Cluster Comput (2006) 9:433–447
DOI 10.1007/s10586-006-0011-6

Exploiting redundancy to boost performance in a RAID-10 style cluster-based file system

Yifeng Zhu · Hong Jiang · Xiao Qin · Dan Feng · David R. Swanson

Received: 16 August 2004 / Revised: 4 December 2004 / Accepted: 8 February 2005

Abstract While aggregating the throughput of existing disks on cluster nodes is a cost-effective approach to alleviate the I/O bottleneck in cluster computing, this approach suffers from potential performance degradations due to contentions for shared resources on the same node between storage data processing and user task computation. This paper proposes to judiciously utilize the storage redundancy in the form of mirroring existed in a RAID-10 style file system to alleviate this performance degradation. More specifically, a heuristic scheduling algorithm is developed, motivated from the observations of a simple cluster configuration, to spatially schedule write operations on the nodes with less load among each mirroring pair. The duplication of modified data to the mirroring nodes is performed asynchronously in the background. The read performance is improved by two techniques: doubling the degree of parallelism and hot-spot skipping. A synthetic benchmark is used to evaluate these algorithms in a real cluster environment and the proposed algorithms are shown to be very effective in performance enhancement.

Y. Zhu (✉)
Electrical and Computer Engineering, University of Maine, Orono, ME 04469, USA
e-mail: zhu@eece.maine.edu

H. Jiang · D. R. Swanson
Department of Computer Science and Engineering, University of Nebraska – Lincoln, NE 68588, USA
e-mail: jiang@cse.unl.edu

D. R. Swanson
e-mail: dswanson@cse.unl.edu

X. Qin
Department of Computer Science, New Mexico Institute of Mining and Technology, Mexico 87801, USA
e-mail: xqin@cs.nmt.edu

D. Feng
Department of Computer Science, Huazhong University of Science and Technology, Wuhan, China
e-mail: dfeng@hust.edu.cn

Keywords CEFT · PVFS · Cluster computing · Data storage · Cluster file systems · Redundancy · RAID

1 Introduction

A reliable and high-performance storage system is critical to I/O-intensive applications in clusters. Due to the steadily widening gap in speed between processors and disks, I/O operations have emerged to be the source of the most severe bottleneck for data intensive applications. Rapid performance advances in general-purpose communication networks used in clusters motivated the deployment of existing inexpensive commodity components to alleviate the I/O bottleneck [1–3]. Without compromising the cost-effectiveness of clusters, this approach utilizes the existing disks on all cluster nodes to build a parallel file system that not only provides a large-scale storage capacity (e.g., TBs in a cluster with one hundred nodes), but also taps into the aggregate bandwidth of these disks to deliver a high-performance and scalable storage service. Meanwhile, reliability is another important issue that must be addressed to make this approach more practical. Most clusters are error-prone due to the fact that the number of nodes involved is large and can reach tens of thousands. Thus using the existing disks on cluster nodes to provide cluster-wide shared storage service requires some form of data redundancy across nodes since all disks attached on failed nodes become inaccessible.

Previous research work on cluster-based storage systems mainly focused on integrating these distributed disks into a single disk volume and incorporating fault tolerance

[1, 4, 6, 7, 9, 10, 12, 17, 24, 25]. These systems do not consider the impact of the likely dual-role characteristics of cluster nodes, serving both as a compute node and also as a data server. While striping balances the I/O workload on all data servers, the disk, memory, network and CPU resources on these data servers can be heavily loaded by applications issued by cluster end-users and the overall workload on these cluster nodes can be highly imbalanced. As the slowest data server determines the performance of parallel I/O services, this workload imbalance can seriously hamper the aggregate throughput delivered out of these nodes.

This paper aims to minimize the performance degradation of parallel I/Os in the presence of workload imbalance among cluster nodes by exploiting the data redundancy to spatially and judiciously schedule I/O requests. Scheduling usually takes two steps in cluster-based storage. The first step, spatial scheduling, is responsible for assigning the I/O requests to data servers. The second step, temporal scheduling, consists of determining the execution order of various I/O requests arriving on a single data server to optimize the throughput of this server. Temporal scheduling is a classic problem that has been extensively studied in the past two decades and we will not delve into this problem in this paper. Assuming that a suitable or standard temporal scheduler is properly installed on each node, the spatial scheduling then becomes critical for a cluster-based storage system, since it aims to balance the workload of all cluster nodes and maximize the overall throughput of this cluster.

In our previous study, we designed and implemented a Cost-Effective, Fault-Tolerant parallel virtual file system (CEFT), which is a RAID-10 style system and combines striping with mirroring by first striping among a group of storage nodes and then duplicating all the data onto another group to meet both the performance and reliability requirements [12, 15]. This paper extends our previous studies presented in [13, 14, 16] and incorporates more experiments to evaluate our proposed approach. Based on the experimental results collected from a real cluster in production mode, this paper helps shed light on the following important design and performance issues: (1) What is the impact of resource contention on the aggregate storage throughput? (2) How to alleviate the negative impact of the load imbalance within each mirroring pair on the read and write performance? (3) How to exploit the data redundancy to improve the read performance?

The rest of this paper is organized as follows. The next section discusses the architectural assumptions of this paper. Then an overview of CEFT is presented in Section 3. Section 4 presents the cluster environment and the performance benchmark used in this paper. Sections 5 and 6 address the issues of write and read performance enhancements by exploiting the data redundancy and hot-spot skipping. The related work is discussed in Section 7. Finally, Section 8 concludes the paper.

2 Architectural assumptions

This paper considers generic clusters where a number of commercial, off-the-shelf personal computers are linked by a high-speed switched network with bandwidths ranging from 100 Mb/s to multiple Gb/s (Ethernet, Myrinet, Giganet, etc.). At least one storage device is attached locally on each node in the cluster. The cluster-based storage architecture considered in this paper assumes the following architectural characteristics.

- *Shared-nothing architecture*. All storage devices attached on a cluster node are only accessible through that node. This architecture is adapted in PVFS [1], xFS [6], Google File System [7], etc. It differs from shared-disk architecture adapted in Storage Area Network (SAN), such as GPFS [5], where each storage device allows direct and equal accesses by a group of nodes. The shared-disk architecture requires special hardware to support direct access protocols, thus compromising the cost-effectiveness of generic clusters.
- *Dual-role cluster nodes*. Each node in a cluster can perform dual roles, serving both as a compute node to run users' applications and as a data server to deliver I/O services. Accordingly, no cluster nodes are dedicated to a specific role and they are all available for end-users. This dual-role design not only provides the flexibility of cluster management, but also achieves better overall system utilization since computation tasks mainly consume CPU cycles while storage tasks mostly stress I/O resources.
- *Data and metadata decomposition*. Two models, i.e., *decomposition model* and *uniform model*, are widely deployed in distributed file systems to achieve high scalability by avoiding any single centralized component along the I/O data path. In the decomposition model, the functions of data and metadata managements are decomposed and all metadata is stored separately on different nodes away from the actual user data. While these nodes, called metadata servers, provide centralized metadata management, large volumes of actual user data are diverted to bypass these metadata servers. PVFS [1], Slice [8] and Google File System [7] use the decomposition model. In the uniform model, the metadata and user data are not separated but stored systematically on all nodes. All storage devices are virtualized into a single block address space and a file system is directly built upon this block space in a way similar to a conventional file system on a single disk device. Distributed locking is required in this model to synchronize concurrent accesses. Systems based on the uniform model include GPFS [5], Petal [9] and RAIDx [10]. This

paper adapts the decomposition model to simplify the design and enable the metadata servers to make sophisticated scheduling of I/O requests.

- *Switched or crossbar network connections*. All cluster nodes are linked through switched or crossbar connections, such as 10 Gigabit Ethernet and Myrinet, which provide aggregate bandwidth that scales with the number of machines on the network. If multiple nodes communicate with a single server simultaneously in clusters using such interconnects, the communication bottleneck is likely to shift from the network switch or crossbar to the local network interface card or the communication stack of the native operating systems on the server.

3 An overview of CEFT

CEFT extends PVFS [1] from a RAID-0 style parallel storage system to a RAID-10 style one that mirrors the striped data between two logical groups of storage nodes, one primary storage group and one backup storage group, as shown in Fig. 1. Files in CEFT are divided into fix-sized chunks and these chunks are placed within one group of data servers in a round robin fashion. On each data server, all chunks that are stripped on the same server and belong to the same file are stored as a regular file in the local file system of that data server. In each group, there is one metadata server that maintains two metadata structures, the system metadata and the file metadata. The system metadata includes the byte-ranged lease information that is similar to the data consistency mechanism in [11] and the configuration information that indicates the dead or live status of the data servers. When one data server is down, all I/O accesses addressed to the failed server will be redirected to its mirror server. Currently, a data server is simply thought to be down if the metadata server does not receive the periodic "heartbeat" message from this data server within a certain amount of time. The file metadata describes the mapping from files to storage chunks, the access control information, and the current mirroring status of each chunk. To access a file, a client needs to retrieve the desired metadata from the metadata servers and then directly communicates with the data servers. Thus, the bulk of file Content does not go through the metadata.

Another important task of the metadata servers is to spatially schedule I/O requests. All data servers monitor the utilizations of their own CPU, memory and network and piggyback this information on the periodic "heartbeat" messages to the metadata servers. For each I/O request, the metadata server makes a decision to choose one node from each mirroring pair by considering the workload disparities among all mirroring pairs or to skip a mirroring pair during striping if both nodes are heavily loaded.

For write accesses in CEFT, we have designed and implemented four novel mirroring protocols, each with distinctive operational and performance characteristics depending on whether the mirroring operations are server-driven or client-driven, and whether they are asynchronous or synchronous. In the server-driven protocols the data servers duplicate the new data to the mirroring groups, while the clients simultaneously write the data to both groups in the client-driven ones. The I/O completion is signaled only when the written data has taken residence on both groups in the synchronous protocols, while in the asynchronous ones residence of written data in the primary group alone signals such completion. These protocols strike different tradeoffs between the reliability and performance. Protocols with higher peak write performances are less reliable than those with lower peak write performances, and vice versa. However, only the asynchronous server-driven mirroring protocol can benefit from the I/O scheduling, as indicated in our study [7, 16]. Thus in the rest of this paper, all write operations are performed under the control of the asynchronous server-driven mirroring protocol.

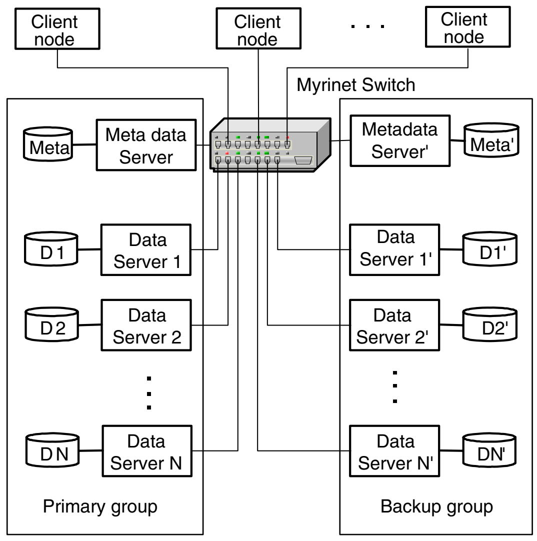

Fig. 1 Block diagram of CEFT

4 Experiment environments and evaluation benchmarks

All performance results presented in this paper are measured on the PrairieFire cluster [28] at the University of Nebraska-Lincoln. At the time of our experiments, the cluster had 128 compute nodes, each with two AMD Athlon MP 1600 processors, 1 GB of RAM, a 2 gigabits/s full-duplex Myrinet card, and a 20 GB IDE (ATA100) hard drive. The memory had a read and write throughput of 464 and 592 MB/s, respectively, measured by using the lmbench benchmark [26], and the PCI

```
For all clients:
        1. synchronize all clients using MPI barrier;
        2. t1 = current time;
        3. open a file;
        4. synchronize all clients using MPI barrier;
        5. loop to read or write data;
        6. close the file;
        7. t = t1 - t2; /* overall completion time */
        8. send t to client 0;

For client 0:
        1. find maximum of t; /* find the slowest client */
        2. calculate aggregate throughput using maximum t;
```

Fig. 2 Pseudocode of the benchmark

bus had a read and write throughput of 236 and 209 MB/s, respectively, measured by the gm_debug benchmark [27]. The Netperf [31] benchmark reported a TCP bandwidth of 126.51 MBs/s with 47% CPU utilization. To measure the disk performance, we used the Bonnie benchmark [32] to read and write a large file of size 2GB in order to significantly reduce the impact of caching. Our measurements showed that the disk read and write bandwidth were 26 MB/s and 32 MB/s, respectively.

In our experiments, our metadata servers are dedicated and there are no other applications running on these nodes. Since our targeted clusters typically have over one hundred nodes, using two dedicated metadata servers will not significantly compromise the system's cost-effectiveness.

A micro-benchmark, similar to the one used in [1, 18–23], was used to measure the aggregate read and write performance. In this benchmark, each client concurrently opens a common file, then reads or writes disjoint portions of this file, and finally closes it. The response time of the slowest client is considered as the overall response time. Figure 2 shows the pseudo-code of this benchmark. The performances were examined with two simple orthogonal approaches: (1) all the clients read or write the same amount of data but the total number of client nodes changes; (2) the total number of client nodes is fixed while the amount of data that each client read or write changes. All the performances reported in this paper were based on the average of 20 measurements.

The read and write operations are studied separately in this paper since they exhibit different characteristics in modern hierarchical storage architectures. Commodity PCs currently have a RAM with a capacity of multiple GB or even TB and hence almost all the data in write operations can be easily buffered in the RAM by the local file systems. Accordingly, the performance of write operations is largely influenced by the memory and network utilizations. On the other hand, the performance of read operations mainly depends on the data locality of applications and on the cache and prefetch functionalities of storage systems.

5 Write performance improvement

In the clusters considered in this paper, each cluster node played double roles: serving both as a compute node and as a storage server. All the users' applications running in these clusters have different requirements for system resources, primarily CPU, disk, memory and network, and accordingly, the utilizations of these system resources on different nodes can be significantly different. To improve the response times of write requests, usually half of the server nodes with relatively small workload are assigned to the primary group and writes are considered completed when the data has been stored in the primary group. The duplications from the primary group to the backup one proceed in the background in a pipelined fashion.

The challenge here is to determine what kind of node is considered less loaded. To address this issue, we will study the impact of different workload conditions of CPU, disk, memory, and network on the write performance in the following section. Then a scheduling algorithm that judiciously selects nodes with lighter workload in each mirroring pair to optimize write performance is proposed and evaluated.

5.1 Impact of system resources on write performance in a simple configuration

Since CEFT, a RAID-10 style system, strides the files among the data server nodes in a round-robin fashion and the write performance is largely determined by the slowest data server in one storage group, it is essential to understand the characteristics and behaviors of individual data servers under a variety of system resource utilizations, in order to be able to make load-balancing decisions dynamically. To make this problem tractable, we measure the performance of CEFT in its simplest configuration, in which either group contains only one data server and one metadata server, and in its simplest I/O access pattern, in which only one client writes a new file to the data server. While we artificially put different stresses on one of the resources of the data server and keep the other resources idle, we measure the write performance with increasing I/O load, i.e., increasing the file size.

5.1.1 Impact of CPU workload on write performance

While CPUs in general are not the bottleneck for I/O operations, they may be heavily loaded by scientific applications, especially computation-intensive programs, thus potentially increasing the I/O response time. The metrics of CPU workload are average CPU usage and load. The CPU usage is expressed as a percentage of total CPU time spent on active jobs since the last update and the CPU load, a parameter reported by Linux kernel, is defined as the exponentially-damped moving average of the sum of the number of processes

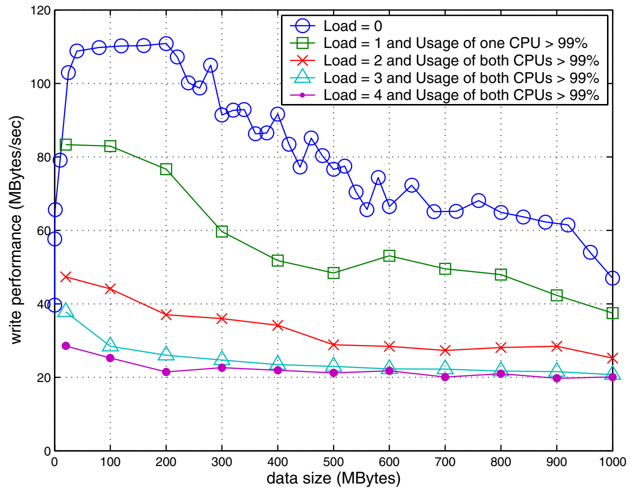

Fig. 3 Impact of CPU load on write performance when the client writes different amounts of data. There is only one data server and one client node in these experiments

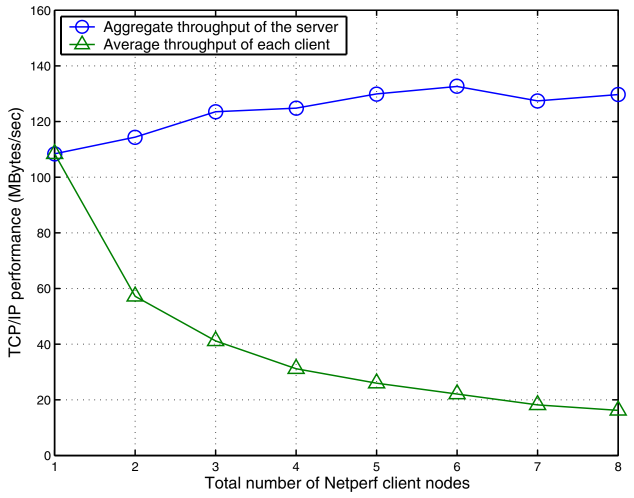

Fig. 4 TCP/IP performance when different Netperf clients concurrently communicate to one Netperf server

waiting in the run-queue and the number currently executing during the last minutes [30]. To artificially make the load of an idle CPU a specific number, such as three, we can fork three processes and let each process execute an infinite busy loop. We found that the impact of CPU load on the I/O performance was insignificant when the usages of both the CPUs on a data server node were below 99%. Figure 3 shows the write performance as a function of CPU load while both CPUs on the data server node are 99% utilized and the memory, disk and network are nearly 100% idle. The experiments indicate that the write performance can be reduced by approximately 31%, 60%, 70%, and 73% on average if the CPU is busy and the average load is 1, 2, 3, and 4, respectively.

5.1.2 Impact of network traffic load on write performance

CEFT uses the TCP/IP to transfer data between the client and server nodes. The TCP/IP performance over the 2 gigabits/s full-duplex Myrinet of PrairieFire was measured using Netperf [31]. Based on the basic client-server model, Netperf measured the performance by sending bulk data between the server and the clients. Figure 4 shows the TCP/IP performance as a function of different numbers of Netperf clients simultaneously communicating with one Netperf server. All the Netperf clients and server were located on different nodes in the cluster. We measured that the server had an average of 126.51 MB/s TCP/IP throughput, which was shared by all the clients. Our tests based on the gm_debug facilities [27] indicated that the PCI bus had a read and write throughput of 236 MB/sec and 209 MB/sec, respectively. While the average CPU usage of the Netperf server was only 47% during the measurement, the bottleneck of the TCP/IP performance was likely located at the TCP/IP stack on the server side, which required an integrated memory copy and thus generated an extra, potentially large latency.

Another important observation from Fig. 4 was that when more than five nodes concurrently communicated with the same node, the average throughput of an individual node was less than the maximum disk throughput, implying that when there are communication-intensive applications running on the server nodes, the bottleneck of I/O operations could potentially shift from disks to their TCP/IP stacks.

The write performance under different numbers of Netperf clients is shown in Fig. 5 where Netperf server and the CEFT data server were deliberately placed on the same node. When the size of I/O request was not large, the Netperf client nodes and the CEFT client nodes shared the TCP/IP bandwidth

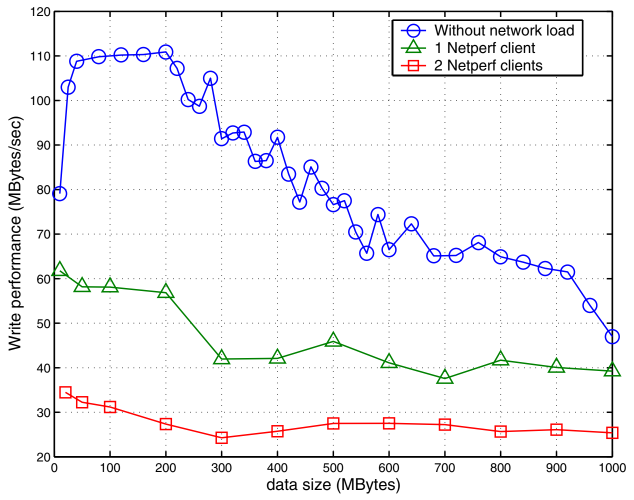

Fig. 5 Write performance under different network traffic disturbances. There is only one CEFT data server and one CEFT client node in these experiments.

```
1. M = allocate( 10 Mbytes );
2. create a file named F;
3. while ( 1 ) {
4. if ( size(F) > 5 GB )
5. truncate F to zero byte;
6. else
7. synchronously append data in M to F.
8.  }
```

Fig. 6 Program to stress the memory and disk on a CEFT data server

nearly evenly. With the increase in I/O requests, the performance further degraded due to the compounded negative impact of memory shortage.

5.1.3 Impact of memory and disk load on write performance

Memory and disk are closely coupled in almost all modern operating systems since most Linux systems employ the virtual memory technology and disks are part of the virtual memory. Data might be paged out from the memory into the disk when necessary. On the other hand, memory serves as buffer cache for disk drives. Reading and writing data on disks will influence the usage of memory. Thus we only analyze the overall impact of disk and memory in this paper.

A simple program is developed to stress the disk and memory on data server nodes. In this program, the synchronous write always guarantees a disk access, but the operating system usually places the most recently used data in the cache buffer in an effort to avoid some disk accesses. Although this caching buffer can be automatically reclaimed by the operating system, the competition for memory between this program and CEFT on the server node will certainly reduce the write performance. When only this program is stressing the disk and memory, both CPUs are nearly 95% idle and therefore CPUs have negligible impact on the write performance during this set of measurements. Another observation from our experiments is that, like the network characteristics shown in Fig. 3, the disk bandwidth is nearly equally shared by all the I/O-intensive processes running on the same node. For example, if there are five processes concurrently writing a large amount of data into the same disk, the I/O performance of each process would be around 8 MB/s when the maximum write throughput of the disk is 40 MB/s. This can be understood from the following aspects. (1) For large writes, the seek time is amortized by the large data transfer time. (2) Data locality in workloads reduce the seek time. It is discovered that the actual average seek time and rotational latency are, respectively, only about 35% and 60% of the specified values in a wide range of workloads [29]. (3) Since we are writing the data in the file system level, the number of disk seek operations is reduced by the cache and buffer management module in operating systems since I/O requests are aggregated, delayed and reordered in buffer so that many seek operations are saved. In this way, several small writes can be combined into a large write.

In the paper, we did not consider the memory bandwidth loading for the following reasons and difficulties. (1) There are no efficient facilities available in the Linux kernel or other application tools to monitor the memory throughput. (2) The memory traffic is very bursty due to the fact that the memory capacity is limited and the memory bandwidth is much faster than other I/O components, such as disks, PCI bus, TCP/IP stacks. (3) It is challenging to predict the memory activities. Since the memory operations are too bursty, it is difficulty to find a good time period to predict future available bandwidths. A short observation period might introduce a large CPU overhead. A long observation period makes the prediction inaccurate since the memory activities have evolved significantly. (4) Memory is not a performance bottleneck for many applications. For example, the memory bandwidth is seldom saturated by disk I/O and network I/O intensive applications. For memory write intensive applications, this saturation will only last for a short period of time since memory is used up soon and paging out data into disk slows down the memory operations.

As shown in Fig. 7, when the disk and memory are stressed by the program described above, the write performance in CEFT drops nearly 64% even when the data size is only a small fraction of the total available memory. Under this heavy disk and memory stress, write performance approximates the disk maximum throughput even when the file size is small enough to be buffered. When data size is large, the write performance drops to around half of the maximum disk throughput since the data cannot fit in the memory and the writes in CEFT have to compete for the disk bandwidth with the stressing program. We conclude that when the CPU load is not high, the disk-memory "compound" plays a more significant role than the network.

5.2 To skip or not to skip a busy node while striping?

When the system resources on one mirroring pair are heavily loaded, it might be beneficial to skip these nodes while striping, in order to balance the write load among the designated group of mirroring pairs. Can skipping the busy nodes compensate for the reduced parallelism? To answer this question, we need to exam how the performance scales with the total number of data server nodes when all the server nodes are lightly and equally loaded.

Figures 8 and 9 show the aggregate performances corresponding to two cases: constant-sized files being written by a variable number of client nodes, and variable-sized files being written by a constant number of client

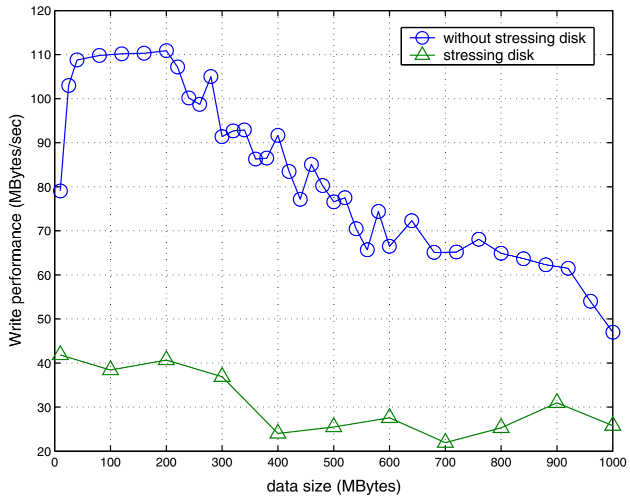

Fig. 7 Write performance when the memory and disk are stressed. There is only one CEFT data server and one CEFT client node in these experiments

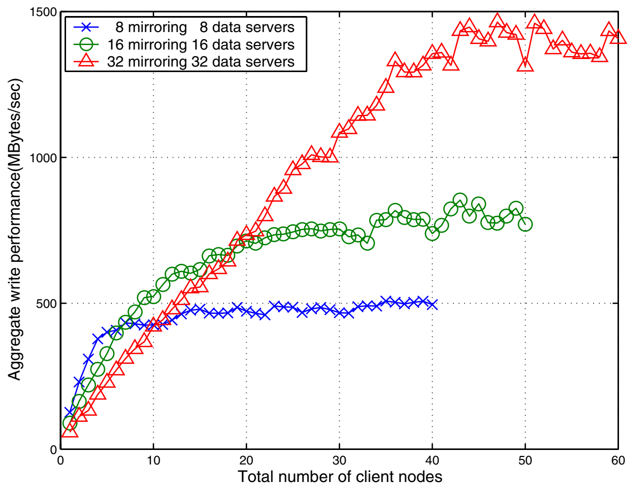

Fig. 8 Aggregate write performance of CEFT when each client writes 16 MB data to the servers. There are 8 data servers in each group.

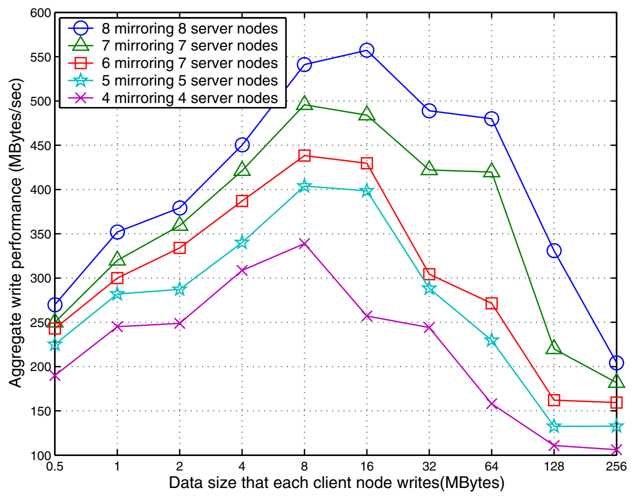

Fig. 9 Aggregate write performance of CEFT when the total number of client nodes is 16

```
Function:
choose a data server between node i and its mirrored node j

Inputs:
NETi, NETj: available network throughput on node i and j
MEMi, MEMj: size of available free memories on node i and j
DISKi, DISKj: available disk throughput on node i and j
CPUi1, CPUi2, CPUj1, CPUj2: average CPU usages on node i and j
LOADi, LOADj: average CPU load on node i and j
DISKmax: maximum disk throughput
FSIZE: size of the desired portion of the destination file

Algorithm:
1.   if min(CPUi1,CPUi2,CPUj1,CPUj2)   99% and min(LOADi,LOADj) > 2
2.       set the skipping flag;
3.   else if min(CPUi1, CPUi2) ≥ 99% and LOADi > 2
4.       choose node j;
5.   else if min(CPUj1, CPUj2) ≥ 99% and LOADj > 2
6.       choose node i;
7.   else if MEMi > FSIZE and MEMj > FSIZE
8.       if NETi ≤ 0.5*DISKmax and NETj ≤ 0.5*DISKmax
9.           set the skipping flag;
10.      else
11.          choose i if NETi ≥ NETj; otherwise, choose j;
12.      end
13.  else if MEMi ≤ FSIZE and MEMj ≤ FSIZE
14.      if min(DISKi, NETi) ≤ 0.5*DISKmax
15.          and min(DISKj, NETj) ≤ 0.5*DISKmax
16.          set the skipping flag;
17.      else
18.          choose i if min(DISKi, NETi) > min(DISKj, NETj);
19.          otherwise j;
20.      end
21.  else
22.      choose i if  MEMi > FSIZE > MEMj and NETi ≥ 0.5*DISKmax
23.      choose j if  MEMj > FSIZE > MEMi and NETj ≥ 0.5*DISKmax
24.      choo se i if MEMj > FSIZE > MEMi and NETi ≥ 0.5*DISKmax
25.      choose j if  MEMi > FSIZE > MEMj and NETj ≥ 0.5*DISKmax
26.          otherwise set the skipping flag;
27.  end
```

Fig. 10 Scheduling algorithms for write I/O operations

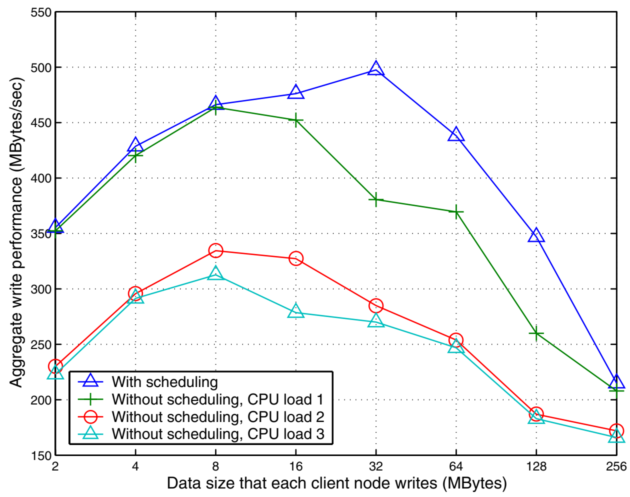

Fig. 11 Aggregate write performance when the CPUs on one data server is stressed

nodes, given that all the nodes are not heavily loaded. The average peak performances in the saturated region in Figure 8 of the three different CEFT configurations are 492, 796 and 1386 MB/s respectively, which are nearly proportional to the total number of data servers, thus indicating a good scalability of CEFT. This scalability, however, does not necessarily hold in the unsaturated regions in both Figs. 8 and 9, implying that a larger number of server nodes do not necessarily result in a proportionally higher write performance. In fact, the opposite is true when the file size falls in the range of 0.5 to 8 MB. In other words, for some file sizes a larger number of server nodes result in lower performance, and vise versa. It is this counter-intuitive property, shown in both figures, that necessitates skipping some data servers to improve the overall performance. In fact, such skipping is necessary even when all the server nodes are well balanced. However, judiciously skipping server nodes, or, equivalently, resizing the striping group, in a well-balanced system to improve write performance, while necessary, is beyond the scope of this paper, and thus will not be addressed.

In a realistic setting of cluster computing, the workload on all the data servers could be significantly different since parallel scientific applications usually are scheduled to run on only a portion of the nodes, instead of every node. It is possible, in fact, rather likely, that one mirroring pair are both heavily loaded, thus degrading the overall performance substantially. In such cases, skipping the busy pair helps alleviate the negative impact of the pair due in part to their dual roles in the cluster as a CEFT server node and as a compute node. Experiment results show that if a mirroring pair is heavily loaded and the maximum I/O throughput that they can provide is only about half of the disk bandwidth, skipping this mirroring pair usually improves the overall performance. This observation is helpful in developing the heuristic scheduling algorithm, to be described next.

5.3 A dynamic scheduling algorithm for write operations

Previous sections presented quantitatively the impact of resource availability of various kinds on the behaviors of writes in a simple configuration and under a simple workload pattern. In addition, experimental results suggest that judiciously skipping some server nodes while striping can be potentially beneficial to performance enhancement, especially for write-once applications. While such simplistic but quantitative results about performance impact of resource availability may not be directly extended to a CEFT with multiple data servers and more complex I/O workload, the relative sensitivities of resource availability of different kinds and the scalability information implied can give useful heuristic hints to the development of a dynamic scheduling algorithm for load balancing.

Since the metadata server is responsible for all the scheduling work, which can potentially form a bottleneck, we try to keep the scheduling algorithm as simple as possible to reduce the scheduling overhead. A straightforward algorithm is developed in this paper for write operations. In this algorithm, we only consider skipping at most one data server in a striping group to reduce the intrinsic scheduling complexity. Based on our experiences, skipping one node that can provide at most half of the maximum disk throughput significantly boosts the overall performance. Thus the value of one half of the maximum disk throughput is used as the threshold to decide on skipping.

The basic idea of this algorithm is that for each mirroring pair, if it is not heavily loaded, one node that could potentially deliver a higher I/O throughput from each mirroring pair is chosen to construct the primary storage group. In addition, according to the skipping criteria, all these pairs are sorted into four groups, each in non-increasing order of the utilizations of CPU, memory, disk, and network, respectively. If none of the utilizations of a particular resource, say memory, of the pairs is over 50%, then the sorted group based on memory utilizations will be empty. While each group is assigned a different priority and the priorities from the highest to the lowest are memory, network, disk and CPU, a pair in the non-empty group with the highest priority will be randomly chosen to be skipped.

In this dynamic scheduling algorithm, the available disk throughput D_i on node i is estimated as $\min(D_{\max} - D_{\text{used}}, D_{\max}/(n+1))$, where $D_{\max}$, D_{used} and n are the maximum disk throughput, the disk throughput of the last interval, and the total number of processes that are carrying out I/O operations, respectively. The available network throughput is estimated in a similar way. The size of the free memory space on a node is obtained from the memory management system of the operating system kernel. All these parameters are stored on the metadata server. The data server nodes collect this information and send it to the metadata server every one-second.

5.4 Write performance evaluation

In this section, we evaluate our dynamic heuristic scheduling algorithm in a configuration of eight data servers in each striping group. To fairly compare the performance with scheduling and without scheduling, the benchmark programs need to be executed in the same environment with identical workload. In a real cluster in production mode, such as the PrairieFire in which CEFT is installed, unfortunately, it is nearly impossible to obtain such a repeatable environment since the workload on each node is constantly changing with the progression of applications running in the cluster. Therefore, instead of doing comparisons in a real environment, we compared performances in an artificially created environment in which the load of a specific kind of resource on a server node was kept approximately constant by using the programs described in the previous sections, although interferences from other computation programs running on the cluster could not be avoided.

To make sure that the bottleneck of the I/O operation was located on the server side rather than the client side, 16 client nodes were used to simultaneously write to the server and the aggregate performance was measured. Two sets of experiments were conducted. In the first set, the workload stress was applied only on one node while its mirroring node is kept almost idle so that skipping will not be necessary. In the second set, the workload stress was put on both nodes of a mirroring pair so that it will become necessary to skip. In each set of experiments, the CPU, network, and the disk-memory compound were each stressed in turn, and the results are presented in the following figures. In each figure, the average write performance of the scheduling algorithm is shown, since under different stress conditions of the same resource, the performances of the scheduling algorithm were very close.

Figures 11 and 12 show results of experiments in which the CPU and network of one primary node were stressed, respectively. In experiments reported in Fig. 13, both the disk and memory were stressed on one node or on two nodes in the same striping group. In Figs. 14 and 15, the CPU and network of one mirroring pair were stressed respectively. Figure 16 presents the comparison when both the disk and memory on one mirroring pair were stressed. The performance of the dynamic scheduling is significantly better than the performance of non-scheduling in the vast majority of the test cases.

In the cases of skipping, shown in Figs. 14 and 16, the aggregate performance of the scheduling algorithm starts to decrease sharply when the data size of each client is larger than 64 MB. This sharp decrease is due to the fact that, as data size from each client node increases, the total file size allocated on each individual server node becomes so significantly larger that the negative impact of load redistribution (as a result of skipping) onto the remaining 7 server nodes quickly offsets the positive gain from skipping. These figures show that when one of the resources on a server node is heavily loaded, our scheduling algorithm derived from the heuristic observations, can significantly improve the write performance.

Figure 17 shows the comparison of our scheduling algorithm with two other algorithms, one solely based on the availability of disk and memory, and the other solely based on the availability of network bandwidth. This figure clearly shows that two simplistic algorithms are inferior to ours since both of them are limited by the amount of information on which their decisions are based while our algorithm bases its decision on a more comprehensive piece of system workload information. The performance in Fig. 17 is a little higher than the performance in the other figures because Fig. 17 is measured immediately after the reboot of our cluster and there is almost no computation application except our load stress program.

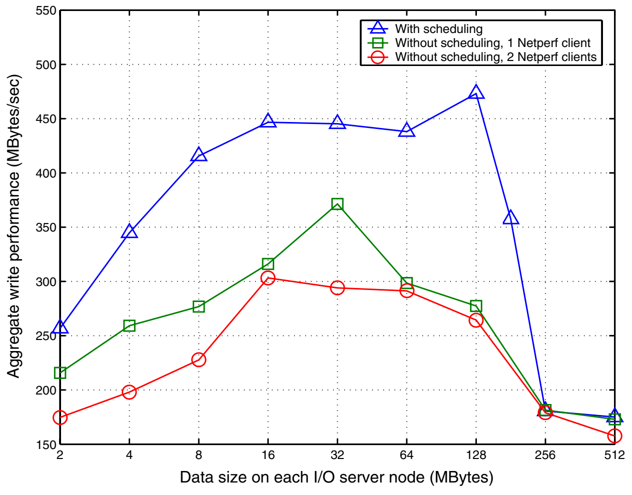

Fig. 12 Aggregate write performance when the network is stressed

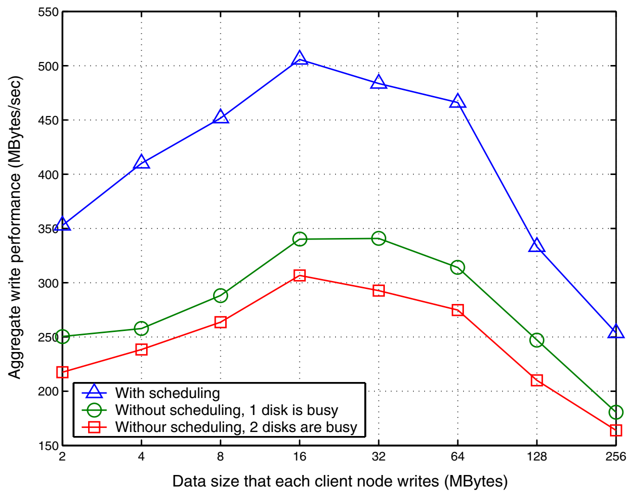

Fig. 13 Aggregate write performance when the disk and memory on one data server is stressed

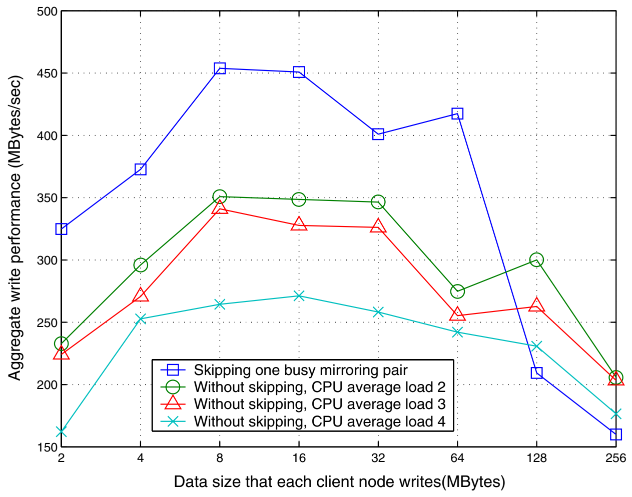

Fig. 14 Aggregate write performance when the CPUs on one mirroring pair of data servers are stressed

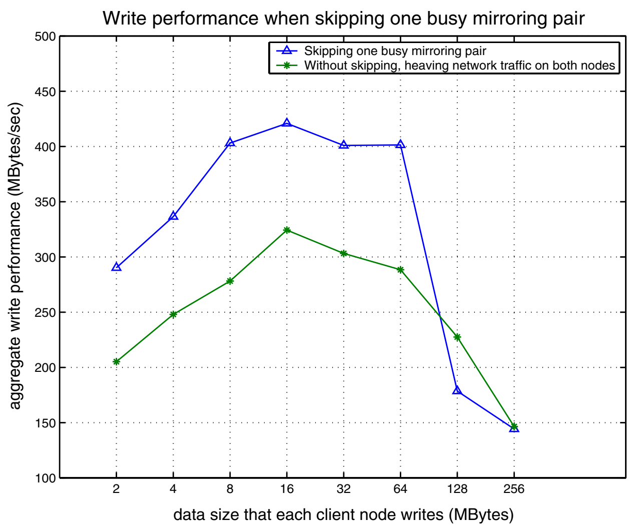

Fig. 15 Aggregate write performance when the network of one mirroring pair of data servers is stressed

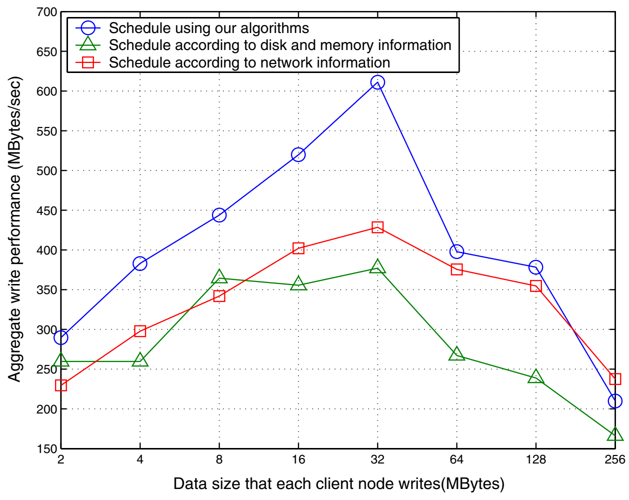

Fig. 17 Aggregate write performance when the disk and memory of one data server is stressed and the network of this mirroring server node is stressed

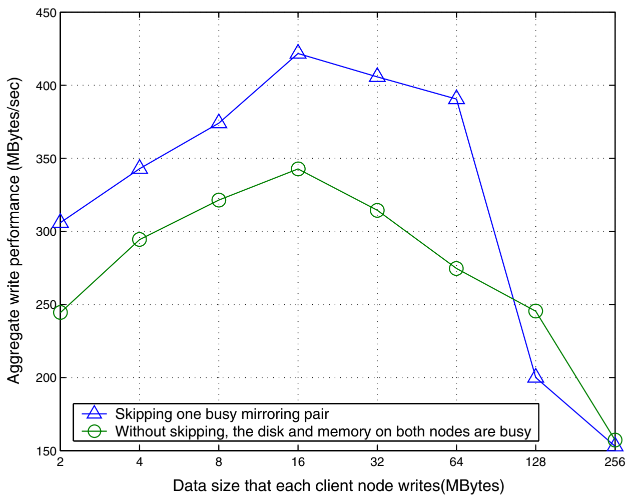

Fig. 16 Aggregate write performance when the disk and memory on one mirroring pair of data servers are stressed

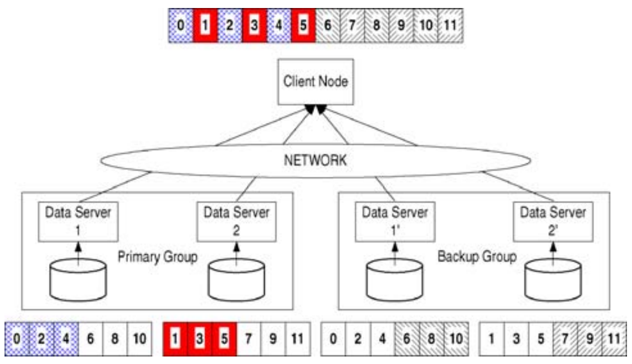

Fig. 18 An example of reading interleaved data from both groups, half from the primary group, and half from the backup group

6 Improving large read performance

The read operations exhibit different behaviors from write operations in modern clusters with RAMs of very large capacity (≥GBs/node). While the write performance is sensitive to the size of available buffer, the read performance highly depends on the data temporal locality of user's applications. In the following sections, we examine two extreme cases: *hot read* and *cold read*. In the case of hot read, all data is most likely to be cached by the memory on the servers and thus the number of disk accesses is kept minimal. The hot read performance is measured by reading the same data repeatedly. In cold read, all data has to be read from the disks. To clear the cache buffer and guarantee that real disk accesses take place, each data server reads a dummy file of 2 GB, twice as much as the total memory size of a data server node on the CEFT-installed cluster (the PrairieFire cluster) at the time of the test, before each measurement, thus displacing any cached data. All the read performances reported below were obtained in a configuration of 18 server nodes, including 8 data servers and 1 metadata server in each group.

6.1 Increasing parallelism of read operations

Any data stored in CEFT will eventually have two copies, one in the primary group and the other in the backup group. The storage space overhead for mirroring can be viewed as trading not only for the significantly increased reliability, but also for the increased read parallelism. Instead of reading the whole data from one storage group, the reading operations can divide their load between the two storage groups. More specifically, the desired data is split into two halves and the client can simultaneously read interleaved blocks, one half from the primary nodes and the other half from their mirroring nodes. Splitting the read loads on both groups is especially effective for large read operations, which are common

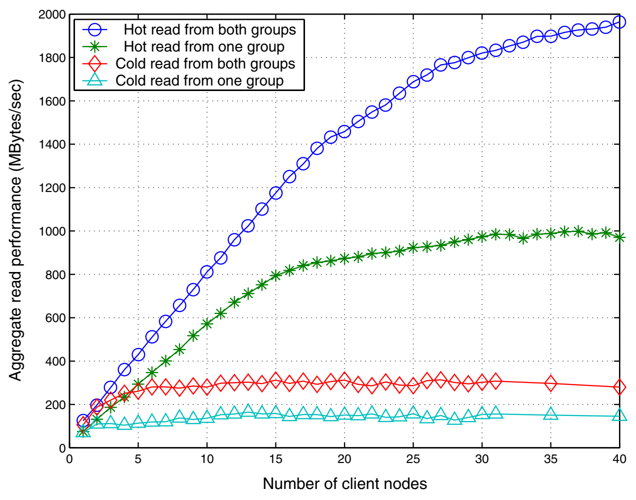

Fig. 19 Aggregate cold read and hot read performance, as a function of the number of client nodes. There are 8 data servers in one group and each client reads 16 MB

in scientific computations. In cases of small reads, this approach may not benefit much since the I/O completion time of small reads is dominated by the network and disk latency, instead of the data transferring time. Figure 18 shows an example in which each storage group is composed of two server nodes and the client node reads the target data from the four servers concurrently.

6.2 Read performance evaluations

Figure 19 shows the performance of the first approach when all servers are lightly loaded by the other applications and each client reads 16 MB data from the servers simultaneously. Table 1 summarizes the aggregate peak read performance and peak performance per server node. As the table indicates, the aggregate performance of hot read reaches its maximum value when all the network bandwidths from the data servers are fully utilized. The performance of cold read enters its saturation region when the throughput of each disk is close to their maximum value of 26 MB/s. These measurements show that the increased parallelism due to mirroring improves the performance nearly 100% for both the hot read and the cold read.

Figure 20 plots the performances measured by the second approach, when there are a total of 16 clients and each of them reads different sizes of data from the servers. The aggregate peak performance and peak performance per server node are summarized in Table 2. In the case of cold read, the performance begins to drop after an initial rise while this drop is not apparent in the hot read. The performance drop is potentially due to the fact that when the file size is too large, these files may not be stored contiguously on the disks so that more disk seeks are performed, causing the total disk access time to increase. In hot read, the peak performances are a little lower than the values given in Table 1 since the number of clients is not large enough to saturate the network bandwidth on the server side. The aggregate peak performance of hot read from both groups can be increased if more clients are added since the network bandwidth utilization on the server side is only 66%. Within the range of data sizes tested, our proposed method improves the cold read performance 76–100%, with an average of 91%, and boosts the hot read performance 22–59%, with an average of 49%, even when our proposed method has not achieved its maximum throughput in these measurements due to an insufficient number of clients.

Table 1 Aggregate peak read performance and peak read performance per server node when each client reads 16 MB

	Peak read performance (MB/s)	
	Aggregate	Per server node
Hot read from both groups	1964.4	122.8
Hot read from one group	998.5	124.8
Cold read from both groups	313.7	19.6
Cold read from one group	164.3	20.5

Table 2 Aggregate peak read performance and peak read performance per server node when 16 clients read different sizes of data

	Peak read performance (MB/s)	
	Aggregate	Per server node
Hot read from both groups	1326.3	82.9
Hot read from one group	897.9	112.2
Cold read from both groups	316.8	19.8
Cold read from one group	160.8	20.1

6.3 Improving read performance in the presence of hot-spot nodes

As an integral part of a cluster, all the data server nodes are not dedicated and they also serve as compute nodes. Their workload can be highly imbalanced, thus potentially degrading the overall I/O performance. Since all data is eventually stored on two different nodes in CEFT, this redundancy in CEFT provides an opportunity for the clients to skip the hot-spot node that is heavily loaded (or down due to failure) and read the target data from its mirroring node. More specifically, the server nodes periodically send their load information, including the load of CPU, the average throughput of disks and networks within each period, to the metadata server. The metadata server schedules the I/O requests and informs the clients of their reading schemes. Figure 21 shows an example, in which Node 2 is skipped and all data is read from its mirror Node 2'.

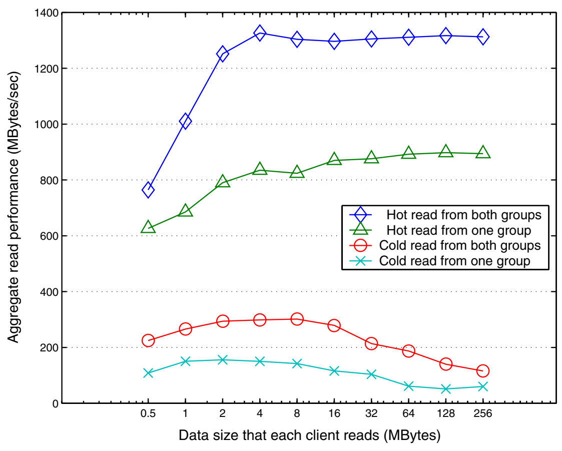

Fig. 20 Performance of cold read and hot read as a function of data size that each client reads. There are 8 data servers in one group and 16 clients

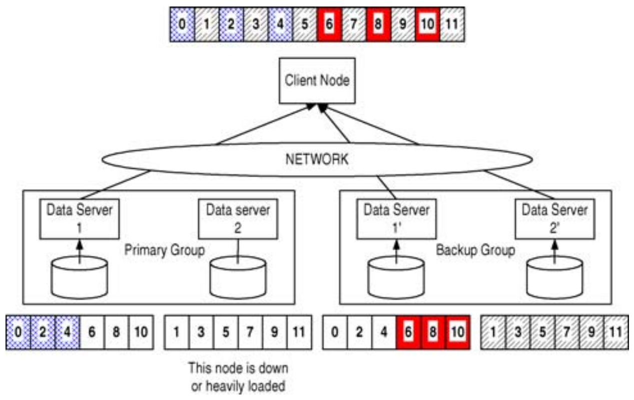

Fig. 21 An example of skipping the heavily loaded data server nodes and reading the data from their mirroring server nodes

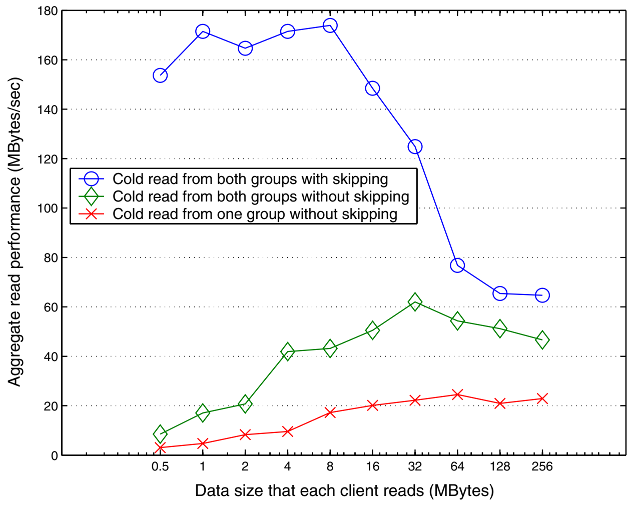

Fig. 22 Cold read performance improvement by skipping one server with heavy disk load and reading the data from its mirror. There are 16 clients and the data size that each client reads changes

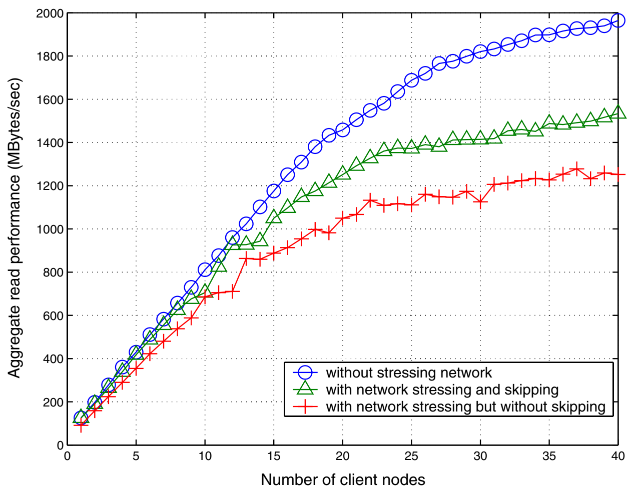

Fig. 23 Hot read performance improvement by skipping the server with heavy network load and reading the data from its mirror. Each client reads a total of 16 MB from both groups simultaneously

6.4 Improving cold read performance

In cold read, the data needs to be read from disks, which generate the largest latency on the critical path of I/O operations, due to the relatively large seek time and small bandwidth of disks. To compare the performance of skipping the hot-spot nodes, we artificially stress the disk on one server node in the primary group by allocating a memory space with 10 MB garbage data and then repeatedly storing these garbage data synchronously onto the disk. Three different methods are used to measure the read performance: (1) from all servers in the primary group without skipping the busy node; (2) from all servers in both groups without skipping the busy node; (3) from both groups while skipping the busy node. Figure 22 shows the performance curves of those methods measured under the same file access pattern, where 16 client nodes read different sizes of data from these servers. When the file size is small, skipping the busy node improves the cold read performance nearly 10 times over reading the data from one group or both groups without skipping. As the data size increases, the benefits from skipping decrease since the total data size from the mirroring node of the skipped node increases at a doubled speed, causing the total disk seek time to increase.

6.5 Improving hot read performance

Contrary to cold read, hot read can most likely find the data in the cache due to the aggressive design of the Linux operating system, which tends to use all the free memory as the cache buffer for the sake of minimizing disk accesses. This local optimization exploits the data locality exhibited in most applications to alleviate the I/O bottleneck. CEFT servers utilize their local file systems to store or retrieve all data and cache the most recently visited data in their memory. As discussed in Section 4, the memory has a read and write throughput of 464 and 592 MB/s, and the PCI bus has a read and write throughput of 236 and 209 MB/s and the TCP bandwidth is only 126.5 MB/s. Thus the network on the server side becomes the bottleneck in the case of hot read.

Figure 23 plots the hot read performance from both groups, under three approaches: (1) without stressing the network; (2) with the network interface of one data server stressed but without skipping this server; (3) with the network interface of one data server stressed and skipping this data server. In all measurements, each client reads a total of 16 MB data. When the total number of client nodes is small, the hot read performance does not show much difference among the three approaches since the bottleneck is on the clients' network interfaces. As the client number increases, the bottleneck gradually shifts from the clients' network interfaces to the servers' network interfaces. Stressing the network of one server node reduces the peak hot read performance from 2 GB/s to 1.25 GB/s. By skipping that network stressed node, the hot read performance is improved to 1.53GB/s, with an enhancement of 22.4%.

7 Related work

Our work is primarily related to I/O scheduling in clusters. Previous research work on scheduling I/O operations in a cluster environment can be classified into two categories, namely, spatial scheduling and temporal scheduling. The temporal scheduling algorithms essentially determine the execution order of the requests at each cluster nodes. For example, Ref. [33] proposes a bipartite graph edge-coloring algorithm to partition all pending I/O requests into subsets such that the requests in each subset do not compete for resource with each other and thus can be executed simultaneously. Ref. [34] designs an approximate algorithm of edge coloring to schedule I/O transfers for systems that only allow a limited number of transfers at a time. The efficiencies of those scheduling algorithms are evaluated based on simulation or theoretical analysis and are not examined in a real dynamic cluster environment.

The spatial scheduling algorithms basically make decisions about which nodes a request should be assigned to. For example, Ref. [35] uses the graph theory of the network maximum flow to evenly schedule I/O requests on all replicas, but it only evaluates their algorithm via theoretical analysis. To improve the scalability of cluster-based storage, Ref. [36] decomposes the storage nodes into three functional groups that serve metadata I/O, small file I/O and bulk data I/O respectively and develops a request routing scheme that schedule I/O requests to their corresponding storage nodes. This scheduling is based on static function of decomposition and does not incorporate data redundancy into the scheduling. Both Ref. [35] and [36] assume that all servers are dedicated and thus not oriented to generic clusters. Ref. [37] presents two heuristic algorithms to dynamically assign data servers in a heterogeneous cluster with both slow and fast disks. While Ref. [37] studies the placement of data servers that are not dedicated, it puts more focus on handling heterogeneity, instead of dealing with resource contention. Ref. [35] also considers the non-dedication characteristics of a generic cluster node and proposes a weighted bipartite matching algorithm with a goal to balance the workload of data servers. However their algorithm is based on the assumption that the data transfer time between different nodes is known before scheduling. In practice, it is challenging to satisfy this assumption due to the unpredictability of the workload on each cluster node.

This paper delves into spatial scheduling in a RAID-10 style parallel I/O system. Our research work distinguishes itself from the above in that we target the dual-role cluster nodes in a generic cluster, fully consider the memory, network, disk and CPU utilizations and incorporate the redundancy based on heuristics motivated from extensive experiments in a real cluster environment.

8 Conclusions

This paper investigates the I/O performance improvement in a generic cluster where each data server is not dedicated but time-shared with compute tasks. Thus nodes in such a cluster usually serve as compute nodes and as data servers simultaneously to preserve the cost-effectiveness of clusters. This paper studies the performance optimizations of a RAID-10 style file system running in such generic clusters. A new heuristic scheduling algorithm is proposed to schedule write operations on the nodes judiciously chosen from all mirroring pairs by considering the workload disparity between the nodes in a mirroring pair. If the nodes in a mirroring pair have already been heavily loaded, skipping this pair during striping is used to avoid these hot spots. The read performance is boosted by scheduling requests on both mirroring groups in order to double the degree of parallelism. In the case that a node becomes a hot spot, this node is skipped and all the data is read from its mirror node. Extensive experiments in a real cluster show that these performance optimization techniques significantly Improve the overall I/O performance in a generic cluster when the system workload is imbalanced.

While we designed and implemented the prototype of the dynamic scheduling algorithms, many important challenges remain. Our future work is to provide a more generic and platform-independent algorithm. We also plan to use more realistic Benchmarks to measure the I/O performance.

Acknowledgments This work is supported by NSF-HEC0621493, NSF-HEC0621526, an UMaine startup fund, an NSF Grant (EPS-0091900), a Nebraska University Foundation Grant (26-0511-0019) and a Chinese NSF 973 Project Grant (2004cb318201). Work was completed using the Research Computing Facility at UNL. Special thanks to the Argonne National Labs and Clemson University for making the source code of PVFS publicly available.

References

1. 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 (October 2000) pp. 317–327.
2. Y. Saito, S. Frolund, A.Veitch, A. Merchant, and S. Spence, FAB: building distributed enterprise disk arrays from commodity components, in: *Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2004)* Boston, MA (October 2004).
3. D. Anderson, J. Chase, and A. Vahdat, Interposed request routing for scalable network storage, in: *Fourth Symposium on Operating System Design and Implementation (OSDI2000)* 2000.
4. P.J. Braam, Lustre white paper, http://www.lustre.org/docs/whitepaper.pdf (Dec. 2003).
5. F. Schmuck and R. Haskin, GPFS: a shared-disk file system for large computing clusters, in: *Proceedings of the First Conference on File and Storage Technologies (FAST)*, Monterey, CA (January 2002).
6. T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang, Serverless network file systems, in: *Proceedings of the 15th Symposium on Operating System Principles (SOSP)*, Colorado, USA (December 1995) pp. 109–126.
7. S. Ghemawat, H. Gobioff, and S. T. Leung, The Google file system, in: *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)* (2003) pp. 29–43.
8. D. C. Anderson, J. S. Chase, and A. M. Vahdat, Interposed request routing for scalable network storage, ACM Transactions on Computer Systems 20(1) (February 2002) pp. 25–48.
9. E. K. Lee and C. A. Thekkath, Petal: distributed virtual disks, in: *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (1996) pp. 84–92.
10. K. Hwang, H. Jin, and R. S. Ho, Orthogonal striping and mirroring in distributed raid for I/O-centric cluster computing, IEEE Transaction on Parallel Distributed System 13(1) (2002) 26–44.
11. A. Adya, W. J. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R.P. Wattenhofer, FARSITE: federated, available, and reliable storage for an incompletely trusted environment, in: *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)* (December 2002).
12. 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 (September 2003).
13. Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson, Improved read performance in a cost-effective, fault-tolerant parallel virtual file system, in: *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID), Parallel I/O in Cluster Computing and Computational Grids Workshop* (May 2003) pp. 730–735.
14. Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson, Scheduling for improved write performance in a cost-effective, fault-tolerant parallel virtual file system, in: *Proceedings of the* 4th *LCI International Conference on Linux Clusters: the HPC Revolution 2003*, San Jose, California (June 2003).
15. Y. Zhu, H. Jiang, and J. Wang, Hierarchical Bloom Filter Arrays (HBA): a novel, scalable metadata management system for large cluster-based storage, in: *Proceedings of 6th IEEE International Conference on Cluster Computing*, San Diego, California (2004).
16. Y. Zhu, H. Jiang, X. Qin, and D. Swanson, A case study of parallel I/O for biological sequence analysis on Linux clusters, in: *Proceedings of 5th IEEE International Conference on Cluster Computing*, Hong Kong (December 2003) pp. 308–315.
17. H. Tang and T. Yang, An efficient data location protocol for self-organizing storage clusters, in: *Proceedings of the International Conference for Supercomputing (SC)* (2003).
18. H. Taki and G. Utard, MPI-IO on a parallel file system for cluster of workstations, in: *Proceedings of the IEEE Computer Society International Workshop on Cluster Computing*, Melbourne, Australia (1999) pp. 150–157.
19. R. Cristaldi, G. Iannello, and F. Delfino, The cluster file system: integration of high performance communication and I/O in clusters, in: *Proceeding of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, Berlin, Germany (May 2002).
20. 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.
21. F. Isaila and W.F. Tichy, Clusterfile: a flexible physical layout parallel file system, in: *Proceedings of IEEE International Conference on Cluster Computing* (2001) pp. 37–44.
22. S. Garg and J. Mache, Performance evaluation of parallel file systems for PC clusters and ASCI red, in: *Proceedings of IEEE International Conference on Cluster Computing* (2001) pp. 172–177.
23. M. Vilayannur, M. Kandemir, and A. Sivasubramaniam, Kernel-level caching for optimizing I/O by exploiting inter-application data sharing, in: *Proceedings of IEEE International Conference on Cluster Computing* (2002) pp. 425–432.
24. 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.
25. E.K. Lee and C.A. Thekkath, Petal: distributed virtual disks, in: *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (1996) pp. 84–92.
26. L. McVoy and C. Staelin, lmbench: portable tools for performance analysis, in: *USENIX Technical Conference*, San Diego, CA (January 1996) pp. 279–284.
27. gm_debug, http://www.myrinet.com, (2002).
28. Prairiefire Cluster at University of Nebraska - Lincoln, http://rcf.unl.edu (2003).
29. W. Hsu and A.J. Smith, The performance impact of I/O optimizations and disk improvements, IBM Journal of Research and Development 48(2) (2004) 255–289.
30. D.P. Bovet and M. Cesati, Understanding the Linux kernel, *O'Reilly & Assoc. Inc.*, Sebastopol, California (2001).
31. Netperf benchmark, http://www.netperf.org (2003).
32. Bonnie benchmark, http://www.textuality.com (2003).
33. D. Durand, R. Jain, and D. Tsytlin, Parallel I/O scheduling using randomized, distributed edge coloring algorithm, Journal of Parallel and Distributed Computing 53 (2003) 611–618.
34. R. Jain, K. Somalwar, J. Werth, and J.C. Browne, Heuristics for scheduling I/O operations, IEEE Transactions on Parallel and Distributed Systems 8(3) (March 1997) pp. 310–320.

35. P. Liu, D. Wang, and J. Wu, Efficient parallel I/O scheduling in the presence of data duplication, in: *Proceedings of the International Conference on Parallel Proceeding (ICPP)* (2003).
36. D.C. Anderson, J.S. Chase, and A.M. Vahdat, Interposed request routing for scalable network storage, ACM Transactions on Computer Systems 20(1) (2002) 25–48.
37. Y.E. Cho, M. Winslett, S. Kuo, J. Lee, and Y. Chen, Parallel I/O for scientific applications on heterogeneous clusters: a resource-utilization approach, in: *Proceedings of the 13th International Conference on Supercomputing (SC)*, Rhodes, Greece (1999) pp. 253–259.
38. J. Wu, D. Wang, and Y. Lin, Placement of I/O servers to improve parallel I/O performance on switch-based clusters, in: *Proceedings of the 17th annual International Conference on Supercomputing (SC)*, San Francisco (2003) pp. 244–251.

Yifeng Zhu received his B.Sc. degree in Electrical Engineering in 1998 from Huazhong University of Science and Technology, Wuhan, China; the M.S. and Ph.D. degree in Computer Science from University of Nebraska – Lincoln in 2002 and 2005 respectively. He is an assistant professor in the Electrical and Computer Engineering department at University of Maine. His main research interests are cluster computing, grid computing, computer architecture and systems, and parallel I/O storage systems. Dr. Zhu is a Member of ACM, IEEE, the IEEE Computer Society, and the Francis Crowe Society.

Hong Jiang received the B.Sc. degree in Computer Engineering in 1982 from Huazhong University of Science and Technology, Wuhan, China; the M.A.Sc. degree in Computer Engineering in 1987 from the University of Toronto, Toronto, Canada; and the PhD degree in Computer Science in 1991 from the Texas A&M University, College Station, Texas, USA. Since August 1991 he has been at the University of Nebraska-Lincoln, Lincoln, Nebraska, USA, where he is Professor and Vice Chair in the Department of Computer Science and Engineering. His present research interests are computer architecture, parallel/distributed computing, cluster and Grid computing, computer storage systems and parallel I/O, performance evaluation, real-time systems, middleware, and distributed systems for distance education. He has over 100 publications in major journals and international Conferences in these areas and his research has been supported by NSF, DOD and the State of Nebraska. Dr. Jiang is a Member of ACM, the IEEE Computer Society, and the ACM SIGARCH.

Xiao Qin received the BS and MS degrees in computer science from Huazhong University of Science and Technology in 1992 and 1999, respectively. He received the PhD degree in computer science from the University of Nebraska-Lincoln in 2004. Currently, he is an assistant professor in the department of computer science at the New Mexico Institute of Mining and Technology. He had served as a subject area editor of *IEEE Distributed System Online* (2000–2001). His research interests are in parallel and distributed systems, storage systems, real-time computing, performance evaluation, and fault-tolerance. He is a member of the IEEE.

Dan Feng received the Ph.D degree from Huazhong University of Science and Technology, Wuhan, China, in 1997. She is currently a professor of School of Computer, Huazhong University of Science and Technology, Wuhan, China. She is the principal scientist of the the National Grand Fundamental Research 973 Program of China "Research on the organization and key technologies of the Storage System on the next generation Internet." Her research interests include computer architecture, storage system, parallel I/O, massive storage and performance evaluation.

David Swanson received a Ph.D. in physical (computational) chemistry at the University of Nebraska-Lincoln (UNL) in 1995, after which he worked as an NSF-NATO postdoctoral fellow at the Technical University of Wroclaw, Poland, in 1996, and subsequently as a National Research Council Research Associate at the Naval Research Laboratory in Washington, DC, from 1997–1998. In 1999 he returned to UNL where he directs the Research Computing Facility and currently serves as an Assistant Research Professor in the Department of Computer Science and Engineering. The Office of Naval Research, the National Science Foundation, and the State of Nebraska have supported his research in areas such as large-scale scientific simulation and distributed systems.