Appears in the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2003), August 27-29, 2003,Chengdu, PR China

DCFS: File Service in Commodity Cluster Dawning4000

Jin He, Jin Xiong, Sining Wu,Yi Lu, Dan Meng
National Research Center for Intelligent Computing Systems
Institute of Computing Technology
Chinese Academy of Sciences, Beijing 100080, China
Email: mailto:{jhe, xj, wsn, luyi, md}@ncic.ac.cn Web: http://www.ncic.ac.cn/

Abstract[-] This paper introduces the design and implementation of DCFS (Dawning Cluster File System), a cluster file system developed for the Dawning4000 LINUX clusters. DCFS provides the single name space and binary compatibility with local file system for users and applications on Dawning4000. In addition, it improves system manageability for administrators by embedded file system management considerations into DCFS implementation and provides an easy use interface. Important features of DCFS are presented in this paper, including a global management domain, a communication abstraction layer, pipelined file I/O, enhanced multi-task capability in storage servers, and fault-tolerance strategy.

Keywords: Cluster file system, Cluster system

1 Introduction

Building clusters with commodity components has been the most popular way to get platforms for high-performance computing and information services[6]. However, this method challenges the I/O subsystems of clusters on reducing the management cost and improving the performance. Although NFS can simplify the management, the defect of the architecture makes it unable to scale. Classic superservers or clusters built with commodity components, such as IBM SP[2] and DEC VAXCluster, both provide their special file systems with single system image (GPFS[2,5] on SP and Frangipani[10] on VAXCluster) to reduce the I/O management cost and improve performance.

The Dawning4000 is a LINUX cluster. In order to resolve the above I/O problem, we designed a cluster file system, DCFS, which provides the following features:

- Single system image: DCFS is a global file system sharing among all nodes. It provides a single uniform name space and system call interface. They can read/write DCFS files without knowing the actual location of the files.
- Single management domain: DCFS provides an integrated management utility, which enables administrators to perform management tasks from any node of the system.
- High availability: DCFS can detect component failure by itself, and will isolate that component in order to make as little impact on other parts of DCFS as possible.
- Ability to run on multiple communication protocols: DCFS encapsulates its communication needs into a communication abstraction layer. Such design enables DCFS not only to use TCP for communications, but also to make use of high performance communication protocols, such as VIA.
- Conventional programming interface: The VFS based implementation makes the applications based on local file system can run on DCFS without any modifications.

The remainder of this paper is organized as follows. Section 2 describes the design issues of DCFS. Section 3 details the implementation of DCFS. Section 4 gives the preliminary performance results of DCFS. Section 5 concludes the paper and outlines our plans for future work.

2 Design Issues

During the design phase of DCFS, we elaborately analyzed the characteristics of classic parallel file systems, distributed file systems and cluster file systems. We studied some famous file systems, such as NFS[1,8], PVFS[7] and GPFS[5]. The following sections will present our design decisions for DCFS.

2.1 Scalable file service

In cluster environment, a single file server like NFS server is prone to be the bottleneck of entire system with the growth of applications' scale. Although a group of standalone file servers can ease this situation, such a system is difficult to maintain and manage. Cluster file systems with single system image can provide single name space, single management domain and UNIX interface. More important, cluster file systems with special design can provide scalable file service as following.

DCFS splits the file service into two orthogonal parts: operations on file contents and operations on file or filesystem metadata. Metadata servers provide storage and maintenance for file attributes, directories and the superblock., while Data storage servers are responsible for actual file I/O and for interfacing with storage devices. In DCFS, metadata and data storage servers store data in ordinary files of the local file system such as EXT2, ReiserFS, etc. In DCFS, data of a file is striped across data storage servers in a style like RAID 0. This approach is similar to that of

[-] This work was supported by the National High-Technology Research and Development Program of China (863 Program) under the grant No.2002AA104410.

PVFS[7]. However, there is only one metadata server (MGR) in PVFS, while DCFS support multiple metadata servers in one file system. DCFS should be more scalable in situations that there are very large number of operations on file attributes and directories, because workload can be dispatched to a group of collaborative metadata servers.

2.2 Reliable file service

As the centered data depository of a cluster system, a cluster file system should provide high performance and reliable file service for critical applications. One of the most important design decision of traditional NFS(not including v4) is that keep no or as little as possible client status in server side. By this way, NFS can easily recovered from disasters such as network fault, server or client down or reboot.

Unlike NFS, DCFS servers keep client status, and it has to carefully deal with problems that are brought up when some components of DCFS fail. These efforts includes how to detect failures, how to inform failures and how to recover DCFS to a consistent state.

2.3 Manageable file service

Management of cluster files systems is notoriously inconvenient Cluster file systems should be as easily managed as other software components of clusters. We improve the manageability in two respects: configuration and adjustment.

Table 1. Important configurable attributes in DCFS

Name	Meaning
LV number	The number of logical volume in system.
LV svrinfo	The location and type information of the servers which belong to the logical volume
LV mntinfo	Information of the clients that have been mounted this logical volume.
LV diskinfo	Information of the physical disks' partition attached to storage servers in the logical volume.
Server state	The state of server, location and type. For meta-data server, additional information includes the meta-data it managed; For storage server information includes the file data that belongs to one logical-volume.
Client state	The state of one client, including the logical volumes that have been mounted in this client.

To be configurable, cluster file systems need to define a metric of configurable parameters firstly. Secondly, as a very simple work model, an "agent" process may be used to handle outside configuration requests. When received administrative requests, the agent will interact with other components of the filesystem to complete the specified operations. Adjustability can be dealt with by the same way. Fig.1 gives the profile of the requests and responses.

In DCFS, configurable parameters or attributes are presented in Table1.

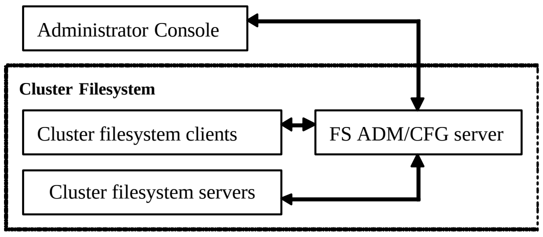

Fig.1 Basic Execution Model of Management Requests

2.4 Evolveable file service

In past 30 years, the Moore's law has driven the high-speed development of semi-conductor industry and information technology[3]. It is predicated that the Moore's law will continue to work in near future. Modern computing systems help mankind obtain unprecedented computing ability. A graceful system design should be able to make use of potential technologies that are not yet developed at present and here we named this "evolveability". For cluster file systems, we think evolveability means flexible architecture and object oriented design.

In cluster file systems, the components that are most heavily influenced by high-speed development of ASIC and magnetic technology are interconnection network, storage subsystem and main memory system in server /client. It would be better for cluster file systems to encapsulate the communication layer as much as possible to make it easily adapt to different network protocols. In order to make full use of main memory system, we can consider of use aggressive cache policies in the server and client side of DCFS. In addition, we should optimize the disk accesses to fully use the bandwidth of modern disk subsystem.

3 DCFS Implementation

DCFS (Dawning Cluster File System) is the cluster file system designed for Dawning4000 which is a Linux cluster built on commodity SMPs. Dawning4000 consists of large numbers of nodes and these nodes are interconnected through multiple networks, including Fast Ethernet, Gigabit Ethernet and Myrinet. Nodes are partitioned to use. Some nodes are used for user login, some for storage servers and some for computing. Compute nodes are usually DCFS client nodes.

3.1 Architecture

Fig.2 describes the framework of a DCFS. DCFS supports network disk striping organized in RAID-0 style. Besides, there're two types of servers in DCFS who are in charge of file data storage and file system meta-data (including super-block, directories, file attributes) storage respectively. In addition, the two kinds of server processes can co-exist on a node at the same time.

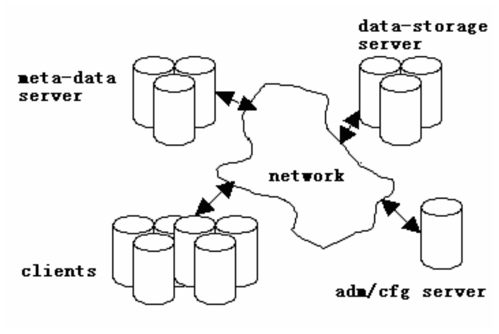

Fig 2. The framework of DCFS

The special adm/cfg server is designed to act as the agent for DCFS management. In section 3.4 we will continue introduce the management subsystem of DCFS.

In system implementation, we defined four different protocol sets in consideration of system components evolution and software maintenance. The basic file system protocol is abstracted and independent of other DCFS protocols. Communication protocol encapsulates all communication mechanisms used in system, including message passing between OS kernel and user process, client process and server process (UDP/TCP and BCL). To abstract high availability protocol from DCFS is fairly harder than other parts because of its strong dependence on basic file system protocol and we managed to do it. The DCFS management protocol is in charge of tasks related to administrator requests.

3.2 Status monitoring

In order to keep DCFS metadata consistency when any fault occurs in the system, we firstly designed a status monitoring and error reporting mechanism, which can detect such a failure. DCFS organizes all servers and clients into two rings: one for all DCFS client daemons, the other for all DCFS server daemons. Every daemon on a ring periodically sends heartbeat messages to its left and right neighbors. If some daemon hasn't got its neighbor's heartbeat for a specific time, then the daemon will send a message to a special daemon to inform the abnormity. This special daemon, which is called HA-master, is in charge of tasks about DCFS reliability. There is a backup of HA-master running on another node of the system. The backup daemon will take over the master when the HA-master is unreachable. Secondly, when the HA-master makes sure that the daemon is unavailable, it will inform all other daemons about the failure. The other daemons must perform properly to isolate the failed daemon from the system. Thirdly, should a DCFS client node fail, the DCFS need to check the metadata logs on the failed client and to do necessary recoveries to make the state of each DCFS server and client consistent. Should a DCFS server fail, another server will automatically take over the failed one, if there is another node in the system and it can access the disk of the failed node. This scheme is based on dual port disks, each of which is connected to two server nodes.

3.3 Multi-task capability in servers

In order to optimize the performance of DCFS, metadata and storage server processes are designed to be multi-threaded working model.

Each metadata server is composed of three threads: main controller thread, backend flushing thread and heartbeat handling thread. Because metadata size of filesystem is usually small, so when clients flush dirty metadata to a metadata server, the server may put it into metadata cache and flush it to disk later by a background flushing thread or forcibly flush it when out of space for new requests by main controller thread. Such a design may hurt the stability of data when the server crashes. We expect to enhance the security of cached metadata with hardware support such as NVRAM.

The working model of storage servers is far more complex than that of meta-data servers. There are four kinds of threads for each storage server, among which the heartbeat handling thread, the back end flushing thread and main controller thread are work like those of a meta-data server.

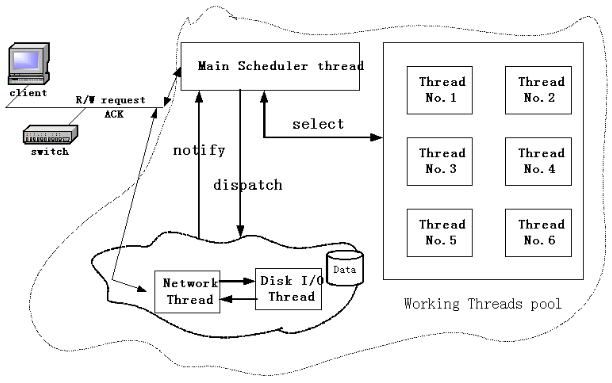

Fig.3 Execution model in data-storage server

Besides the three threads, the storage server has a group of working threads. The actual file read/write operations are performed by them. Here we introduce the concept of thread pair, which is composed of two threads, one is in charge of data transfers through network and the other is responsible for writing those data into or reading data from storage subsystem. As illustrated in Fig.3, when a file read/write request arrives, the main controller thread decides whether to dispatch a thread pair or a single thread to serve this request according to the requested data size. With the help of the thread pair, a single I/O request will be executed in storage server in a pipelined style to improve the throughput of a single I/O request.

3.4 Management

As mentioned in previous section, we designed a agent in cooperation with an administration tool to complete common management work, including starting and stopping DCFS service, creating new file system, adjusting the size of an existed file system, dynamically adding cluster file system server components and report the system status. The administration tool has both a GUI and a text user interface. Administrators can manage all DCFS file system through this tool.

In DCFS, the agent process, which is called ADM/CFG server process, holds all the configuration and state information of DCFS filesystems and is in charge of receiving management requests from the administration tool, passing them to relevant DCFS client/server daemons and waiting for all the answer.

4 Performance Results

We present some preliminary performance results of a DCFS prototype on a Linux cluster at the NCIC. The cluster was configured as follows at the time of our experiments. There were 64 nodes, each with two 2.2-GHz Pentium IV processors, 1-Gbytes RAM, a 80-Gbytes IDE disk, a 100 Mbits/sec fast-ethernet network card operating in full-duplex mode, and a 64-bit Myrinet card. The nodes were running Linux 2.4.18-10smp. The communication on Myrinet uses the fast communication library BCL[4], which was another software developed by NCIC for Dawning series clusters. The TCP/IP communication bandwidth provided by BCL on Myrinet is about 142MB/sec (by Netperf for 16KB message size). And the maximum read bandwidth of EXT2 over local IDE disk is about 40MB/sec, and the maximum write bandwidth is about 36MB/sec (read/write bandwidth is measured by iozone).

Fig.4 shows the aggregate read/write bandwidth of DCFS and PVFS with 8 storage server nodes. In these tests, each client node just ran one test process which read/wrote a 512MB file in DCFS (or PVFS). These processes operated on different files. As the Fig.4 shows, the aggregate read bandwidth of DCFS went down lower than that of PVFS when client number reached 4 or more. However, the aggregate write bandwidth of DCFS was always slightly higher than that of PVFS when the client number increased. The differences are caused by the different implementation between DCFS and PVFS. The storage server IOS in DCFS is multi-threaded while PVFS's IO server IOD is single-threaded. And we found that in LINUX the aggregate read bandwidth of multiple threads (or multiple processes) decreases dramatically when the number of threads increases. Table 2 shows the aggregate bandwidth vary with the number of threads that reads (or writes) to different files of local EXT2 simultaneously.

Table 2. Multiple Threads Read/Write Local EXT2 Files

Number of threads	Size/thread (MB)	Write (MB/s)	Read (MB/s)
1	4096	36.04	40.41
2	2048	32.12	20.94
3	1365	30.94	26.34
4	2048	31.05	13.76
8	2048	24.68	10.23
12	2048	22.84	9.87
16	2048	22.31	9.45

Fig.5 shows the aggregate throughput of DCFS and PVFS with 8 storage servers. The throughput is measured by the well-known benchmark PostMark. These tests were performed on 16 client nodes each of which ran a PostMark test process. Each process performs 500 transactions. The X-axis is the total number of files each of the test process operates. When the number of files is equal to or more than 800, files are put into 10 subdirectories. When the number of files is equal to or more than 6400, files are put into 100 subdirectories. As Fig.5 shows, the aggregated throughput of DCFS is higher than that of PVFS. This is just as we expect. Although PVFS and DCFS have the same number of storage servers, PVFS has only one metadata server. For DCFS, all those storage servers are also metadata servers. So metadata processing of DCFS outperforms that of PVFS. However, as Fig.5 shows, DCFS's aggregated throughput was going down sharply with the number of file increased. This is because metadata servers and storage servers of DCFS are based on EXT2 which performs poorly when the number of files in a directory is larger than 50000.

Fig.6 and Fig.7 show the performance enhancement when the number of storage servers increases in DCFS. In these tests, each client ran a process that read (or wrote) a 2GB file in DCFS. The processes read (or write) different files. Because very process operated a large file, now the performance bottleneck was not the interconnection network but the disks of

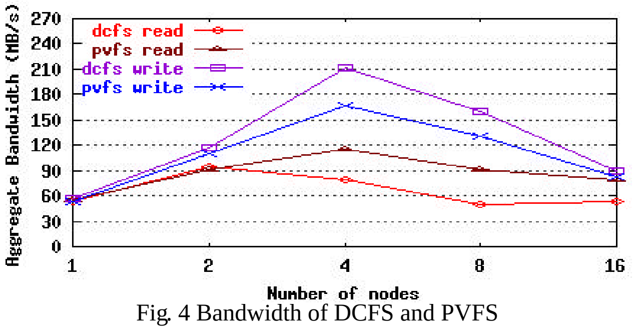

Fig. 4 Bandwidth of DCFS and PVFS

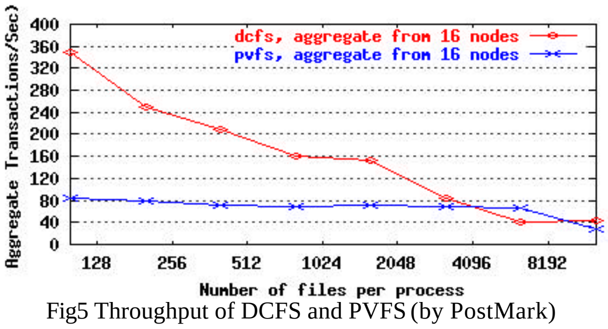

Fig5 Throughput of DCFS and PVFS (by PostMark)

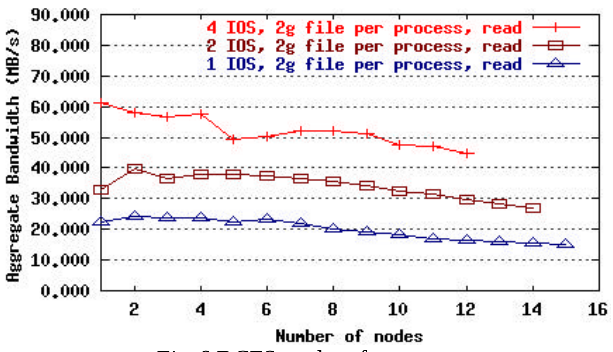

Fig. 6 DCFS read performance

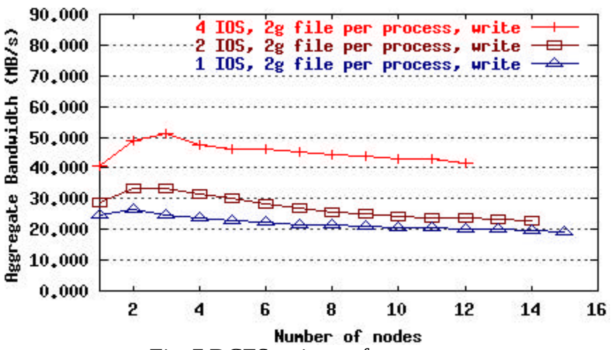

Fig. 7 DCFS write performance

the storage servers. As Fig.6 shows, the aggregate read bandwidth reached 24MB/sec with one storage server, 39.9MB/sec with 2 storage servers, and 61.1MB/sec with 4 storage servers. As Fig.7 shows, the aggregate write bandwidth reached 26.3MB/sec with one storage server, 33.4MB/sec with two storage servers, and 51.3MB/sec with 4 storage servers. The performance enhancement is not proportional to the number of servers added. This indicates that the protocol cost of DCFS increases when the number of servers increases.

5 Conclusions

In this paper, we described the design and implementation of DCFS, a cluster file system for the Dawning4000 Linux cluster system, and presented the preliminary performance test results of DCFS. Comparing with the previous cluster file system COSMOS[9] for Dawning2000 and Dawning3000, DCFS has similar architecture, however, DCFS exploits a different meta-data distribution policy and a more flexible striping policy, has no client cache, and integrates an administration tool. These measures simplify the implementation and management, and improve performance.

Comparing with PVFS, DCFS is easier to manage, and provides better throughputs. Its parallel I/O bandwidth is almost matchable with that of PVFS.

Our future work includes more detail performance analysis of DCFS and protocol optimization, implementation of client-side cache, improvement of the reliability in case of handling various component failures, and implementation of a user level file interface in order to better support of MPI-IO. We expect that client-side file data cache and metadata cache will substantially improves the overall performance of DCFS.

Acknowledgements

We would like to thank Tang Rongfeng, Fan Zhihua, Cao Liqiang, and Sun Guozhong. They helped us to do the performance tests.

References

[1] B. Callaghan, *NFS Illustrated*, Addison-Wesley, April 2000.
[2]D. Quintero, Z. Borgosz, A. Botura, D. Gilchrist, S. Kister, O. Lascu, K. So, *RS/6000 SP Cluster: The Path to Universal Clustering*, IBM Redbooks, SG24-5374-01.
[3] J. L. Hennssy and D. A. Patterson,. *Computer Architecture A Quantitative Approach*, Morghan Kaufmann Publishers, Inc., 1996, second edition.
[4] J. Ma, J. He, D. Meng and G. Li, " BCL-3: A High Performance Basic Communication Protocol for Commodity Superserver DAWNING-3000", *Journal of Computer Science & Technology*, Vol.16, No.6, November 2001, pp.522-530.
[5] J. Barkes, M R. Barrios, F. Cougard, P. G. Crumley, D. Marin, H. Reddy, and T. Thitayanun, " GPFS: A Parallel File System" , SG24-5165-00, April 1998.
[6] K. Hwang, Z. Xu, *Scalable Parallel Computing Technology, Architecture, Programming*, 1998 by McGraw-Hill, Inc.
[7] 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*, pp. 317—327, USENIX Association, Atlanta, GA, 2000.
[8] S. Shepler, C. Beame, B. Callaghan, M. Eisler, D. Noveck, D. Robinson and R. Thurlow, " NFS Version 4 Protocol" ,RFC 3010, November 2001.
[9] J. Wang, "A Scalable File System with Single System Image"**,** Ph.D thesis, Institute of Computing Technology, the Chinese Academy of Sciences, June 1999.
[10] C. A. Thekkath, T. Mann, E. K. Lee, "Frangipani: A Scalable Distributed File System", *Proceedings of the 16th ACM Symposium On Operating Systems Principles*, 1997, Saint Malo, France