Reliable Parallel File System Using RAID Technology

Sheng-Kai Hung
Department of Electrical Engineering
National Tsing-Hua University
phinex@hpcc.ee.nthu.edu.tw

Yarsun Hsu
Department of Electrical Engineering
National Tsing-Hua University
yshsu@ee.nthu.edu.tw

Abstract

Providing data avaliability in a cluster environment is very important. Most clusters either use RAID technology or redudant nodes to ensure reliability. High performance clusters usually use parallel file systems to increase throughput. However, when parallel file system is involved in a cluster system, some mechanism must be provided to overcome the side-effect of using striping. PVFS is a popular and open source parallel file system implementation in the Linux environment, but provides no fault tolerence. We propose an idea of using distributed RAID technology to ensure the reliability of using striping. We also introduce a concept called delay write to overcome the write performance penalty incurred in the traditional RAID technology. The evaluation of our RPVFS(Reliable Parallel Virtual File System) shows that the read performance is almost the same when compared with that of original PVFS design. As to the write performance. there only exists a little performance degradation.

keyword: *Reliable Parallel Virtual File System, Redundent Arrays of Inexpensive Disks, Cluster, Delay Write*

1 Introduction

As the cluster becomes a vehicle for low-cost and high-performance computing environment, using clusters to solve scientific problems has become a useful technique. Where there are off the shelf commodity hardwares, there are clusters. Linux operating system with Intel x86 machine becomes a new platform for implementing cluster systems. Many freely distributed software packages such as TCP/IP networking and message passing interface have also been matured and ported to support Linux clusters. People learn to build clusters, use them, and tune their performance. However, one area lack of support is the parallel file system, especially for Linux platform. In the past, parallel file system was a proprietary product belonging to specific commercial machines, such as PFS to Intel Paragon[1], VESTA[2] to IBM SP2 machine. Due to lacking support of a parallel file system, most Linux cluster can just use network file system (NFS[3]) to perform I/O access and guarantee a single view of the file system. However, NFS is notorious for its performance, and not a good solution for I/O intensive operations.

To the best of our knowledge, PVFS[4] is a practical remedy for providing high performance I/O in the Linux environment. It is good at both providing high throughput I/O and offering an interface to the legacy applications. However, experience of using PVFS tells us that its reliability is a big problem even if each node's disks in the cluster use RAID technology[5]. This is because PVFS uses a RAID-0 like strip mechanism across different nodes to enhance I/O performance. Striping is a good technique to allow parallel accesses, but lacking reliability support. Using striping without any fault tolerance technique, the MTTF(Mean Time To Failure) of PVFS may be lowered by a factor of $\frac{1}{N}$, where N is the number of I/O nodes in the cluster system. In this situation, what will happen if a disk in one of the nodes fails in the PVFS file system? All data in the parallel file system can not be accessed again unless the node comes back to work. Things become even worse since not only disk failures must be considered but also other components of each node in the cluster may cause a node to fail.

The reliability of PVFS can be improved by purchasing additional disks and using either hardware RAID controllers or software RAID technique[6]. These techniques have two disadvantages, one is the cost incurred, and the other is the MTTR (Mean Time To Repair) of the

This work was supported by National Science Council, ROC under Grants NSC 93-2752-E-007-PAE and NSC 92-2213-E-007-052

overall parallel file system. The first one is obvious and all of us tend to reduce the cost of constructing a COTS(Commodity Off The Shelf) cluster system. The second one can be thought of the failure of nodes. Although using RAID can provide a better MTTF, this does not help a lot. The lifetime of a computer component, like disks nowadays, is shorter than we could expect. We often upgrade a computer system before its components are broken. The availability of a system can be expressed as $\frac{MTTF}{MTTF+MTTR}$, this turns our attention to reduce the MTTR. If one of the nodes in a cluster system using PVFS fails, even it uses RAID technology, the striped data within that node can not be accessed unless we replace the broken devices with new ones. Using RAID in each node can just only guarantee the availability of data within each node, it provides no ideas as far as the total file system is concerned. The solution to this condition may use some redundant nodes, and when detecting failures the redundant nodes can just be used instead. However, even use redundant nodes, there still exists detectable MTTR in the total file system. In some cases, especially for some server farms, 99.999% availability[7] is required and we must ensure that there only can be a 5 minute down time in a year. Since fault can not be avoided, providing a system with low MTTR is a requirement to guarantee a 99.999% availability.

In order to provide the data availability of a Linux cluster using parallel file system, we propose a distributed RAID technique which can both increase the MTTF and reduce the MTTR of the overall I/O system. The reliable PVFS file system can not only provide low cost compared with that using RAID at each node, but also provide the ability to repair failure to improve the availability. RAID-4 technique is used in our design to provide high read throughputs. As to the write problem incurred in RAID-4, we provide a mechanism called delay write. By using delay write, the write performance can be improved. We believe that this is a low cost solution and could be done in the Linux cluster very well. This paper is organized as follows: the related research topics are covered in section 2; a quantitative analysis of the overall availability will be presented in section 3; we will show the implementation and evaluation of our reliable parallel file system versus the original PVFS in section 4; finally the future work and conclusions are discussed in section 5.

2 Related Researches

In this section, we will present some file systems used in the cluster environment, either distributed or parallel one. Sun Microsystems' NFS(network file system) is widely used in the traditional UNIX environment, but it lacks support of parallel semantics and the server node is always a single point of failure. It provides neither fault tolerance mechanism nor striping technique. The well known feature of NFS is its easy use. However, it's performance is also notorious when serving many I/O clients. As we know, some of the clusters in the world still use NFS as their file system and depend on MPI-IO for parallel access support.

Swift[8] , like PVFS provides conventional file access semantics in its striping structure. It also supports the same parity method used in RAID level 2 or higher. By the use of parity disks, error can be recovered as long as the error is not happened at the metadata server. Like Swift, Zebra[9] is also such a file system. These two file systems may be called the ancestors of strip file system across different I/O nodes. Berkeley's xFS[10] decentralizes Zebra file system and makes the global file system cache available. xFS was a good research project, but did not have further development due to some license restriction. To avoid the failures of nodes, GPFS[11] and GFS[12] connect disk storages and clients by interconnection switch. This makes the concept of servers disappeared and eliminates the failures caused by servers. However, these proprietary hardwares cost more and can not be purchased in the COTS clusters.

CEFT-PVFS[13], like PIOUS[14] and Petal[15] provides RAID-1 like technique. Since CEFT-PVFS is based on PVFS and implements RAID-1 on it, CEFT-PVFS can be regarded as a RAID-10 like parallel file system. Although CEFT-PVFS extends the original PVFS design and provides the fault-tolerance mechanism, it has some problems. The first is the consistency problem. CEFT-PVFS uses an additional RAID-1 technique over the original PVFS RAID-0 design. By doing so, it needs to guarantee data consistency between working nodes and standby nodes. The second is incurred by the first. Unlike RAID controllers which use bus to transfer data to individual disks, distributed RAID technique relies on the network to transfer data into individual nodes. However, network resource is an important factor that would impact the performance of parallel applications. Using CEFT-PVFS, the network bandwidth would be consumed to some extent. Furthermore, more disk space is needed when using CEFT-PVFS. This is because the original RAID-1 design uses mirroring, and it wastes 50% of disk space. This may not be a big problem since the price of disk nowadays is cheaper than that before.

3 Quantitative Analysis

In this section, we provide the MTTF of three kinds of mechanisms used when PVFS is involved. There are: PVFS without any redundant technique ($MTTF_{PVFS}$), PVFS with RAID technique in each node ($MTTF_{PRAID}$), and our reliable PVFS ($MTTF_{RPVFS}$).

Here, we must define some terms in advance. Let $MTTF_D$ denote the mean time to failure of a single disk; N denote the number of nodes in a cluster; $MTTF_{RAID}$ express the MTTF of RAID disks and $MTTF_S$ indicate the MTTF of other components of a single node in the cluster system.

Since $MTTF_{PVFS}$ just uses striping without any fault tolerance mechanism, the MTTF of this type can be expressed as

$$MTTF_{PVFS} = 1/(\frac{N}{MTTF_D} + \frac{N}{MTTF_S})$$

By the analysis of RAID technique in the paper[5], we can know that the $MTTF$ of using RAID disks can be expressed as

$$MTTF_{PRAID} = 1/(\frac{N}{MTTF_{RAID}} + \frac{N}{MTTF_S})$$

$$MTTF_{RAID} = \frac{(MTTF_D)^2}{(D + C *^n G) * (G + C - 1) * MTTR}$$

- D is the total number of data disks
- G means the data disks in a group
- C is the number of check disks *(disks contain parity information)* in a group
- ${}^nG$ denotes the number of groups *(each group has a check disk at least)*

In a cluster system using distributed RAID-4 technique, each I/O node can be used as a parity node or a data node. This tells us that $D + C *^n G = (G + C) *^n G = N$. Using this equation we can rewrite the above equation as.

$$MTTF_{RAID} = \frac{(MTTF_D)^2}{N * (\frac{N}{^nG} - 1) * MTTR}$$

Thus we can express $MTTF_{PRAID}$ as,

$$\begin{aligned}
& 1/(\frac{N^2 \times (\frac{N}{^nG} - 1) \times MTTR}{(MTTF_D)^2} + \frac{N}{MTTF_S}) \\
= \; & 1/(\frac{N^2 \times (\frac{N}{^nG} - 1) \times MTTR}{(MTTF_D)^2} \\
& \qquad + \frac{1}{MTTF_{PVFS}} - \frac{N}{MTTF_D}) \\
\leq \; & 1/(\frac{1}{MTTF_{PVFS}} - \frac{N}{MTTF_D}) \\
= \; & \frac{MTTF_{PVFS} * MTTF_D}{MTTF_D - N * MTTF_{PVFS}} \\
= \; & \frac{MTTF_D}{MTTF_D - N * MTTF_{PVFS}} \times MTTF_{PVFS}
\end{aligned}$$

Besides, by using $MTTF_{RAID}$ shown in the above, we can also get the MTTF of our reliable parallel virtual file system, which can be shown in the following equation.

$$\begin{aligned}
MTTF_{RPVFS} &= \frac{(MTTF_{Node})^2}{N * (\frac{N}{^nG} - 1) * MTTR} \\
&= \frac{\left(\frac{1}{\frac{1}{MTTF_D} + \frac{1}{MTTF_S}}\right)^2}{N * (\frac{N}{^nG} - 1) * MTTR}
\end{aligned}$$

Using the above two equations, we can obtain

$$\begin{aligned}
MTTF_{RPVFS} &= \frac{N^2 * (MTTF_{PVFS})^2}{N * (\frac{N}{^nG} - 1) * MTTR} \\
&= \frac{N * {}^nG}{(N - {}^nG)} \times \frac{(MTTF_{PVFS})^2}{MTTR} \\
&\geq \frac{N * {}^nG}{N} \times \frac{(MTTF_{PVFS})^2}{MTTR} \\
&= {}^nG \times \frac{(MTTF_{PVFS})^2}{MTTR}
\end{aligned}$$

In the real case MTTF or MTTR is usually measured by hours. If we could make MTTR as small as possible(such as 1 hour), we could rewrite the above equation as.

$$MTTF_{RPVFS} \geq {}^nG * MTTF_{PVFS}^2$$

This means that our reliable parallel virtual file system(RPVFS) can provide a much better MTTF than the original PVFS. Besides, we can also increase ${}^nG$ to improve the $MTTF$ of our $RPVFS$ more. Increasing ${}^nG$ is helpful when the number of nodes in a cluster becomes larger and lager since $MTTF_{PVFS}$ is inversely proportional to the number of nodes in a cluster system.

4 Implementation and Evaluation

As an extension module of PVFS, we implement the fault tolerance mechanism within the PVFS. This implementation includes three parts: parity striping, fault detection, and on-line recovery. We will discuss them in this section. The original striping mechanism of PVFS does not support parity, just like a RAID level 0. It is easier to implement RAID 10 (RAID 0 + RAID 1) in PVFS just like CEFT-PVFS did, but its disk efficiency is limited to just 50%. RAID 5 technology has been proven as the best trade-offs between performance and efficiency, and naturally be taken into our consideration when implementing RPVFS. However, to simplify our design we use RAID 4 instead. The difference between RAID-5 and RAID-4 is only the write performance, especially for concurrent write to the parity disk. This problem can be solved in our design because of the use of delay write. We would describe the delay write in the following paragraph.

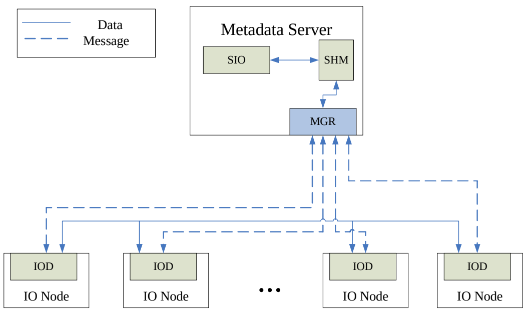

Figure 1: Normal Operation

Figure 1 shows the normal operation of our modified PVFS. In our RPVFS, the metadata server is used as a spare io node. The spare io node stores the parity information calculated whenever a file is created in the system. The SIO means the spare io daemon which comes from the original IOD daemon. SIO is the modified version of IOD to let the MGR daemon directly communicate with the SIO without going through the Ethernet. Besides, we must guarantee that the spare io is just used to store the parity information and can not communicate with other IOD daemons in the cluster system. The SHM(SHare Memory) block in figure 1 is the mechanism we mentioned before, called delay write. In a RAID technology with parity support, each write process must recalculate the parity and thus need to read the whole data set and rewrite the new data along with parity. If several writes come together, the process of recalculating and rewriting parity is not necessary, since the newly written one overwrites the old one. In our design, whenever a write operation is performed in the client node, the corresponding io node would send the difference of data (the result of XOR) to the metadata server to indicate that the parity must be recalculated. The metadata server would store the data getting from the io nodes in the share memory region(SHM block). The metadata server would perform parity recalculation whenever there is an io node failure or the corresponding io node calls a file close function. Using this technique can have two advantages. The first is the protection of data loss between successive writes, since each information is stored in the share memory between every write. The second is the number of parity recalculation needed could be lowered. This in turn saves the computing power in the metadata server.

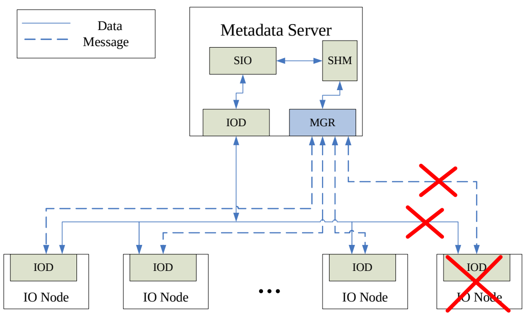

Figure 2: Failure Operation

Whenever there is a node failure, the metadata server would start a normal IOD daemon within it and regards it as the substitution of the broken one. All file operations remain and the io clients would not see any difference. Figure 2 shows the operation of this condition. The reconstruction of the broken data could be performed whenever the data is needed or when the failure is discovered by the metadata server. We call the former as passive recovery and the latter as active recovery. Using passive recovery would save the computing power in the metadata server but may increase the response time and the MTTR. With active recovery, the metadata server may be busy with reconstructing the broken data when a fault is discovered, but the overall MTTR could be lower. To simplify our design, we use passive recovery method in the current study.

Table 1 indicates the platform hardwares used in the evaluation experiments. Each node has a single AMD Athlon XP 2400+, except for the metadata server node. The metadata server node has a dual AMD Athlon XP 2400+ inside. The operating system is Redhat Linux 7.3 with kernel version 2.4.20 in each node. In our evaluation platform, eight io nodes are used with a single io client. Each test is based on the traditional TCP/IP network with 100 Mbps Ethernet card. We use the file system benchmark bigbonnie[16] to perform the test, each test use traditional UNIX file semantics. During the tests, data size is increased from 1 MB to 8192MB, the striping size is based on the data size and can be computed as $\frac{\text{Data Size}}{8}$.

Table 1: Evaluation Platform

	Disk	Memory
Metadata	160G SCSI	1 GB
IO node	4G IDE	512 MB
Client node	4G IDE	512 MB

Figure 3 shows the performance evaluation of read performance. During this test, we measure the block read performance of each of the file systems. From the figure, we know that the read performance is almost the same either in the original PVFS design or in our RPVFS. The CPU utilization of the test can be shown in Figure 4. The higher utilization of RPVFS may be caused by the modified MGR daemon, which needs to detect node failures and starts the new IOD daemon when failure occurs.

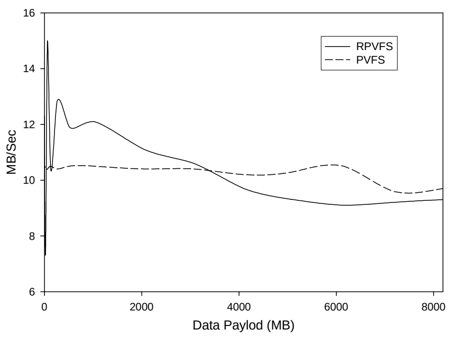

Figure 3: Read Performance

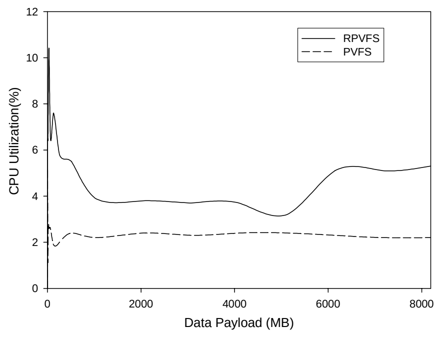

Figure 4: CPU Ultilization when Reading

Figure 5 is the result of block write performance test. In this figure, we compare the write bandwidth of original PVFS, our RPVFS without delay write, and RPVFS with delay write, denoting as PVFS, RPVFS and RPVFSDW respectively. From the data we get, when data payload is smaller than 512 MB, the results show that the write bandwidth of RPVFS with delay write is higher than that of PVFS. This may be caused by the buffer effect of each io node, since each io node has a 512 MB memory within it.

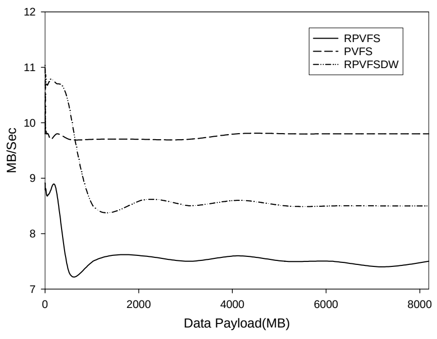

Figure 5: Write Performance

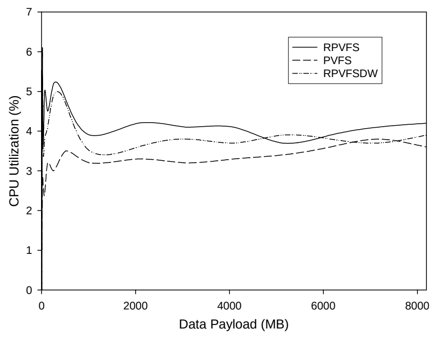

Figure 6: Utilization of Block Write

The CPU utilization of block write is also measured. As we can expect, PVFS needs the least CPU power, RPVFSDW is the next and RPVFS needs more computation power in order to calculate the parity each time when data is altered. These can be shown in the figure 6.

5 Conclusions and Future Work

We have successfully designed and implemented a reliable parallel virtual file system (RPVFS) which can continue to supply data to a requesting client even if one of the IO nodes fails due to any reason. Our RPVFS design can provide almost the same read performance compared with

PVFS. As to the write performance, with the concept of delay write, the write performance is also acceptable. When compared with CEFT-PVFS, the disk efficiency of our RPVFS is much better, and we do not have the consistency problem. However, our current design has a single point of failure, that is the metadata server as in PVFS version 1. We do not provide any fault tolerance mechanism in the metadata server node. The reason is that we can just use a mirroring node to continuously backup the data of metadata server. And if errors happen in the metadata server, the backup one can be used instead. The mirroring technique is widely used in today's cluster system, especially for server node. Using a single mirroring node has little consistency problem and we think this is a simple solution.

We plan to use distributed technique to overcome the single point of failure mentioned before. If this could be done, we can also implement RAID-5 in our RPVFS and compare its write performance with our RAID-4 based delay write one. We also plan to implement active recovery method and evaluate its impact on performance in the future.

References

[1] Intel Supercomputer System Division, *Paragon User's Guide*, Jun 1994.

[2] S. J. Baylor P. F. Corbett and D. G. Feitelson, "Overview of the vesta parallel file system", in *IPPS '93 Workshop on I/O in Parallel Computer System*, April 1993, pp. 1–16.

[3] S. Kleiman D. Walsh R. Sandberg, D. Goldberg and B. Lyon, "Design and implementation of the sun network filesystem", in *Proc. Summer USENIX Technical Conf.*, 1985, pp. 119–130.

[4] Robert B. Ross Philip H. Carns, Walter B. Ligon III and Rajeev Thakur, "PVFS: A parallel le system for linux clusters", in *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta GA, 2000, pp. 317 – 327, USENIX Association.

[5] Garth Gibson. Randy H. Katz David A. Patterson, "A case for redundant arrays of inexpensive disks (raid)", in *International Conference on Management of Data (SIGMOD)*, 1988, pp. 109–116.

[6] Jenwei Hsieh, Christopher Stanton, and Rizwan Ali, "Performance evaluation of software RAID vs. hardware RAID for parallel virtual file system", in *9th International Conference on Parallel and Distributed Systems*, December 2002, pp. 307–313.

[7] Brown. A. and D. A. Patterson, "Embracing failure: A case for recovery-oriented computing (ROC)", in *2001 High Performance Transaction Processing Symposium*, Asilomar, CA, October 2001.

[8] Luis-Felipe Cabrera and Darrel D. E. Long, "Swift: Using distributed disk striping to provide high I/O data rates", *Computing Systems*, vol. 4, no. 4, 1991.

[9] John H.Hartman and John K.Ousterhout, "The zebra striped network le system", in *Proceedings of the Fourteenth ACM Sympo-Sium on Operating Systems Principles*, 1993, pp. 29–43.

[10] Curtis Anderson Wei Hu Mike Nishimoto Adam Sweeney, Doug Doucette and Geo Peck, "Scalability in the xfs file system", in *USENIX 1996 Annual Technical Conference*, 1996.

[11] Frank Schmuck and Roger Haskin, "GPFS: A shared-disk file system for large computing clusters", in *Proceedings of the Conference on File and Storage Technologies (FAST '02)*, Jan. 2002, pp. 231–244.

[12] K. Preslan M. O'Keefe S. Soltis, G. Erickson and T. Ruwart, "The global file system: A system for shared disk storage", http://citeseer.nj.nec.com/soltis97global.html, 1997.

[13] Yifeng Zhu, Hong Jiang, Xiao Qin, Dan Feng, and David R. Swanson, "Design, implementation and performance evaluation of a cost-effective, fault-tolerance parallel virtual file system", in *International Workshop on Storage Network Architecture and Parallel I/O*, Sept. 2003.

[14] S. 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*, 1994, pp. 71–78.

[15] Edward K. Lee and Chandramohan A. Thekkath, "Petal: Distributed virtual disks", in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996, pp. 84 – 92.

[16] Textuality, "Bonnie", http://www.textuality.com/bonnie/.