main idea of this manner is to replicate each I/O node so that every I/O node has at least a node to be the backup in this system. Each replica of a file is stored on a different node. When a node fails, the replicated copies of its files can be used to provide uninterrupted service to its clients. This is a highly available and reliable solution, but also an expensive solution because duplication slows down the system speed and wastes the storage space. A great revision of the duplication is the similar method like RAID. In RAID-3, 4, 5, only one extra node with the parity is added so that the cost of duplication on the performance and the I/O nodes are less.

- **Logging/Journaling**

We try to keep the consistency because we don't know what will happen if a system crashes, how many data has been saved. The approach to record this is called logging or journaling. Any modification to the file data or inode info on the disk would only take effect when the record that logs those actions is done. Those logs are stored on an area of disk that contains the records that describe what is changed in the file system and they are kept separate from the file structure they describe to avoid losing the file data and its log together.

2.4.6 Some File Systems

High performance, scalability, high throughput and high availability are four basic features of clusters [12]. But there is no perfect parallel file system in this world that has all of these features concurrently. The usage of a cluster determines which of them is requisite and which is dispensable. Science computing keeps driving high performance; Business requires high availability; Web service needs high throughput. A variety of requirements cause mixed products.

Intel's Concurrent File System (CFS) [17], frequently cited as the canonical first generation parallel file system, and its successor, PFS [18], are examples of file systems that provide a linear file model to the applications, and offer a Unix-like mount interface to the data. There are four IO modes in CFS, 0, 1, 2, 3. By using different IO modes, it is very easy to decompose the data across the disks.

Zebra [19] combines LFS (Log-structured File System) and RAID so that both work well in a distributed environment. Zebra uses a software RAID on commodity hardware (workstation, disks, and networks) to address RAID cost disadvantage, and LFS batched writes provide efficient access to a network RAID. Furthermore, the reliability of both LFS and RAID makes it feasible to distribute data storage across a network. Several striping file systems, such as Bridge [20], strip data within individual files, so only large files benefit from the striping. Each Zebra client coalesces its writes into a private per client log. It commits the log to the disks using fixed-sized log segments, each made up of several log fragments that it sends to different storage server disks over the LAN. Log-based striping allows clients to efficiently calculate parity fragments entirely as a local operation and then store them on an additional storage server to provide high data availability. Zebra's log-structured architecture significantly simplifies its failure recovery. Like LFS, Zebra uses checkpoint and roll forward to implement efficient recovery. Although Zebra points the way toward serverlessness, several factors limit Zebra's scalability. First, a single file manager tracks where clients store data blocks in the log; the manager also handles cache consistency operations. Second, Zebra relies on a single cleaner to create empty segments. Finally, Zebra stripes each segment to all of the system's storage servers, limiting the maximum number of storage servers that Zebra can use efficiently.

Log-structured File System (LFS) [21] was developed at Berkeley. xFS is also implements based on the LFS [22]. It provides high performance write, simple system recovery and a flexible method to locate the file data. LFS treats the disk like an appending log. This approach solves a big problem for the file system on small files writes. It is feasible to implement soft RAID on this file system. LFS uses a data structure, called imap to locate the inode. The imap which contains the current log pointers to inodes is stored in memory and periodically saves the checkpoints to disks. These checkpoints are the key to the system recovery. After a crash, only the consistency of the log tail needs to be checked. LFS runs from the checkpoint and update the metadata. Only the part of the log that last checkpoint creates since the crash happened is used to recover. The main drawback of LFS is the log cleaning. Sometimes it is the bottle in a system [23][24] .

The General Parallel File System [25][26] developed by IBM was designed to achieve high bandwidth for concurrent access to a single file, especially for sequential access patterns. GPFS is implemented as a lot of separate software subsystems or services. Each service may be distributed across multiple nodes within an SP system. GPFS is also a client-server cache design and consistency is maintained by the token manager server. This is employed for scalability reasons: distributing the task to the mmfsd reduces serialization at the token manager server. GPFS 1.2 has some functionality limitations: it doesn't support memory mapped files; when clients send data to the servers faster than it can be moved to disk, GPFS have a performance limitation; the data path also describes the potential bottlenecks; data are copied twice with the client; when the applications access the file in small pieces, sequential access patterns can be a disadvantage [27].

The Parallel Virtual File System (PVFS) [28] project is an effort to provide a parallel file system for PC clusters on Linux. It provides a high-performance and scalable parallel file system. There we give a brief description of it. We will discuss it in details in next chapter as the prototype. PVFS spreads data out across multiple local disks in cluster nodes. Thus, applications have multiple paths to data through the multiple disks on which data is stored by providing cluster-wide consistent name space. The architecture of PVFS is composed of one IO library and two kinds of daemon: The *manager* daemon manages metadata associated with PVFS files (e.g. file attributes, stripe unit size, and list of IO nodes) and runs on one node of the cluster. The IO daemons run on several node and store and retrieve data of PVFS file in parallel. The IO library provides parallel IO functions and interacts directly, such as MPI-IO with the daemons. The main advantage of such architecture is that there is no need to modify the underlying operating system. PVFS' constraints are these two points: no file locks implemented; no fault tolerance. Its successor PVFS2 **Error! Reference source not found.** has implemented some redundancy.

A cost-effective, fault-tolerant parallel virtual file system (CEFS-PVFS) [36][37] is a revised PVFS. It implemented RAID-1 mode on PVFS to incorporate fault-tolerant into parallel file system by mirroring. In CEFT-PVFS, the system has been separated into two independent groups. Each group has its own *mgr* node and I/O nodes. Four mirroring protocols have been designed in this system to evaluate its performance. Client nodes connect these two groups in different ways in these four protocols. Like our system, this RAID-1 mode wastes 50% of disk space for mirroring.

Modularized redundant parallel virtual file system (MRPVFS) [38] is another extension module to PVFS. RAID-4 mode redundancy has been introduced in this system. The functions include parity striping, fault detection and on-line recovery. MRPVFS has a parity cache table to solve the concurrent write problem. An extra SIOD is used to store the parity information of other IOD, on the mgr node. Therefore this mgr server becomes the weakest part of MRPVFS. Parity calculation and metadata storing require a powerful and stable hardware.

Chapter 3. System design and implementation

In this chapter, we will illuminate the system design and implementation in detail. This design is build on a popular parallel file system, the Parallel Virtual File System (PVFS). Hence, firstly we will describe this file system.

3.1 PVFS

Like many parallel file system, the primary goal of PVFS is to provide high speed access for applications. Other important features of PVFS are a consistent file name space across a cluster, transparent access for clients and user-controlled striping of data across some or all of I/O nodes.

3.1.1 System Structure

The client/server mode is adopted in PVFS. There are 3 elements in an entire system: clients, a manager server and I/O nodes. They all run on user level. PVFS relies on the local native file system at each computer.

The following figure shows the system overview [29].

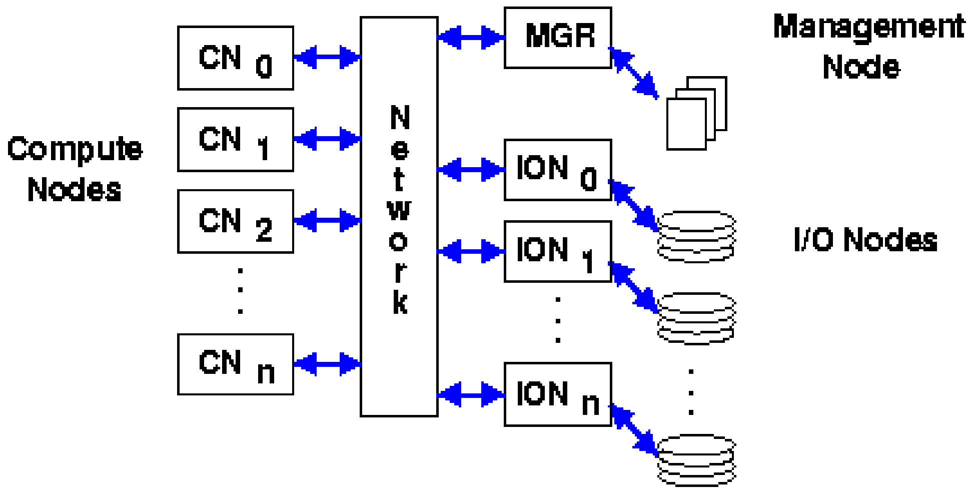

Figure 3-1 PVFS Overview

Clients are computers where user daemon runs and from there the requests are sent to the PVFS. PVFS supplies two approaches to access its file system: PVFS library and kernel module. PVFS library has dozens of native function, like *pvfs_open*, *pvfs_read*, *pvfs_write*, *pvfs_close*, *pvfs_lseek*, *pvfs_access*, *pvfs_ftruncate*. This API gives us a powerful and flexible manner to develop our applications. The kernel module is not compulsive but it makes those simple file manipulations more convenient. The commands from PVFS, like *pvfs-ls*, *pvfs-mkdir*, *pvfs-ping*, *pvfs-truncate* etc, are similar with those Unix/Linux systems although only some basic commands are supported.

A manager daemon runs on the manager server that manages those file metadata, such as file name, its place in the directory hierarchy, its owner and distribution info across nodes in the cluster. It doesn't store any real file data on itself. Its duty is to receive the requests from the clients, check the requests with the metadata, determine the requested file distribution and transfer the order to the relative I/O nodes. In this progress the manager does not participate in read/write operations; the client library and the I/O daemons handle all file I/P without the intervention of the manager [16]. Of course the manager will record the changes on these file data. This edge reduces the data transfer over the network, liberates the manager servers from those heavy I/O operations and drives this file system to speed up.

I/O nodes where I/O daemon runs on store the file data under that manager server. Those file data are split up into some pieces by Round-Robin algorithm and stored on the disks of these I/O nodes. PVFS gives the users the chance to determine how to distribute these files, i.e. where the file will be stored from, how many I/O nodes will be used and how big the stripe size is. In details, those parameters are defined in a data structure *pvfs_filestat* .

```
Struct pvfs_filestat
{ int base;  /*The first I/O node to be used */
  int pcount; /* The number of I/O nodes being used */
  int ssize; /*stripe size */}
```

In the function *pvfs_open (…, struct pvfs_filestat *dist),* we can stripe the file data in the way we like.

The picture below shows the data flow of the access to PVFS [30][31].

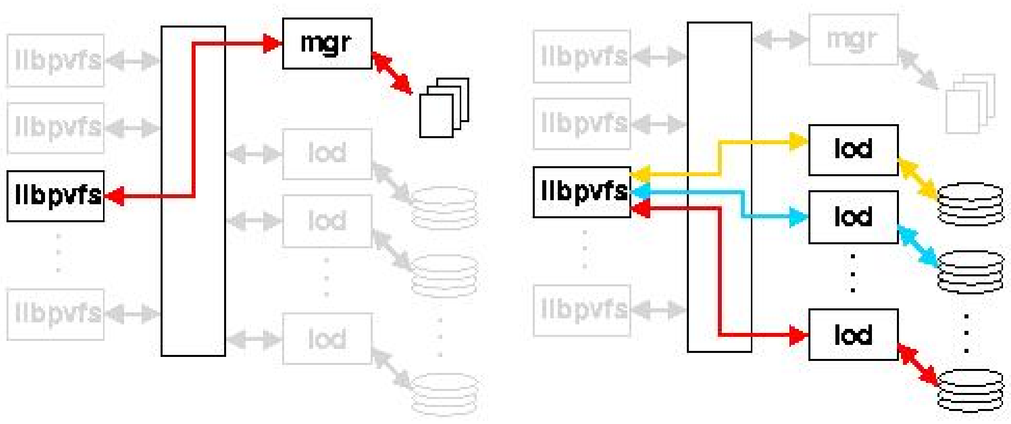

Figure 3-2 Data flow on PVFS

This picture indicates the data flow through the kernel [32]. In this mode, those existing programs can simply access PVFS without any modification. For this, it is transparent for users. The PVFS is mounted just like a device. The VFS receives the access requests from applications and transfer it to the kernel. */dev/pvfsd* is the bridge between that loadable kernel module and the pvfsd daemon. The daemon pvfsd is the signalman whose duty is to send/receive the network transfer from/to clients/manager server of PVFS.

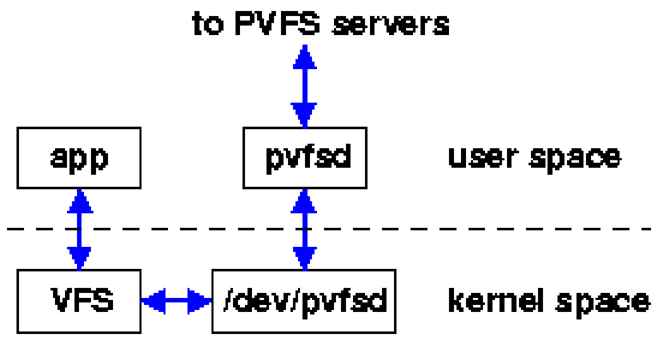

Figure 3-3 Data Flow Through Kernel

These three components are running on the user level being daemons. It is doable to run some of them on a machine without interference. We even can run all on a computer to do a test.

PVFS even supports the logical partition that allows an application access only a part of a file by describing the related regions. The offset, group size and stride are parameters to implement this feature in the structure *fpart* of function *pvfs_ioctl(…&fpart)*. The offset is the distance in bytes from the beginning of the file to the first byte in the requested partition. Group size is the number of continuous bytes included in the partition. Stride is the distance from the beginning of one group of bytes to the next.

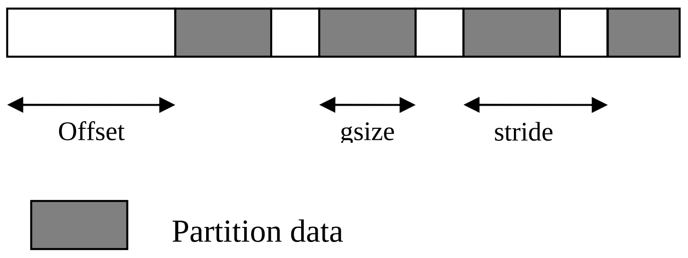

Figure 3-4 Partitioning parameters

3.1.2 Advantages and Vulnerabilities

After booming in 90's, PVFS still survive and be popular, the following reasons make it possible:

- High performance.
- Easy to install, configure.

- Flexible access methods.
- Support team and developer team.

Of course, we must put its vulnerabilities on another scale of our balance.

- No redundancy and recovery.
- Potential bottleneck on manager server.
- TCP/IP network.
- Single thread.

There is not a perfect file system that can do everything very well, performs any kind of task wonderfully. In fact only those applications that apply for large data queries can benefit a lot from PVFS, like datamining. For those small files queries, we can imagine that these small files are segmented to smaller files on I/O nodes; the disk head will move very frequently; it will create many fragments and waste disk space.

3.2 System design and implementation

Our goal in this thesis is to dig out a way to introduce the redundancy into PVFS and evaluate the effects coming along with this new character.

3.2.1 Twin-PVFS Architecture

The daemons of PVFS are running on user level. It is allowed to run some same or different kind of daemons on a single machine to play respective roles. For example, we can load the client daemon, manager daemon and I/O daemon on a single computer; or, we can run the same kind of daemon belonging to different PVFS on the same node.

To implement concurrent access mode, here we mount two PVFS daemons, *pvfsd*, on a single client node. Obviously these two daemons on the same computer must have different ports to respectively contact their PVFS manager daemons, *mgr*. There is a cryptic option *–o port=xxxx* that does not appear in PVFS official documents. The default port of PVFS manager daemon *mgr* is *3000* that is defined in its source code and can not be changed after the installation. Here we set the default port to one PVFS and a different port to another PVFS. Therefore these two PVFS mounted on the same client computer will not interfere with each other. We send and receive the same file data storing on these two PVFS from and to the client node. This mode looks like RAID-1 for these two PVFS overall and RAID-01 for each node partially.

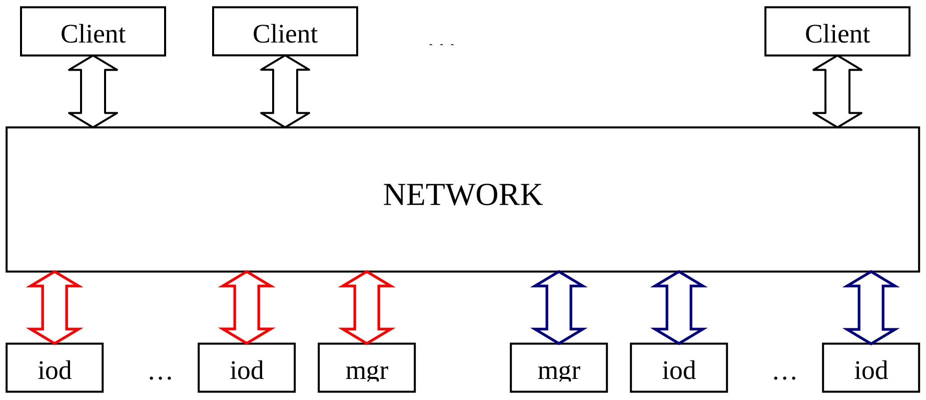

Figure 3-5 Twin PVFS

3.2.2 Access

As a mounted Linux VFS, PVFS can be accessed directly by using BSD like command. System commands, such as *cp, rm, ls, mv…* and system calls, such as *open, read, write, close, lseek* … can directly manipulate the files in the mount point folder. In our Twin-PVFS, the POSIX standard API (known as pthreads) is used to compile our own API to implement RAID-1 parallel access combined with system calls. All operations will be directed to two sub PVFS by its kernel module.

- Read

In RAID-1, the required data are not read the whole amount from both sources. Sub-read access is separated into two parts and each part retrieves the data alternately from both nodes in parallel. When sub-read is done on both nodes, the read operation returns the data to the buffer. If one of read operations fails, that part of system will read that data again. If the return is still wrong, it will stop and report.

- Write

Similarly, write operation will duplicate data to both nodes. Write operation will be done when both sub-write finish writing to the files. If an error occurs during writing file to two subsystems, it will be reported to user and the sub-write will be done again. If the wrong return value is still there, it will report again and stop, wait for solving the problem.

Other functions have the same mechanism.

3.2.3 Fault-tolerance and Recovery

The client mounts these two PVFS on two different mount points. We divide all I/O nodes into 2 groups belonging to 2 manager servers. Each PVFS is independent of another one. Since we write data to them respectively, if some nodes fail during the write operation, another PVFS should still work very well and the return value of that bad node will not match the requested data amount or if it doesn’t return after a waiting time. After that, an error message will be showed on the client node. It is the same for read operation. In RAID-1 mode, read can be run in parallel by retrieving alternate blocks from both disks. If a sub-PVFS crashes during twin-read, the return result will not match the data amount we have requested or if it doesn’t return and then it will report read error and a single read from a healthy node will do again.

By reason of that users manipulate the *client* nodes, system error on them will be sensed lightly. Hence the focus has been put on the *mgr* and *I/O* nodes.

Since access files request from *client* nodeson PVFS will be started from opening this file by contact *mgr* node, then *mgr* node will contact *I/O* nodes. If some bad things happen here, it will return an error. For closing files, the procedure is the same, *mgr* will ask *I/O* nodes to close the files and return to the *client* node. Between these two steps, *client* nodes and *I/O* nodes will communicate directly. For users, faults on *I/O* nodes will be easily detected on the *client* nodes. A simple ping-like monitor is set on *client* nodes. It sends the network requests to both the mgr nodes and I/O nodes periodically. Here we set the time slot to 30 seconds to see if any of them fails. This watchdog will guard the Twin-PVFS when no above activities on the system.

So far there is no automatic recovery in Twin-PVFS. After receiving an error report, it’s the user’s responsibility to find the type of trouble and repair or replace the fault node. We assume there is at least one good copy of data because the possibility of that two systems crash simultaneously will be rare. What we should do is just copy or update good data to the repaired or replaced node.

3.2.4 File lock and consistency

PVFS is on the top of Linux VFS. In our system, we didn’t design a local file lock protocol, just use the lock provided by VFS. Since *client* nodes must access data file through the *mgr* node, *mgr* will protect global locking on the same file in the same system.

This parallel access mode assures the data consistency. For read, the result is combined from two subsystem if read is done successfully. If it’s not, we can find out the problem and fix it because read will not change the data file. For write, if a subsystem is down, we still have a good subsystem with right data file. Duplication of this good subsystem to a fixed system will rebuild a health Twin-PVFS.

3.2.5 Parallel Access Mode

With fork() function in Linux, PVFS API can be used to implement multi-process access. Since API is the most efficient method to access PVFS, a small revised program combined fork and PVFS API is presented to do RAID-1 write and read operations.

3.2.6 Journaling

The redundancy can keeps the system's running without a total crash. However, after the redundancy part breaks down, the left part should run with less insurance. We do not exactly know when and where the accident happens so that we can not recover the previous works. The PVFS does not supply the journaling function. Even in Twin-PVFS model, if one portion of it fails, the whole system is still running, but it becomes a normal PVFS and the failed part can not play its role until the trouble is resolved, the last job is done and Twin-PVFS is ready to do new jobs. The PVFS runs on user space above the VFS. It might not perform better than those parallel file systems running on lower level. But it provides us an advantage also -- it can use the journaling technique supplied by EXT3. Here we change the data journaling from the default setting "*data=ordered*" to the fully logged mode "*data=journal*". The mode *data=journal* provides full data and metadata journaling. It ensures the consistency of data and its metadata.

To evaluate what kind of benefits or costs the parallel access brings to us, we also try to use the serial access mode to compare with the parallel mode. We modify the test program. That is we serially read/write the same amount of file data from/to two PVFS mounted on the same client machine.

Of course here we assume the probability that two or more nodes in these two PVFS fail coincidently should be very rare.

Chapter 4. Results

It can be forecasted that the performance of PVFS should be affected owing to we append some new members to an original PVFS. Redundancy means that the same file data will be operated more at the same moment. Apparently it will not speed up the system. For a file system, the best way to evaluate its performance is to do some experiences and analyze the results.

In this chapter, we will describe the test environment, execute the system test to appraise our revision, compare the results of the original PVFS and our revised system and analyze the data to find the influence of our revision on PVFS.

4.1 Test Environment

We create a small cluster with 7 nodes as our test bench. All of them have two Intel PIII 500MHz CPU's. One node has an IBM DDRS-39130D Ultra2-LVD SCSI disk and 512MB memory, we choose it as the client machine; two nodes have an IBM DDRS-39130D Ultra2-LVD SCSI disk and 256MB memory, we choose them as the metadata Servers; and four nodes have an IBM-DJNA-371350 IDE disk and 256MB memory, they are our I/O nodes. The operating system running on these nodes is Red Hat 9 Standard Edition with kernel 2.4.20-8smp. Those nodes are connected by 100Mbps Ethernet with a small SU-EP-808X \3B 8-port switch and 1.28Gbps Myrinet with M2M-DUAL-SW8 Dual 8-port Myrinet-SAN Switch and Myrinet M2M-PCI32C Cards. The *ttcp* benchmark reports a bandwidth of TCP/IP at

11MBytes/s for Ethernet and 60Mbytes/s for Myrinet. Bonnie[43] shows the hard disk bandwidth of write and read is, both 4MB/s per char, 52MB/s and 219MB/s for block, rewrite and seeks is 80MB/s and 19MB/s.

4.2 Test Components

To highlight the difference the redundancy brings in, we measure the PVFS and our revision on different conditions, from data size to network bandwidth.

We should clear it before analyzing the results that our client machine only has 512MB memory, it means, in those read tests, the data will be saved in some virtual memory on the disk when the file is too big. Of course this memory switch between the main memory and the virtual memory will cost a lot of time. Therefore we can not judge which one is better at that file size. The maximum size in our tests is 200 MB.

Since many parallel file systems use their own API to access the file data, a simple benchmark, similar with Ref.[16][36][37][39][40][41], was adopted to test the system performance. In this benchmark, each *client* node concurrently opens a data file, write new data on it and finally closed it; similarly, each *client* node open a file, alternatively read data from two subsystem and combine it in a buffer and finally close it. To test Twin-PVFS in a more complicated environment, the benchmark iozone[42] also was run on our system because its throughput mode supports POSIX pthreads access, similar with our system. Since its full and automatic test mode will not allow pthreads, here only write/rewrite and read/re-read modes have been tested.

4.2.1 PVFS

First, we test the original PVFS to build up our benchmark for the following comparison. It will take many experiments to fully examine a parallel file system because the basic operations, read and write, usually have different performance on different situations, such as the required data size, numbers of client nodes and I/O nodes, bandwidth. A chain of files with different sizes have been chose to operate basic read/write functions. In this step, we choose 1KB, 2KB, 5KB, 10 KB, 20KB, 50KB, 100KB 200KB 500KB, 1MB, 2MB, 5MB, 10MB, 20MB, 50 MB, 100MB, 200MB as the file sizes. To compare PVFS performance on different parallel scales, we build up several PVFS respectively with 4 I/O nodes, 2 I/O nodes and 1 I/O node. The local file system test is also put into our results to measure the PVFS performance. To avoid some random system delays, each test has been done 50 times and the result is the average of these 50 tests.

The data range we select is very wild, from 1KB to 200 MB; the cost time also differs from milliseconds to some dozens of seconds. Therefore it is impossible to show all data clearly in one chart. We divide those data into 3 groups, 1KB-50KB, 100KB-5MB, 5MB-200MB by the results trend.

First, we test those data respectively on 4 I/O nodes, 2 I/O nodes, 1 I/O nodes, with 100Mbps Ethernet. Then we do the same tests with Myrinet and compare the results. The test on all-in-one local node will just do one time.

- Ethernet

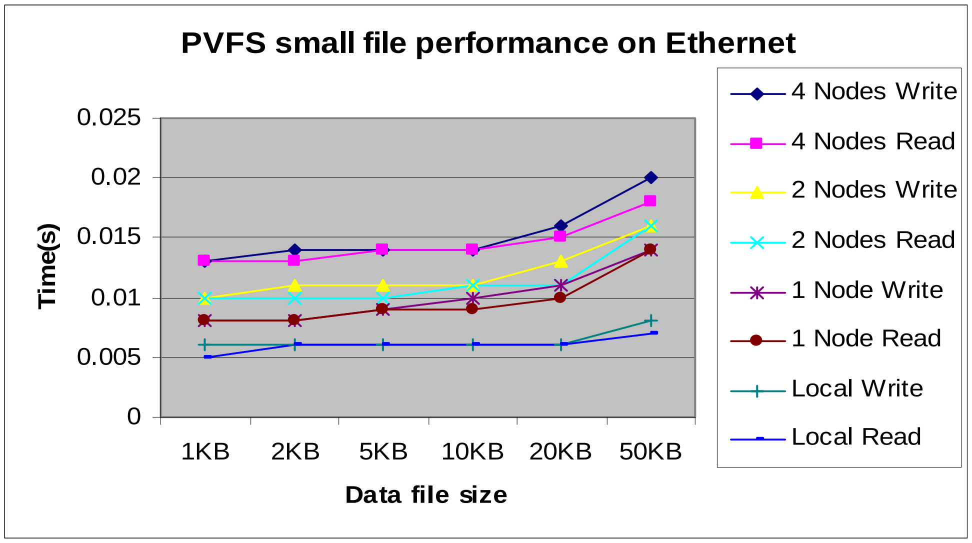

Figure 4-1 PVFS Small File Performance with Ethernet

In this small file test, we can find out that, of course, the local operations cost less time; PVFS on 1 node costs more; PVFS on 2 nodes is the third and PVFS on 4 nodes is the slowest. It illuminates that PVFS does not work well when transfer files are small. These results are obvious. When the files are small they will be transferred quickly through the network and they will be operated, read or wrote, quickly from or to the disk. Here, the bottleneck is the network transfer system. A modern hard disk can read or write these small files in a few milliseconds. Theoretically the Ethernet bandwidth is 100MB/s. It means in the real world it can transfer several dozens KB of data in a millisecond. But after counting in the system response time, message packing/unpacking and sending/receiving, it will be slower than the hard disk. Also, we can imagine if those small files are partitioned to smaller pieces, these tiny data may cost less time to be operated but they might still cost at least the same time to transfer. It is wasting the cluster resources.

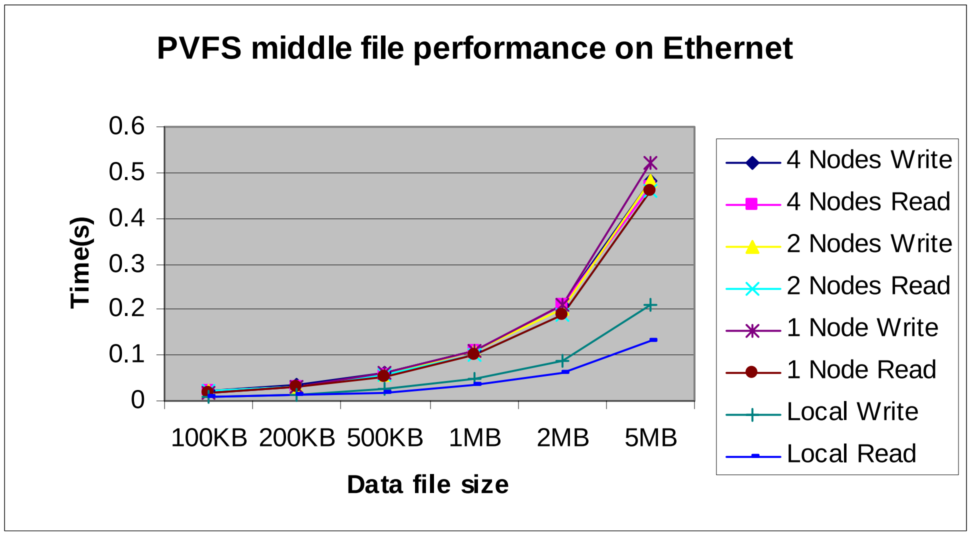

Figure 4-2 PVFS middle File Performance with Ethernet

For those middle size files, the local operations are still fastest. The operations on 4 nodes have the same results with those on 2 nodes. But they all a little exceed the PVFS on 1 node, especially the write operation. The data transfer speed of our I/O node disk is about 10MB/s. We can see the data approach the peak. Hard disk progressively becomes the bottleneck of these systems.

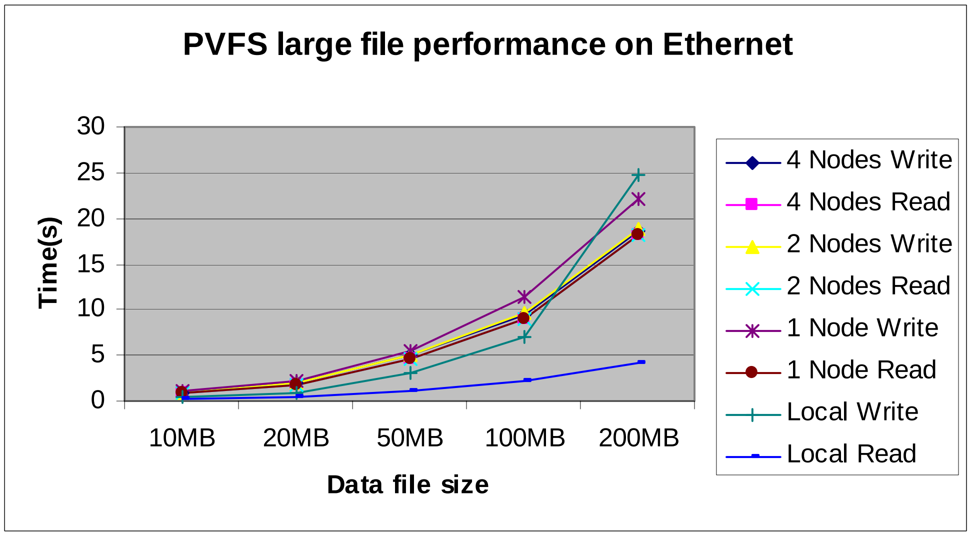

Figure 4-3 PVFS Large File Performance with Ethernet

Distinctly, the results are a little complicated. Local read is the fastest but local write is not. It is worse than the PVFS on 4 nodes and the PVFS on 2 nodes even the PVFS on 1 node. It might be caused by that the write operation needs more time and the write jobs are jammed in the queue. Parallelism shows its advantage here. The write speed is going a bit faster when the node number increases but all the read results are similar. PVFS on 4 nodes and PVFS on 2 nodes have the same data. The reason might be that the network bandwidth limits their ability.

- Myrinet

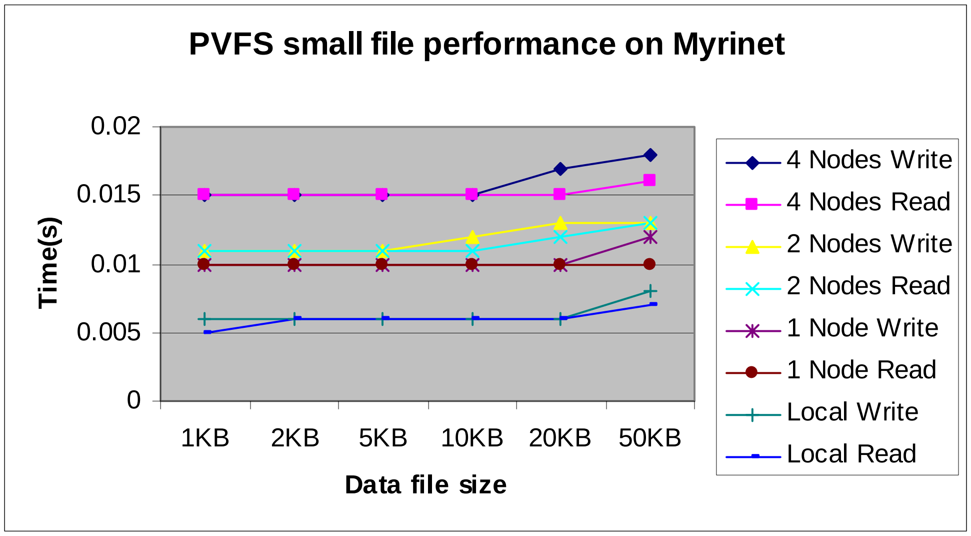

Figure 4-4 PVFS Small File Performance with Myrinet

For these small files, local operations cost the least time. PVFS on 1 node costs more and PVFS on 2 nodes is the third but they are very close. PVFS on 4 nodes, as we suppose, is the slowest. Even though the bandwidth increases, these small files still don't benefits from PVFS.

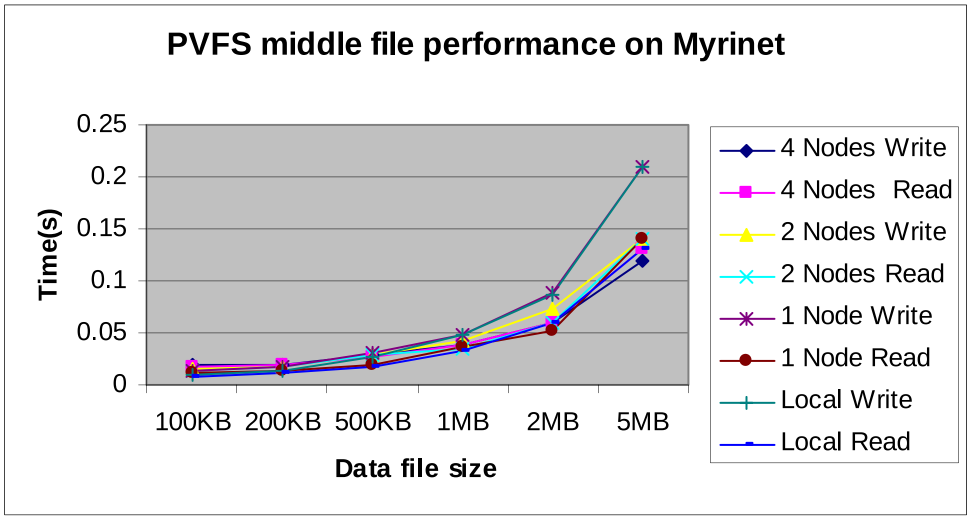

Figure 4-5 PVFS Middle File Performance with Myrinet

From this figure we can see how close the results are. Totally, the first one is not too fast and the last one is not too slow. Hence local write and write on 1 node highlight themselves coordinately as doubtless laggards. As we ratiocinate above, in this part, the disk approaches its maximum speed. Myrinet supplies enough bandwidth so that the local operation and PVFS on 1 node have very close marks. But PVFS does not show its advantage very clearly because the PVFS with 4 nodes and PVFS with 2 nodes have a similar performance.

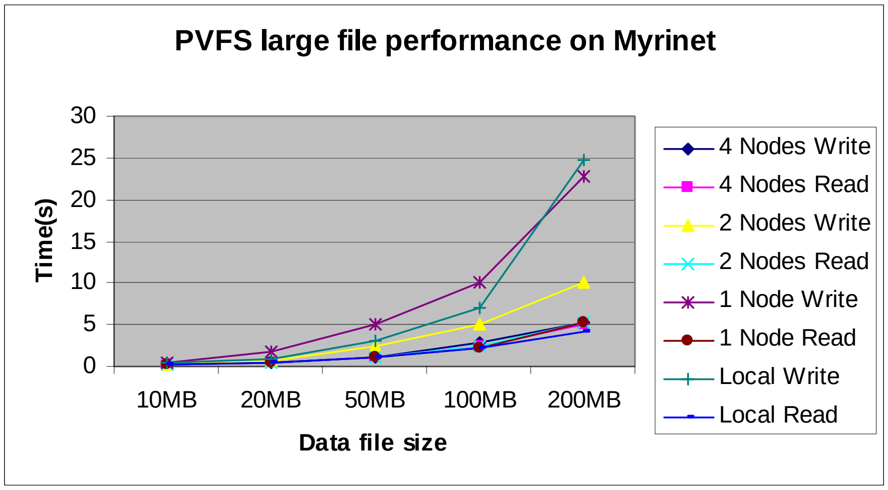

Figure 4-6 PVFS Large File Performance with Myrinet

PVFS with 4 nodes is the fastest one by its absolute preponderancy. All its read speed and write speed are fastest. PVFS on 2 nodes also has a good speed. Those 2 PVFS's get a better write performance than the local system. There is no doubt that PVFS suits the large file operations and the parallel scale helps the performance, especially for the write. For read, the speed of these PVFS in the same file size doesn't change much. Read on 4 nodes and on 1 node has a similar result. It seems the PVFS does not optimize the read operation very well.

4.2.2 Twin-PVFS

To compare with the original PVFS, we test our Twin-PVFS apart over Ethernet and Myrinet. As we observed above, PVFS is not good at small file operations. We assume that Twin-PVFS will do more disk operations to implement the availability. Therefore the test results might not exceed those original PVFS. Here we just test those larger files from 10KB to 200MB and drop out those files less than 10KB. Also, those data will be put into three groups, 10KB-500KB, 1MB-10MB and 20MB-200MB. Our Twin-PVFS has 4 I/O nodes and each PVFS has 2 I/O nodes. The client machine has 512MB memory, it mounts 2 PVFS metadata servers on itself, and each metadata server manages 2 I/O nodes. The instructions from the client will be sent to these two metadata servers simultaneously. There is no comparability between Twin-PVFS and the PVFS with 1 node or local file system. Certainly we just compare it with the PVFS on 2 I/O nodes.

- Read

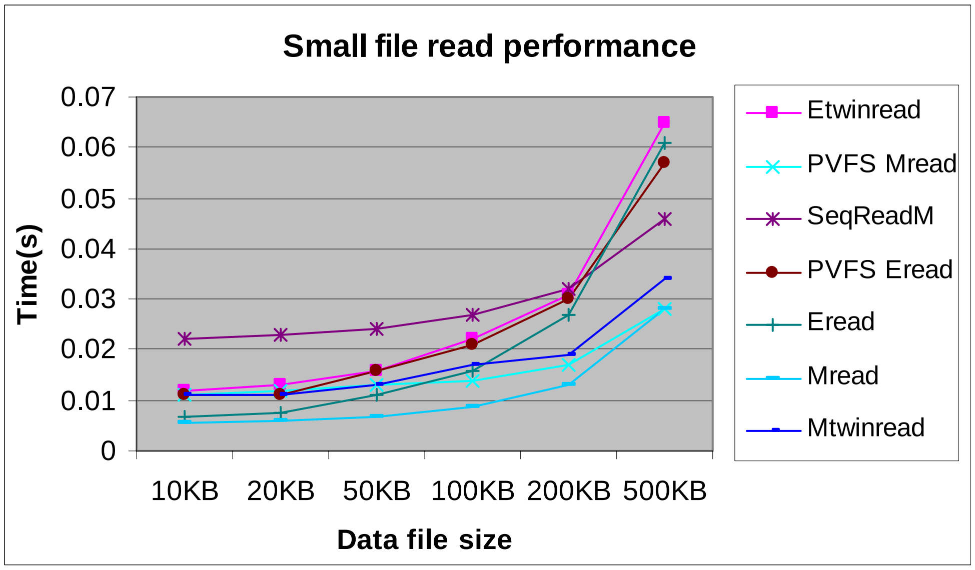

Figure 4-7 Small File Performance

Just like we assume, Twin-PVFS does not gain a good performance. It is worse than the original PVFS much, less than double time of them. Its read and write operations have similar results. When the file size increase, the network has showed its performance.

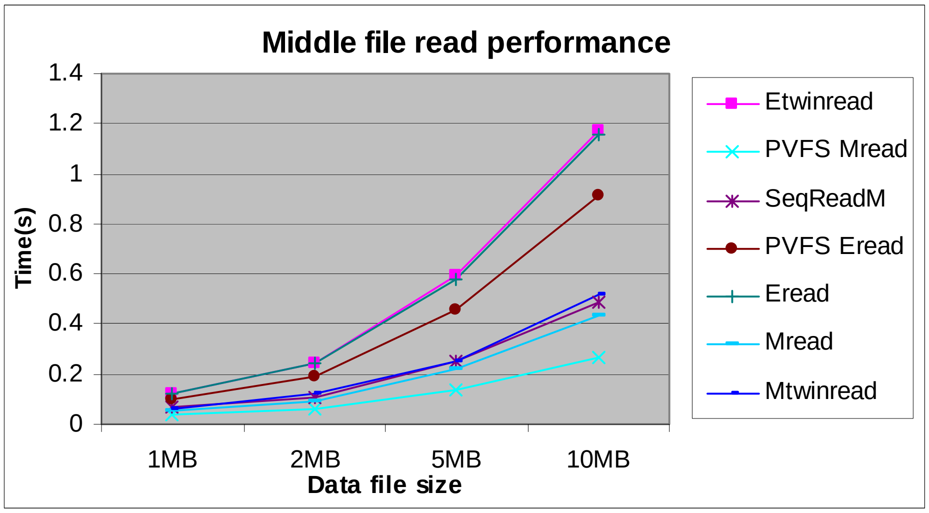

Figure 4-8 Middle File Read Performance

For these middle file operations, Twin-PVFS also does not perform well. PVFS API access is still the best. Twin-PVFS read scores are very close to the local PVFS kernel read.

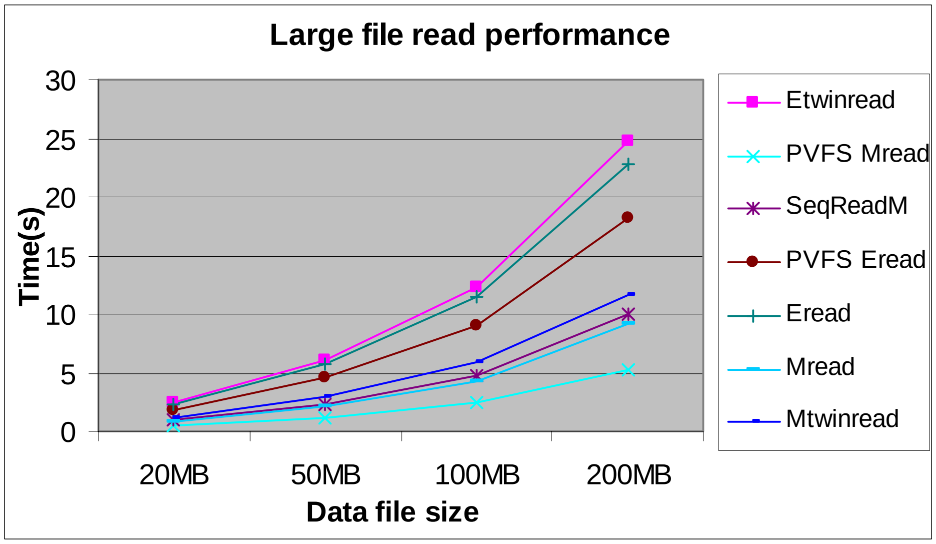

Figure 4-9 Large File Read Performance

The disparity between Twin-PVFS and PVFS becomes less in this part. The performance has been clearly divided.

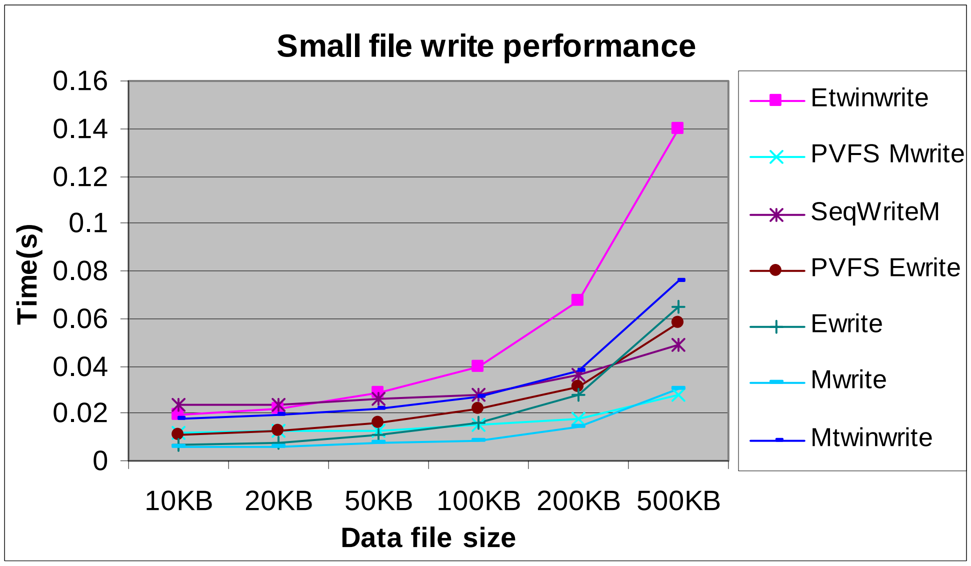

Figure 4-10 Small File Write Performance

Write performance has been separated earlier, even the file size is not big, Twin-PVFS is the lowest. And the network affects the result much.

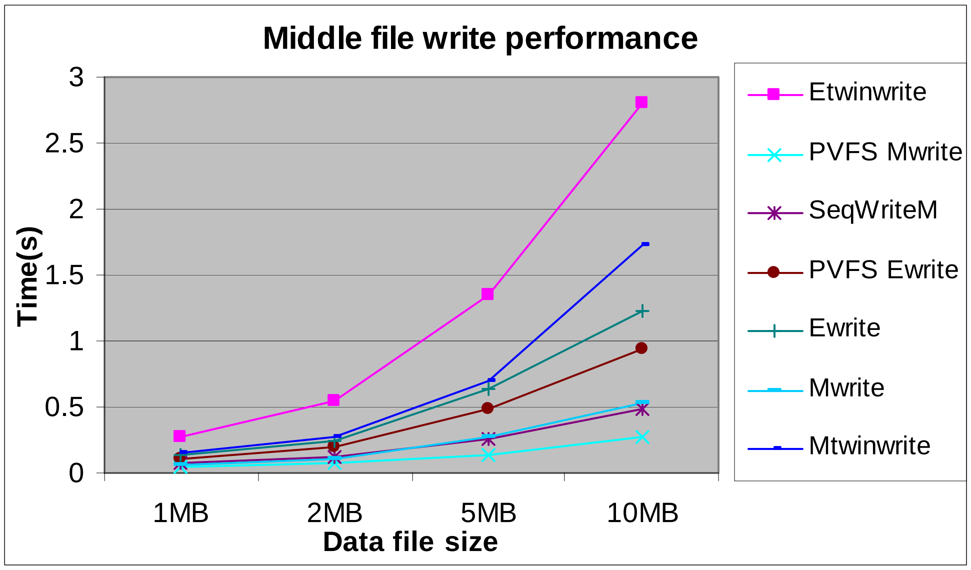

Figure 4-11 Middle File Write Performance

The results are becoming clearer. With the increase of file size, the gap between Twin-PVFS and others is enlarged.

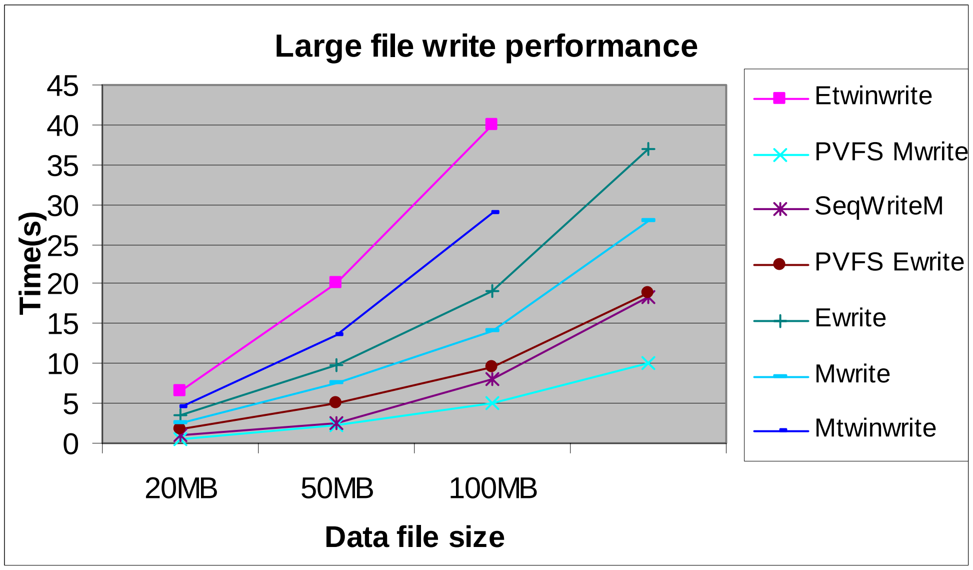

Figure 4-12 Large File Write Performance

When the file size is 200MB file, Twin-PVFS even can not finish the test, an error on SCSI disk has been showed. PVFS API still keeps the best.

4.2.3 Serial Access

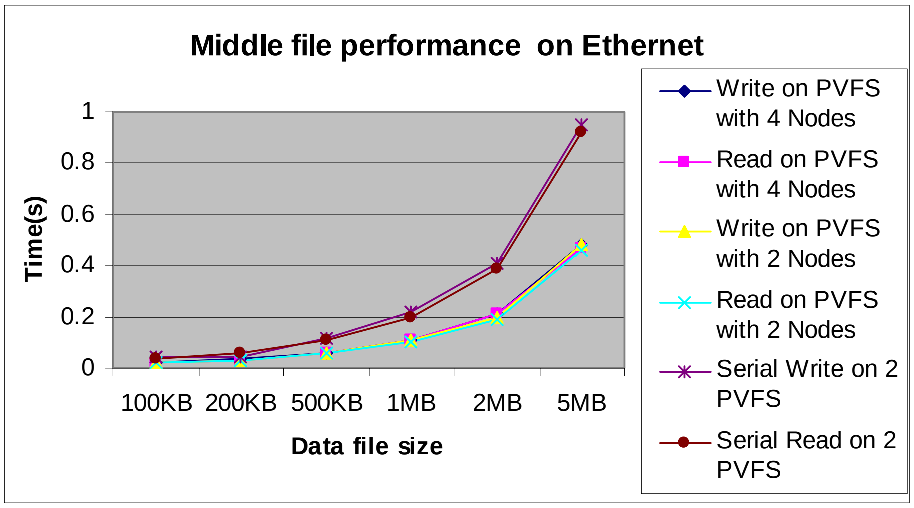

Figure 4-13 Middle file serial access performance on Ethernet

This figure shows serial access has similar performance with parallel mode. The taken time is almost twice as much time on original PVFS mode. The data amount is also twice as many. That means their performances are very close.

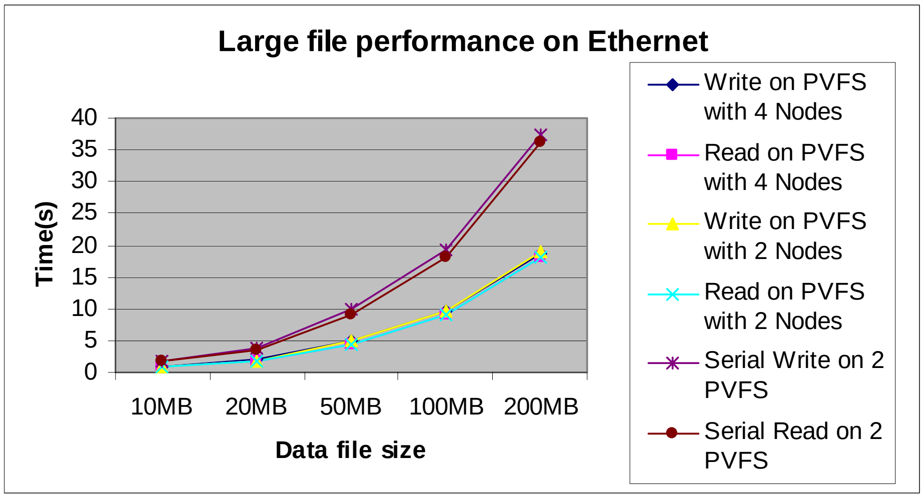

Figure 4-14 Large file serial access performance on Ethernet

The results are the same as the middle file. Parallel mode doesn't show its advantage.

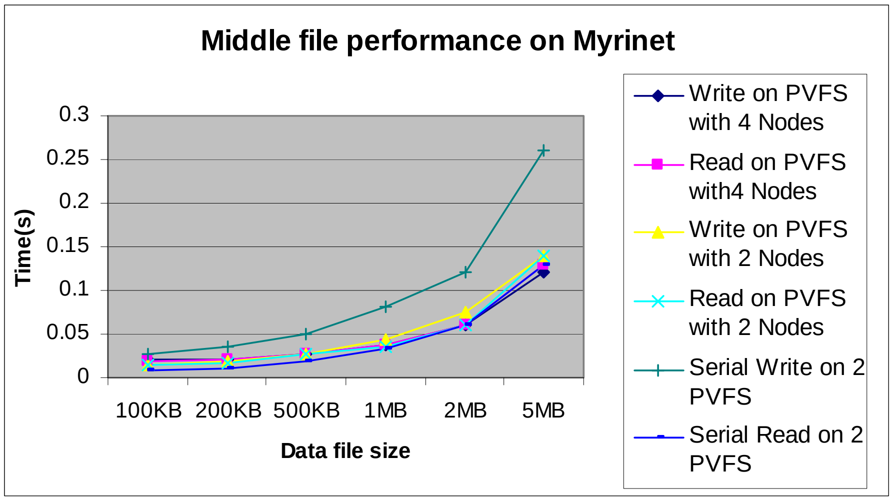

Figure 4-15 Middle file serial access performance on Myrinet

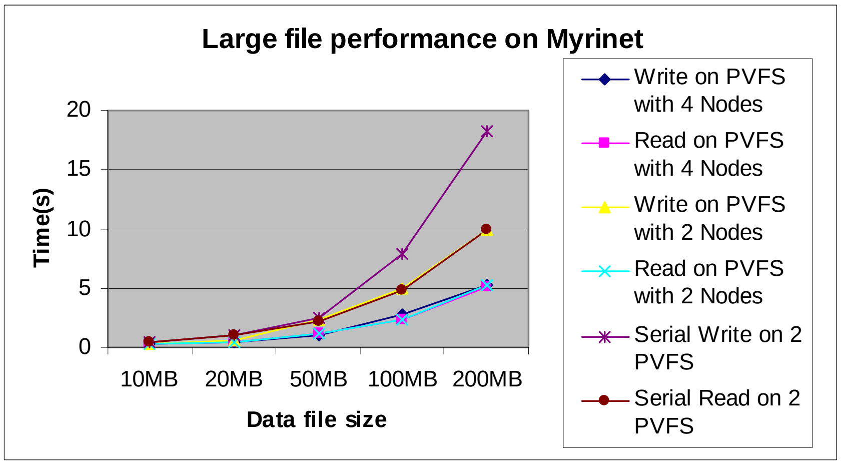

Figure 4-16 Large file serial access performance on Myrinet

On Myrinet, the results are little different. It validates again that parallel access benefits a lot from a higher bandwidth, especially the transferring data is huge.

4.2.4 Parallel Access

We also tested the PVFS API in parallel access mode with different file size.

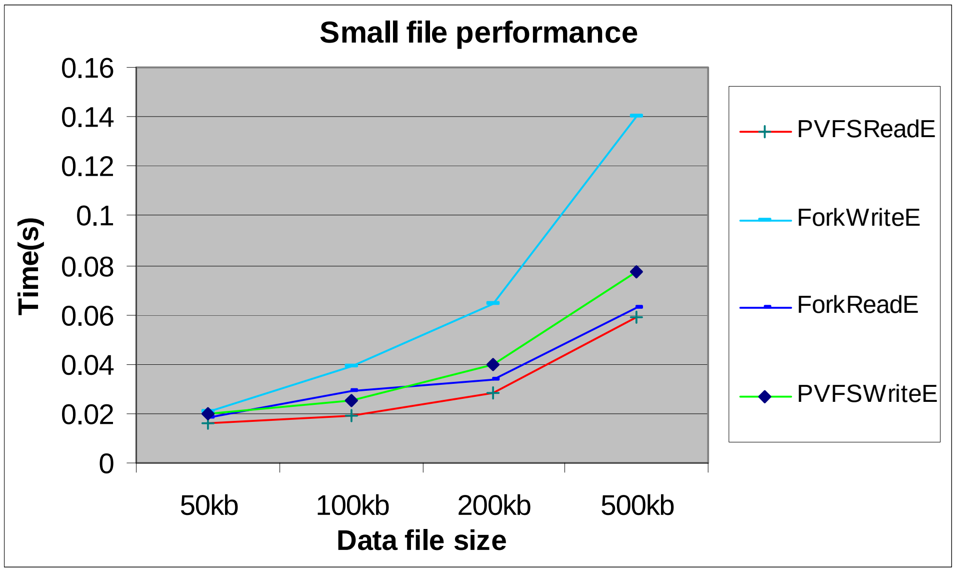

Figure 4-17 Small file performance in parallel mode on Ethernet

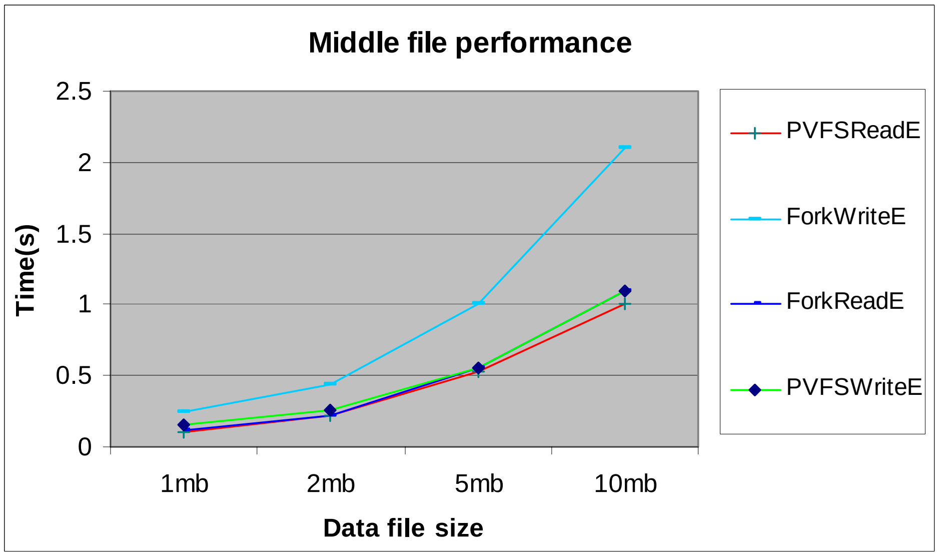

Figure 4-18 Middle file performance in parallel mode on Ethernet

In these three groups of file size, write in parallel mode performs like two write operations but parallel read is similar with the original PVFS read.

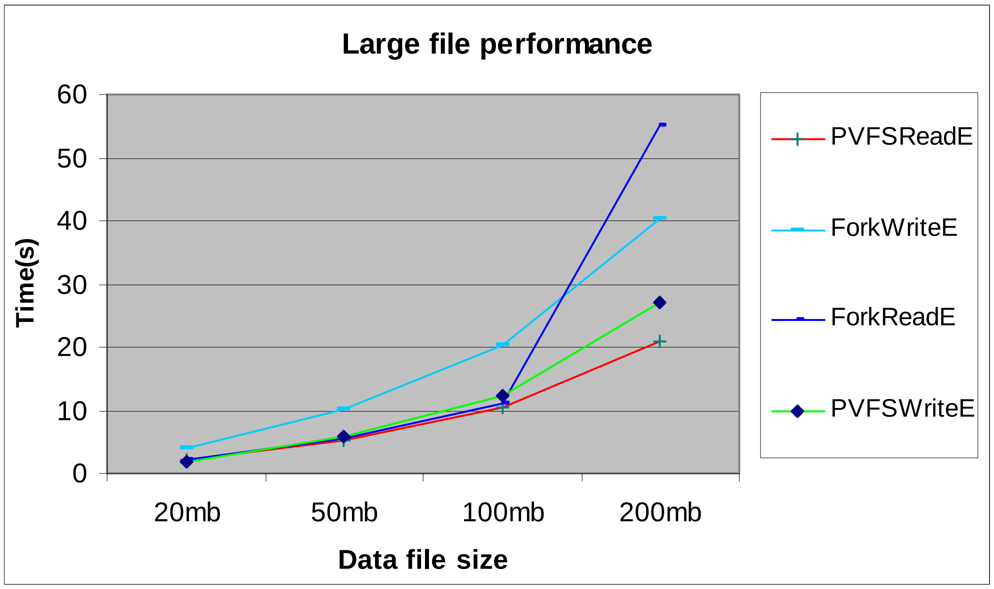

Figure 4-19 Large file performance in parallel mode on Ethernet

On Myrinet, the situation is different. This parallel access mode performs better, the gap between these two become smaller.

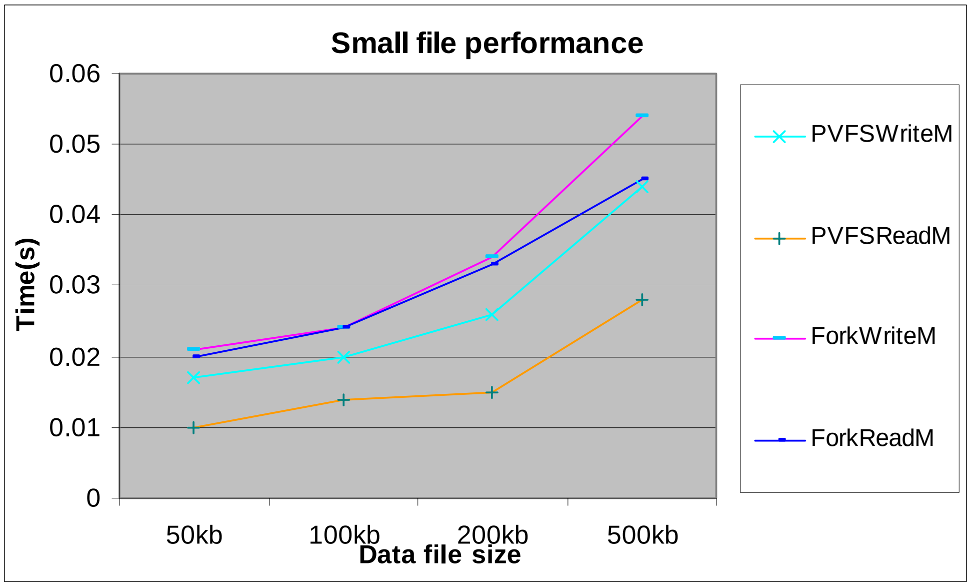

Figure 4-20 Small file performance in parallel mode on Myrinet

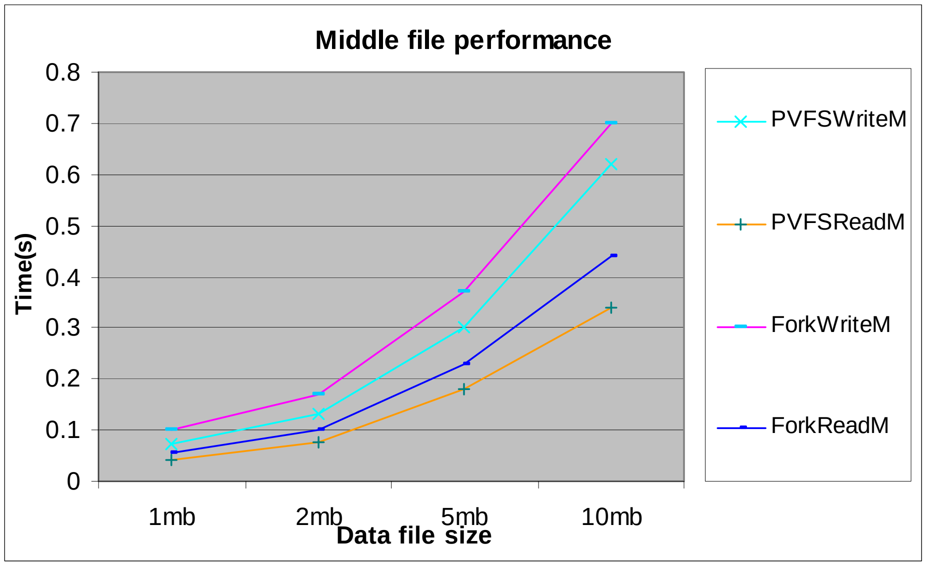

Figure 4-21 Middle file performance in parallel mode on Myrinet

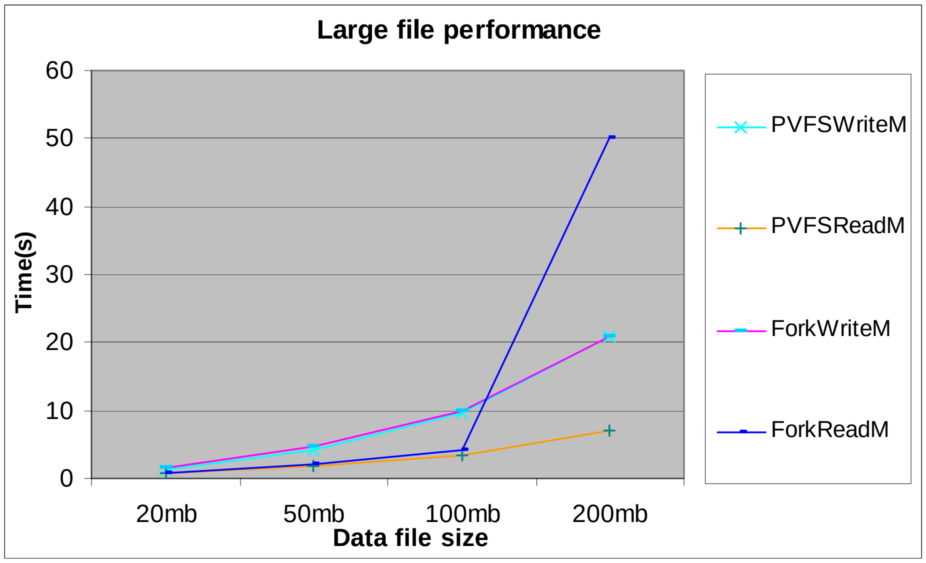

Figure 4-22 Large file performance in parallel mode on Myrinet

Even parallel write operation result is very close to the original write. But when the file size is 200MB, parallel read is busy with transferring data between memory and hard disk, a larger memory should be helpful to finish a further test.

4.2.5 Journaling

The journal only records the data and metadata. If they are changed, the journal will be updated. Distinctly, only write operation changes both data and metadata. Read only does not changes the data and metadata. Hence the journal will not be changed. Our preliminary tests also show journaling doesn't affect the read operation. In this test we only do the write operation. We test different cases, PVFS, serial access and Twin-PVFS, middle size files and large size files, Ethernet and Myrinet. The combinations of these individual elements present interesting results.

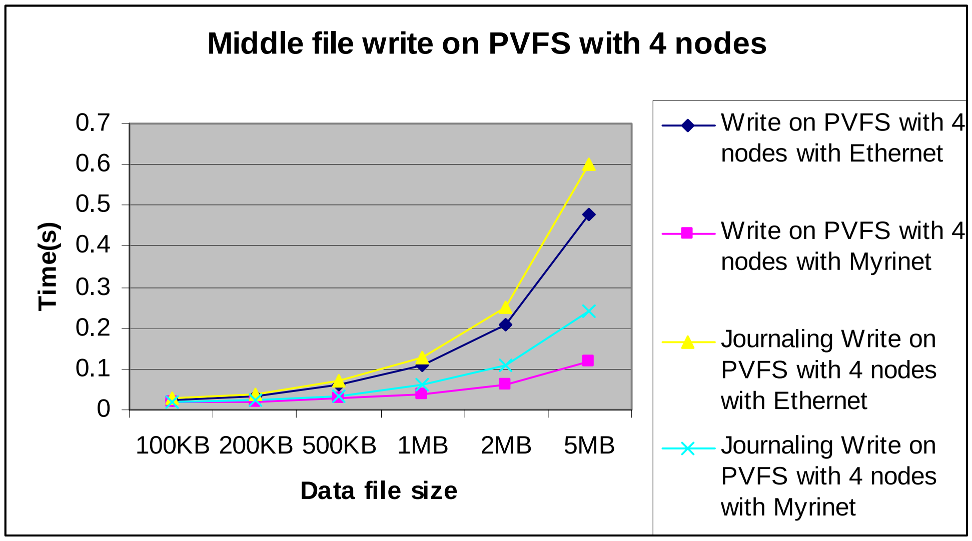

Figure 4-23 Middle file write performance on PVFS with 4 nodes

For middle files, journaling has different results. It looks like journal will slow down the system when the PVFS scale increases. A faster network can help enhance the performance.

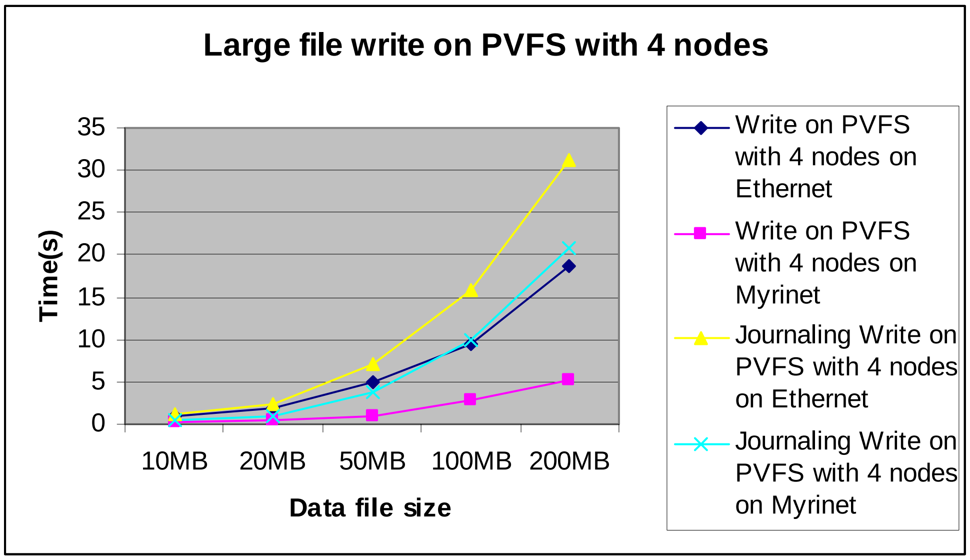

Figure 4-24 Large file write on PVFS with 4 nodes

Large file accesses causes the network jam on Ethernet, thus journaling doesn't make the performance much worse. But Myrinet separates them. On PVFS with 4 nodes, the trend keeps going.

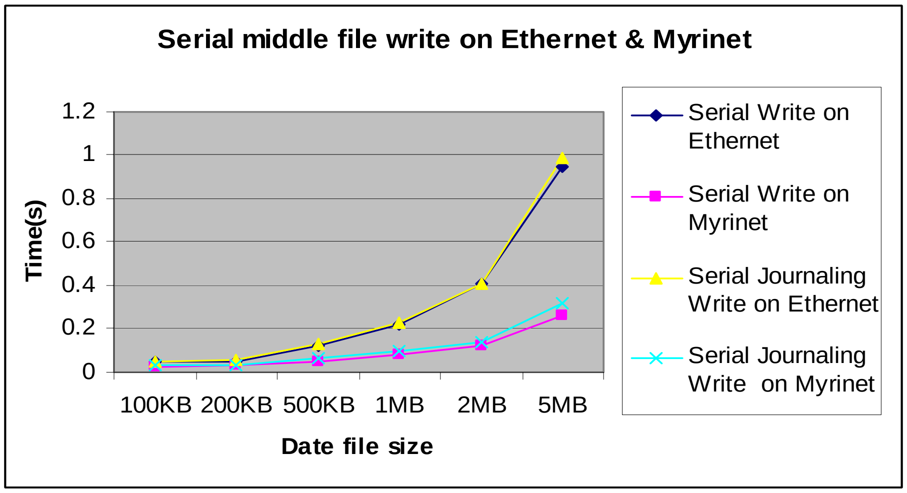

Figure 4-25 Serial middle file write

From this figure, we can see that journaling does not affect the serial performance of middle size files much. The results are very close. It looks like the local file system can handle this kind of journaling without affecting the system performance.

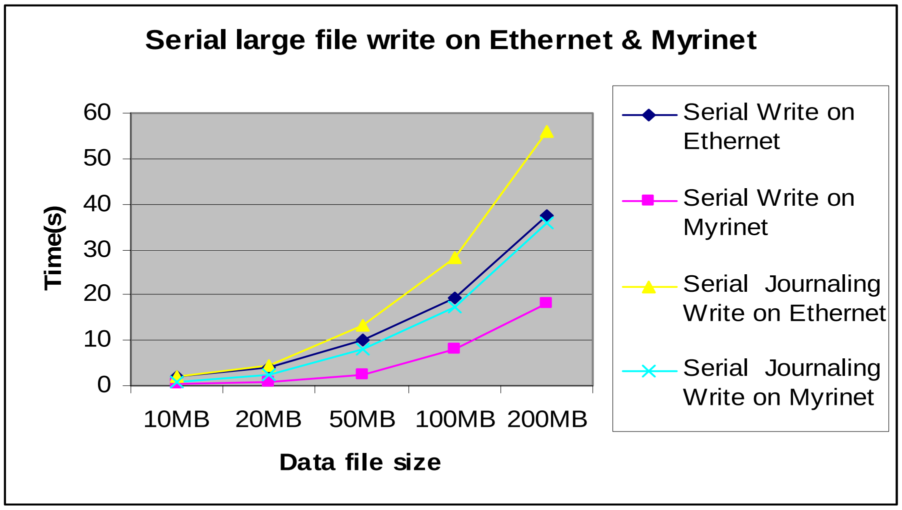

Figure 4-26 Serial large file write

For large files, it's different. Journaling slows down the system very much. Recording that size of journaling does cost much time. The influence of network bandwidth also can be found from those figures.

Journaling also can improve the system availability and also slows down the system. It's not obvious if the transferring data is small. But if the file size is calculated by dozens of MB, the system performance will drop much. An interesting result is that if we use more I/O nodes with journaling, PVFS does not perform well because its own parallel mode is serial for a single client. A file is divided into several fragments and each part will be journaled sequentially. That is a longer process.

4.2.6 Network overhead

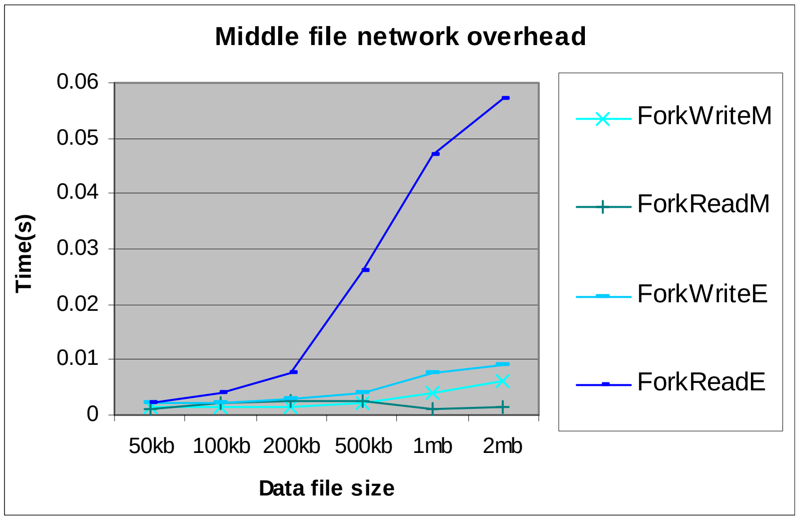

Figure 4-27 Middle file network overhead

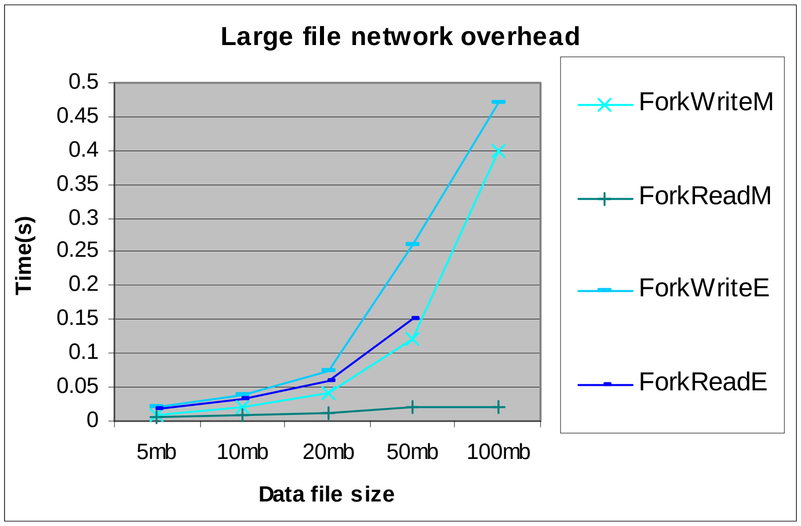

Figure 4-28 Large file network overhead

From these two figures, we can find Myrinet has less overhead, especially the file size is bigger. Since we use sync mode to write file data, it will increase the operation time to wait for the return. For the lack of memory, we didn't test big files with above 200MB size.

4.2.7 iozone

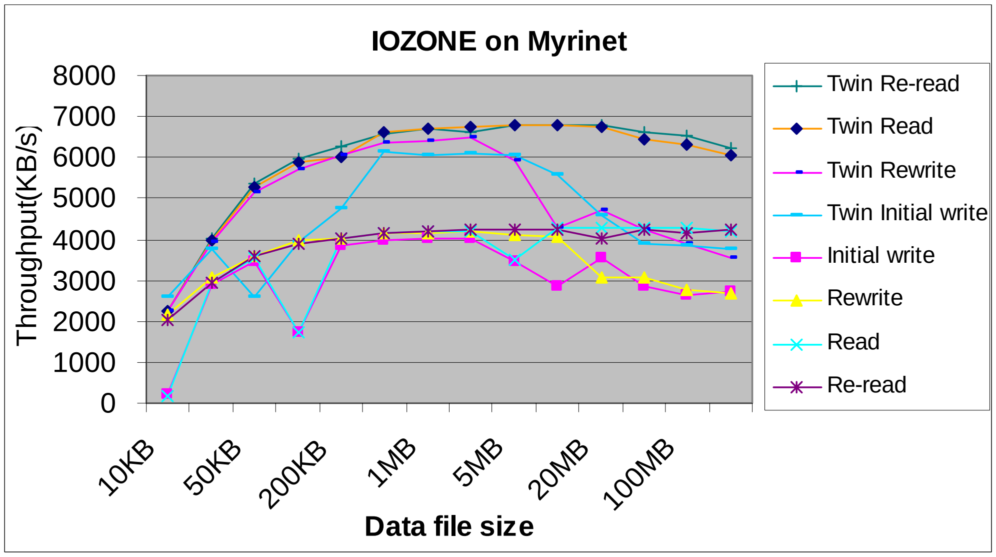

Figure 4-29 IOZONE Test on Myrinet

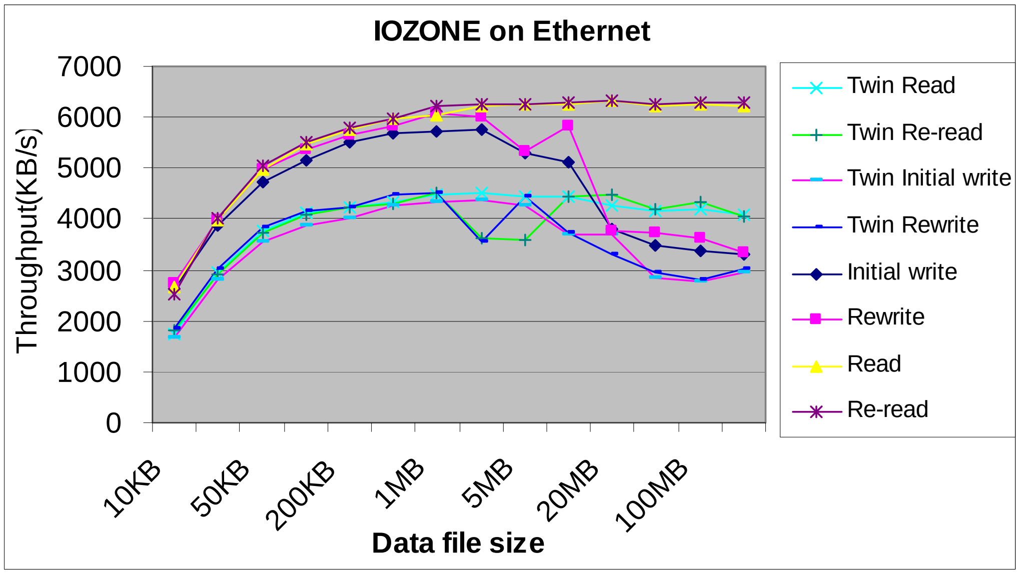

Figure 4-30 IOZONE Test on Ethernet

From this result we also can find a faster network will help improve the performance. Write performance of “Twin system” is not very good, but acceptable; its throughput is higher although part of it is not useful until the system is down. When the file is larger, write operations on both Ethernet and Myrinet dive much. During that time, hard disk becomes the bottleneck.

4.3 Results summary

From the data set above we can find that PVFS is not a good choice for such small files, less than 50KB, not only over Ethernet but also over Myrinet. That is also said in PVFS manual [33]. It relates to the stripe size of parallel file system. If the file size is bigger than 100KB, PVFS can have a good performance. Ethernet is not a good company for PVFS because its bandwidth is only 100Mb/s. Myrinet we have in this test can supply us 1Gb/s bandwidth. It is really helpful to obtain a better performance. We guess that the large files write will work well with the parallel scale's growth. There is no winner in read operations but all scores seem good.

As we forecast, our revision, i.e. the Twin-PVFS, does not win the original PVFS in all tests. We believe that it is caused by the redundancy. For a computer, if its hardware doesn't change, the maximum of computer capacity is constant, the redundancy occupies more running time and storage space, finally depress the performance. When the workload is above its maximum capacity this computer will be jammed. The PVFS kernel module is not designed to obtain a high performance because it relies on the kernel processing cost, daemon cost. Multi-thread also run on the top of kernel and can't rescue our Twin-PVFS. In a better network, read operation doesn't perform very badly, but write performance is almost unacceptable although it has redundancy. API is still the best way to achieve high performance in the system like PVFS. It’s worth doing further study in this way. The mount mode also can be kept for the convenience.

The full journaling as expected, does also slow down the system, especially the file size is big. It’s easy to understand that the local file system needs time to record a big size of journal with the full journaling mode.

In our tests, when the file size grows, the hard disk and the network become the system bottleneck. A faster network can help reduce the queue time but the system speed is controlled by the bottleneck.

Chapter 5. Conclusions

5.1 Conclusions

In this thesis, we introduce the RAID-1 Level redundancy to the PVFS and measure the effects on system preliminarily. We call this PVFS with this new function as Twin-PVFS.

Firstly we build up a cluster and connect the nodes by Ethernet and Myrinet. To evaluate the system performance, we design a series of tests. The variables include the parallel scale and the data file size required by the client. The data size varies from 1KB to 200 MB and the parallel scale is from 1 I/O node to 4 I/O nodes. We use the results of the original PVFS as the benchmark to compare our Twin-PVFS. Our Twin-PVFS has also been tested with some file data.

The results show that PVFS has a good performance on large file operations, especially when the system has high bandwidth. It even exceeds the local file write. This is the fascination of parallel file systems. Furthermore, the system performance increases someway with the parallel scale's growth. But PVFS does not accelerate the small file operations, less than 50KB, in our test environments; they even slow the system down more with the parallel scale's growth. We observe surprisingly that PVFS has similar read speed no matter what the file size is or what the parallel scale is.

As we surmise, our Twin-PVFS does not improve the system speed. On the contrary it keeps the slowest in the tests. This RAID-1 mode has good redundancy but of course it can not have a good hardware level performance. The mount-based access is not a good choice for high performance field, but it's still a convenient approach for general usage. Our results also show the network bandwidth will help the system enhance the performance.

Journaling also can improve the system availability and slows down the system. It's not obvious if the transferring data is small. But if the file size is calculated by dozens of MB, the system performance will drop much. An interesting result is that if we use more I/O nodes with journaling, PVFS does not perform well because of its parallel mode.

5.2 Future works

Limited by the experiment equipments, we only test a PVFS with 4 I/O nodes and Twin-PVFS with 4 I/O nodes. We believe that a PVFS with more I/O nodes can have a better performance. Our Twin-PVFS might still be the last one but the performance might be better because the I/O nodes finish the work faster. This assumption needs more tests to prove it. The network we adopt in this experiment is 100Mbps Ethernet and 1Gbps Myrinet. This device M2M-DUAL-SW8 is fast but it might be out of date. A new modern faster switch can help us evaluate the bandwidth factor ulteriorly.

Here is no doubt that a good way to implement the data redundancy is to add this function in this file system. But it will change the PVFS whole structure, data structure, message passing parameters and so on. Furthermore we can not evaluate the comparison between systems with and without redundancy. This work can not be done in a short time. Fortunately, the PVFS staff is developing its second edition, PVFS2 with the redundancy. But it is still the beta version [34].

Bibliography

[1] http://www.es.jamstec.go.jp/esc/eng/ES/hardware.html

[2] http://www.publicaffairs.noaa.gov/releases99/sep99/noaa99061.html

[3] http://www.quadrics.com/Quadrics/QuadricsHome.nsf/NewsByDate/106FAD951034B31680256E6600638C9B

[4] Donald J. Becher, Thomas Sterling, John E. Dorband Daniel Savarese, Udaya A. Ranawak, and Charles V. Packer. Beowulf: A parallel workstation for scientific computation. In *Proceedings of International Conference on Parallel Processing*, 1995.

[5] D. Johansen and R. van Renesse. Distributed Systems in Perspective, Distributed Open Systems, pp. 175-179 , ISBN 0-8186-4292-0, IEEE, 1994

[6] http://www.top500.org/lists/2003/11/press-release.php

[7] John M. May. Parallel I/O for High Performance Computing, Morgan Kaufmann Publishers, 2001.

[8] Maurice J.Bach. The Design of The Unix Operating System, Prentice-Hall Inc., 1986.

[9] Patterson, David A., Garth A. Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *proceedings of the 1988 ACM SIGMOD Conference on Management of Data*, pp. 109-116, June 1988.

[10] Peter M. Chen, Garth Gibson, Randy H. Katz, and David A. Patterson. An Evaluation of Redundant Arrays of Disks Using an Amdahl 5890. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1990.

[11] Jim Gray, Bob Horst, and Mark Walker. Parity Striping of Disc Arrays: Low-cost Reliable Storage with Acceptable Throughput. In *Proceedings of the 16th Very Large Database Conference*, pp. 148-160, 1990. VLDB SVI.

[12] Rajkumar Buyya. High Performance Cluster Computing, Prentice Hall PTR, 1999.

[13] Edward K.Lee and Randy H.Katz. Performance Consequences of Parity Placement in Disk Arrays. In *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, pp. 190-199, April 1991.

[14] P.M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-Performance, Reliable Secondary Storage. ACM Computing Surveys, vol. 26, no. 2, pp. 145-185, June 1994.

[15] Gould, E., and M. Xinu. The network file system implemented on 4.3 BSD, *USENIX Association Summer Conference Proceedings*, pp. 294-298, June 1986.

[16] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System For Linux Clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GA, pp. 317-327, October 2000.

[17] P. Pierce. A concurrent file system for a highly parallel mass storage system. In *Fourth Conference on Hypercube Concurrent Computers and Applications*, 1989, pp. 155-160.

[18] Intel Corporation. Paragon System User's Guide, April 1996. Includes a chapter on using PFS but has little information on its underlying design.

[19] John H. Hartman and John K. Ousterhout. The Zebra Striped Network File System. *In ACM Transactions on Computer Systems 13*, pp. 279-310, 3, August 1995.

[20] Peter C. Dibble, Michael L. Scott, and Carla Schlatter Ellis. Bridge: A High-Performance File System for Parallel Processors. In *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS)*, pp.154-161, 1988.

[21] M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Trans. on Computer Systems*, 10(1): pp. 26–52, February 1992.

[22] M. Holton and R. Das. XFS:A next generation journalled 64-bit filesystem with guaranteed rate I/O. SGICorp. http://www.sgi.com/Technology/xfs-whitepaper.html.

[23] M. Seltzer, K. Bostic, M. McKusick, and C. Staelin. An Implementation of a Log-Structured File System for UNIX. In *Proc. of the 1993 Winter USENIX*, pp. 307–326, January 1993.

[24] M. Seltzer, K. Smith, H. Balakrishnan, J. Chang, S. McMains, and V. Padmanabhan. File System Logging Versus Clustering: A Performance Comparison. In *Proc. Of the 1995 Winter USENIX*, January 1995.

[25] M. Barrios et al. *GPFS: A Parallel File System*. IBM Cor.,SG24-5165-00, 1998, http://www.redbooks.ibm.com/.

[26] Peter F. Corbett, Dror G. Feitelson, Jean-Pierre Prost, George S. Almasi, Sandra Johnson Baylor, Anthony S. Bolmarcich, Yarsun Hsu, Julian Satran, Marc Snir, Robert Colao, Brian Herr, Joseph Kavaky, Thomas R. Morgan, and Anthony Zlotek. Parallel file systems for the IBM SP computers. *IBM Systems Journal*, 34(2): pp. 222–248, January 1995.

[27] Terry Jones, Alice Koniges, R. Kim Yates. Performance of the IBM General Parallel File System. In *14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, May 2000**.**

[28] W. B. Ligon and R. B. Ross. An Overview of the Parallel Virtual File System. In *Proceedings of the 1999 Extreme Linux Workshop*, June 1999.

[29] http://www.parl.clemson.edu/pvfs/desc/desc-system-1.png

[30] http://www.parl.clemson.edu/pvfs/desc/desc-flow-meta-1.png

[31] http://www.parl.clemson.edu/pvfs/desc/desc-flow-io-1.png

[32] http://www.parl.clemson.edu/pvfs/desc/desc-kernel-path-1.png

[33] http://www.parl.clemson.edu/pvfs/user-guide.html

[34] http://www.pvfs.org/pvfs2/

[35] http://www.seagate.com/docs/pdf/marketing/Seagate_Cheetah_15K-4.pdf

[36] Y. Zhu, H. Jiang, X. Qin, D. Feng and D. Swanson. Design, Implementation, and Performance Evaluation of a Cost-Effective Fault-Tolerant Parallel Virtual File System. In *Proceedings of International Workshop on Storage Network Architecture and Parallel I/Os*. In conjunctions with *12th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, New Orleans, LA, Sept. 27 - Oct. 1, 2003.

[37] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson. Improved Read Performance in a Cost-Effective, Fault-Tolerant Parallel Virtual File System (CEFT-PVFS). In *Proceedings of IEEE/ACM Cluster Computing and Computational Grids (CCGRID)*, pp. 730-735, May 2003, Japan.

[38] Sheng-Kai Hung, Yarsun Hsu. Modularized Redundant Parallel Virtual File System. In *Asia-Pacific Computer Systems Architecture Conference*, pp. 186-199, 2005.

[39] Steven A. Moyer and V. S. Sunderam. PIOUS: A scalable parallel I/O system for distributed computing environments. In *Proceedings of the Scalable High-Performance Computing Conference*, pp. 71–78, 1994.

[40] Hakan Taki and Gil Utard. MPI-IO on a parallel file system for cluster of workstations. In *Proceedings of the IEEE Computer Society International Workshopon Cluster Computing*, pp. 150–157, Melbourne, Australia, 1999.

[41] Rosario Cristaldi, Giulio Iannello, and Francesco Delfino. The cluster file system:Integration of high performance communication and I/O in

clusters. In *Proceed of the 2nd IEEE/ACM international symposium on Cluster computing and the grid,* Berlin, Germany, may 2002.

[42] http://www.iozone.org

[43] Tim Bray.Bonnie.http://www.textuality.com/bonnie

Appendices

I. Top 10 Supercomputers, released in SC2005.

http://www.top500.org/list/2005/11/

Rank	Site	Computer	Processors	Year	R_{max}	R_{peak}
1	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution IBM	131072	2005	280600	367000
2	IBM Thomas J. Watson Research Center United States	BGW - eServer Blue Gene Solution IBM	40960	2005	91290	114688
3	DOE/NNSA/LLNL United States	ASC Purple - eServer pSeries p5 575 1.9 GHz IBM	10240	2005	63390	77824
4	NASA/Ames Research Center/NAS United States	Columbia - SGI Altix 1.5 GHz, Voltaire Infiniband SGI	10160	2004	51870	60960
5	Sandia National Laboratories United States	Thunderbird - PowerEdge 1850, 3.6 GHz, Infiniband Dell	8000	2005	38270	64512
6	Sandia National Laboratories United States	Red Storm Cray XT3, 2.0 GHz Cray Inc.	10880	2005	36190	43520
7	The Earth Simulator Center Japan	Earth-Simulator NEC	5120	2002	35860	40960
8	Barcelona Supercomputer Center Spain	MareNostrum - JS20 Cluster, PPC 970, 2.2 GHz, Myrinet IBM	4800	2005	27910	42144
9	ASTRON/University Groningen Netherlands	Stella - eServer Blue Gene Solution IBM	12288	2005	27450	34406.4
10	Oak Ridge National Laboratory United States	Jaguar - Cray XT3, 2.4 GHz Cray Inc.	5200	2005	20527	24960