A Load-Aware Data Placement Policy on Cluster File System

Yu Wang[1,2], Jing Xing[1], Jin Xiong[1], and Dan Meng[1]

[1] National Research Center for Intelligent Computing Systems,
Institute of Computing Technology, Chinese Academy of Sciences
[2] Graduate University of Chinese Academy of Sciences
{wangyu,xingjing,xj,md}@ncic.ac.cn

Abstract. In a large-scale cluster system with many applications running on it, cluster-wide I/O access workload disparity and disk saturation on only some storage servers have been the severe performance bottleneck that deteriorates the system I/O performance. As a result, the system response time will increase and the throughput of the system will decrease drastically. In this paper, we present a load-aware data placement policy that will distribute data across the storage servers based on the load of each server and automatically migrate data from heavily-loaded servers to lightly-loaded servers. This policy is adaptive and self-managing. It operates without any prior knowledge of application access workload characteristics or the capabilities of storage servers. It can make full use of the aggregate disk bandwidth of all storage servers efficiently. Performance evaluation shows that our policy will improve the aggregate I/O bandwidth by 10%-20% compared with random data placement policy especially under mixed workloads.

Keywords: Cluster File System, Data Placement.

1 Introduction

Cluster computing has been widely used in many fields such as weather research, geophysics, finance, information service and so on [1]. These applications not only need high-performance computing ability, but also generate large amount of data, hence I/O performance has a significant influence on the execution time of the applications. Moreover, the data set size of many applications can be petabyte and is increasing gradually now and in the future [2, 3, 4].

In almost all of the environments, the file size distribution is approximately lognormal or a mixture of lognormal [5, 6, 7]. The file size among files differs so great that it will arise some problems if files are not placed properly. If large files from applications are all stored on the same servers, the disk space of these servers will be exhausted while that of others will be under-used. However, if the server's disk is over-utilized, its I/O performance will degrade acutely [8] and even worse, the write operation may fail if there is no space left in the disk for the newly created files. It will be the bottleneck and failure point of the whole system despite the average disk usage of the system is so low that it has much large space left really. Additionally, in

E. Altman and W. Shi (Eds.): NPC 2011, LNCS 6985, pp. 17–31, 2011.

some environments, the read/write requests are distributed uniformly among storage servers [9], while in others, the distribution is so uneven [6] that the data access workloads are very different among storage servers. The I/O performance of those I/O-overloaded servers will decline because there are many read/write requests waiting for disk I/O. Hence, those servers will be the performance bottleneck of the whole system and the response time will increase drastically. With the rapidly growing I/O demand of data-intensive or I/o-intensive applications [10], the impact of disk I/O on overall system performance is becoming more and more important.

In general, I/O load imbalance and disk space saturation will engender the system bottleneck, decrease the read/write performance and increase system response time. It is a vital issue to avoid such situations as possible .An efficient data placement policy is potentially one of the best ways to relieve or eliminate these problems by distributing and arranging the application's data among the storage servers intelligently and efficiently.

In this paper, we propose a load-aware data placement policy to adaptive distribute data among storage servers. It defines method to evaluation the load of storage system, which will consider both workload and disk utilization. Based on the evaluation result, it chooses the location of a new created file. Multiple locations will be chosen if the file keeps multiple replicas. When load imbalance or disk saturation occurs, the evaluation method can also guide data migration process to achieve system balance. The evaluation shows that our data placement policy can diminish the situation of disk saturation. By adjusting data distribution through evaluating system load, higher I/O throughput can be achieved as we can maintain workload balance in most of the time.

The rest of this paper is organized as follows. In section 2 that follows, related work in the literature is briefly reviewed. In section 3, we describe the prototype file system DCFS3. Section 4 describes the load-aware data placement policy for cluster file system. Evaluation is given in section 5. Finally we will conclude this paper in section 6.

2 Related Work

Pseudo-random hash method has been widely used in file systems and storage systems to distribute files among storage servers. All of the hash related works do not consider the I/O access difference of storage servers. However, some of them have taken the heterogeneity of disks into account. Chord [12] adapts consistent hashing[11] to select the data storage location. However, it does not take the difference of I/O workload and storage usage among servers into account. Choy [13] has proposed extendible hashing for perfect distribution of data to disks. But it does not support necessary features such as weighting of storage servers and data replication. Honicky[14] presents the RUSH algorithms for balanced distribution of data to disks, which support weighting of disks and data replication. However, these algorithms rely on iteration for producing the same sequence of numbers regardless of the number actually required, and the large-scale iterations increase the allocation time. CRUSH[15] most closely resembles the RUSH family of algorithms upon which it is based. It fully generalizes the useful elements of $RUSH_P$ and $RUSH_T$ which resolving previous issues. But it also does not consider I/O

load balance of servers. Brinkmann[16] proposes an efficient, distributed data placement policy of pseudo-random distribution data to multiple disks using partitioning of the unit ranges for SAN. Wu and Burns [17] build on this work to provide a storage system which can balance metadata workload by moving file sets among storage servers. These methods can relocate data sets to rebalance the load. However, they all do not support the placement of replicas.

A large body of work can be found in the literature that addresses the issue of balancing the load of disk I/O. Many dynamic load management techniques for parallel systems are designed for homogeneous clusters. Workload is transferred from heavily loaded servers to lightly loaded ones. Another series of techniques allow for server heterogeneity but requires all servers to periodically broadcast I/O load and available storage space. Utopia [18] uses prior knowledge of non-uniform server capabilities and makes load adjustment decisions based on the available CPU utilization, free memory size and disk bandwidth updates for each server. Zhu et al [19] use knowledge of server storage capacity and employ a metric that combines available CPU cycles and storage capacity of each server to select a server to process the incoming requests. Lee et al. [20] propose two file assignment algorithms that balance the load across all disks. The I/O load balancing policies in these studies have been shown to be effective in improving overall system performance by fully utilizing the available hard drives. Xiao Qin et al. [21] develop two effective I/O-aware load balancing schemes, which make it possible to balance I/O load by assigning I/O-intensive sequential and parallel jobs to the node with light I/O load. However, the above techniques are insufficient for automatic computing platforms due to the lack of adaptability. Dynamo [22] is a distributed replicated storage system used in Amazon. It maps the virtual node which is I/O-intensive to the server node which can provide higher I/O bandwidth to achieve the I/O load balance. It also distributes all virtual storage nodes to the hash ring uniformly to achieve the balance of storage utilization. It is only suitable when the request size is similar. In the system Sorrento, Tang et al [24] weight two factors of storage utilization and I/O workload of the providers to get an integral load factor, which will be used to choose the segment's storage location. The system dynamically adjusts the segments' location to balance I/O load and storage usage among the providers. But it relies on the prior-knowledge of the applications to determine how to combine these two factors together efficiently and its I/O workload factor can only reflect the disk I/O status of one special moment. BASIL[28] achieves load balancing by modeling of workloads and devices, however, it only solve balance problem with focus on I/O workload and not space utilization.

Existing data placement methods considers either disk usage or I/O load. And some of them requires almost equal request size, or rely on prior knowledge of workload characteristics. We propose an adaptive and self-managing data placement method that takes both disk storage and I/O load into account. Our method is designed for common environments with very diverse I/O requests. And it operates without any prior knowledge of workload characteristics or the capabilities of storage servers.

3 Background

In this section, we will give a brief overview of the basic architectural features of DCFS3 which is the fundamental system for this work. DCFS3 is a high performance cluster file

system designed for supercomputers. It consists of Metadata Server (MDS), Object-based Storage Device (OSD) and Client, which are connected by AMP (Asynchronous Message Passing). The architecture of DCFS3 is shown in Figure 1.

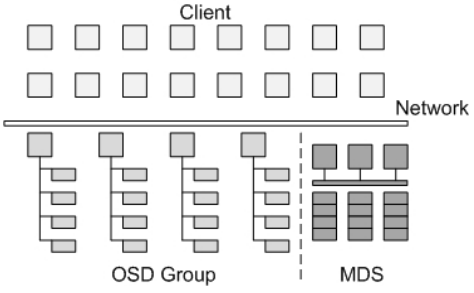

Fig. 1. DCFS3 architecture

The primary goals of this architecture are scalability, performance and reliability. DCFS3 maximizes the separation of file system metadata management from the storage of file data. Metadata information and operations are collectively managed by MDS. In DCFS3, MDS doesn't only maintain namespace metadata, but also maintain data location. To provide high performance management, MDS holds all the metadata in memory [27]. Clients interact directly with OSDs to perform file I/O operations. All OSDs are divided into multiple OSD groups, and OSDs within the same group have the same hardware configurations. When creating a new file, MDS will choose an OSD group and stripe the new file across all the OSD within the same group. To support high availability, replica technology is implemented in data storage. If a file will be maintained with 3 replicas, MDS will choose 3 OSD groups to place its 3 replicas when creating this file.

4 Load-Aware Data Placement and Load Balance Mechanism

In this section, we first describe how to measure the storage server's load and how to choose file's storage location based on the load. Then we will discuses load balance mechanism.

4.1 Storage Server Load

Due to unexpected data access workload and big difference of file size and file life time, the following two conditions are unavoidable. One is that some storage servers are saturated while others are relatively vacant. The other is that some servers' disk I/O is heavy while others' is light. In these two conditions, those overloaded servers will be the bottleneck of the whole system. As a result the throughput will decrease and the response time will increase. Therefore, we should combine the server's I/O workload and its disk storage utilization efficiently when measuring its load.

We use an ordered pair to define the load f, $f = (f_l, f_s)$ where f_l is I/O workload of the server and f_s is disk storage utilization. In our data placement policy and load

balance mechanism, we will first consider its I/O workload f_l, then its disk utilization f_s. In the following sections, we will treat these two factors separately and orderly.

For the server's I/O workload, we measure it by the average bandwidth utilization ratio of the disk during the two contiguous load collections. By extending the method of Linux command *iostat* when computing the disk bandwidth utilization ratio of a special hard disk during a small time interval, we can get the average disk bandwidth utilization ratio $f_l \in [0,1]$ during the load collection time interval. For the server's storage utilization, we measure it by disk space usage of the disk, which can be computed from the fields by calling *vfs_statfs*. We use it as the storage utilization factor $f_s \in [0,1]$.

An efficient data placement policy should consider the disparity of I/O workload among storage servers when arranging the data locations. To fully utilize all storage server's disk bandwidth, the I/O workload should be averaged among all storage servers. Meanwhile, storage utilization must also be taken into account to avoid disk saturation, which is also the cause of performance bottleneck.

4.2 Loads-Aware Data Placement

When creating a new file, the Client will send a message to MDS which will determine the OSD location for this new file and MDS must avoid the previously mentioned bottleneck problems. Due to this, we propose a location selection method based on the probability distribution of the loads on all OSDs. Each OSD group has a load factor *f* which we discussed before. The selection probability of one OSD group is determined by its load proportion of all OSD groups load. The larger the percentage, the smaller probability it will be selected. The proportion is negatively correlated to the selection probability. During the load collection interval, all the newly created files will be distributed among all the OSD groups statistically.

Let's take an example to explain how this method works. We assume that there are 3 OSD groups and every OSD group has a load 0.2, 0.8, 0.4. The load can be either *fl* or *fs*. The selection probability of each OSD group is like the following:

$$p_1 = \frac{1/0.2}{1/0.2 + 1/0.8 + 1/0.4} = \frac{4}{7}, p_2 = \frac{1/0.8}{1/0.2 + 1/0.8 + 1/0.4} = \frac{1}{7} \quad p_3 = \frac{1/0.4}{1/0.2 + 1/0.8 + 1/0.4} = \frac{2}{7}.$$

Then [0, 1) can be divided into 3 sub-ranges, [0, 4/7) is assigned to OSD group 1, [4/7, 5/7) is assigned to OSD group 2, [5/7, 1) is assigned to OSD group 3. When creating a new file, a random real number $\varepsilon \in [0,1)$ will be generated and the OSD group whose range contains ε will be selected. With this method, the newly created files will be distributed among all OSD groups statistically according to their load.

To maximize the system performance by fully utilizing the aggregate disk bandwidth, when evaluating the load of storage system, we firstly considered the workload and then the disk utilization. After an OSD group is chosen in Formula-1, its dist utilization must be checked to skip disk saturation. If any OSD in the group exceeds 95%, it will be discarded and the selection procedure will repeat. The pseudo-code is shown in Figure 2.

```
1: Algorithm: File Storage Based on Load (FSBL)
2: find_flag = 0;
3: unavail_osd_group = Φ ;
4: cur_group_ok = 1;
5: if fl_i∈[μ(f_l)−3×σ(f_l),μ(f_l)+3×σ(f_l)],∀i=1,2,⋯,group_num then
6:         calculate p_i with Formula- 2 for each group i;
7: else
8:         calculate p_i with Formula -1 for each group i;
9: endif
10: while find_flag == 0 do
11:        generate a real random number ε in [0, 1);
12:        for each group i in the system do
13:                 if ε < Σ_{j=0}^{i} p_j then
14:                         cur_group_id = i ;
15:                         break;
16:                 endif
17:        endfor
18:        for each osd in osd_group[cur_group_id] osd_ i do
19:                 if osd_i.disk_usage >= SATURATION_FLAG then
20:                         cur_group_ok = 0;
21:                         break;
22:                 endif
23:        endfor
24:        if cur_group_ok == 1 then
25:                 find_flag = 1;
26:                 return cur_group_id;
27:        else
28:                 if unavail_osd_group == all_osd_group then
29:                         retrun -1;     // file allocation failed.
30:                 else
31:                         add cur_group_id to unavail_osd_group;
32:                         continue;
33:                 endif
34:        endif
35: endwhile
```

Fig. 2. Pseudo-code of load-aware data placement

1) Check if I/O workload of OSD groups in the system is balanced.
2) If not, use the I/O workload as the load factor to choose the file storage location. By applying the above method, the probability of selecting OSD group i is as follows, N is the number of OSD groups.

$$p_i = \frac{1/f_{li}}{\sum_{j=1}^{N} 1/f_{li}}, i = 1,2,\cdots,N \qquad (Formula-1)$$

3) If yes, use the storage utilization as the load factor to choose the file storage location. By applying the above method, the probability of selecting OSD group i is

$$p_i = \frac{1/f_{si}}{\sum_{j=1}^{N} 1/f_{si}}, i = 1,2,\cdots,N \qquad (Formula-2)$$

4) When we get an OSD group based on the p_i and the Random value $\varepsilon \in [0,1)$ (Algorithm FSBL line 10-17), we need to check the disk usage of all OSDs in this group (Algorithm FSBL line 18-23). If there is someone whose disk usage exceeds 95%, then go back to step 1) to choose another OSD group (Algorithm FSBL line 24-26). If all the selections fail, it returns error as it indicates that the average disk usage of the system has exceeded 95% (Algorithm FSBL line 28-29). The system need to be expanded by mean of adding new storage servers.

4.3 Load Balance Mechanism

After files are stored based on storage server's load, disparity of I/O workload and disk saturation may also take place as the application workload is keep changing and unpredictable. We first identify these two conditions and then take actions as data migration to eliminate them.

4.3.1 Load Balance Measurement

The above two circumstances should be differentiated separately in that they are two different aspects that influence the system performance. For I/O workload imbalance, we identify it by using the general method that combines mean deviation and standard deviation, that is $\mu(f) \pm C \times \sigma(f)$. The constant C can be adjusted. Through evaluation in section 5.1 we set it to 3 to achieve the best balance between performance and efficiency. Then we can get the confidential interval $[\mu(f_l) - 3 \times \sigma(f_l), \mu(f_l) + 3 \times \sigma(f_l)]$. When I/O workload of all storage servers is within the range, it means that I/O workload of the whole system is balanced. If there is one server whose I/O workload goes beyond the upper bound, it indicates that this server's I/O workload is so heavy that I/O workload of the whole system is imbalance. In this situation, data migration must be triggered to rebalance the load. In our system, we will use OSD group as unit to determine its I/O workload balance. Because the data of a file is striped across an OSD group and the variance of file access frequency results in I/O workload difference among OSD groups. And I/O workload of OSDs within the same OSD group is almost balanced. Hence, load balance of all OSD groups is equivalent to system-wide load balance.

For the server disk space saturation, we identify it by determining whether the disk usage is more than or equal to a SATURATION_FLAG, which we set it as 95% to achieve balance between disk utilization and disk availability. In a cluster system which runs with many applications, there will be numerous fragments in the disk. When the disk usage ratio exceeds 95%, not only the write performance will decrease rapidly, but also the server is in danger of running out of disk. This server will be the access bottleneck or the write failure point. Herein, some data from the saturated disk should migrate to other disks as soon as possible.

MDS collects the two load information periodically, and determines whether the load is balance. If not, the data migration will happen.

4.4 Data Migration

Each object file has a dynamic access frequency, which will change over time. A hot object file is the one that is being actively accessed recently, while a cold object file is the one that has not been accessed for quite a while. Usually, the hot ones are more likely to be accessed in the near future than the cold ones. The last access time (LAT) will be used to measure the temperature of the object file, that is, a more recent LAT stands for a higher temperature while an ever long LAT represents a lower temperature. And the distribution of object files in relation to temperature is typically bimodal. Almost all of the object files are either hot or cold, with few lukewarm ones between them.

We should adopt different data migration strategies for access overload and disk saturation respectively. However, they are all including the following four key aspects. The pseudo-code is shown in Figure 3 and it includes two sub-algorithms DMDS and DMII.

1) Data Migration Occasion. We will trigger the data migration operation under two situations. When MDS has got the system's load information, it first check whether there is a server whose disk usage ratio has achieved or exceeded 95% (Algorithm DMDS line 2-7). If there is, it should check if there exists some server whose disk usage is smaller than 95%. If it exists, the data migration will happen, otherwise, it indicates that the disk usage of all the servers is greater than 95%. At this situation, new storage devices need to be added to the cluster file system to provide continuous high performance disk I/O. Second, if the workload of a OSD group beyond the confidential interval as in Algorithm DMII line 2-3, data migration must be triggered to cool the hot OSD group.
2) Data Migration Source and Destination. That means data will be moved from which OSD and to which OSD. For I/O workload imbalance, data will be moved out from OSD group with the highest I/O workload and to one OSD group with the lowest I/O workload (Algorithm DMII line 4). Each OSD within the source OSD group and the destination OSD group will be one-to-one corresponded as the source and destination. For server disk saturation, its data will be moved to one or several OSDs whose disk usage is low (Algorithm DMDS line 8-23). The reason that we choose several destinations is to avoid the destination to be the bottleneck after data migration.
3) Data Migration Object. That means what data and how much should be migrated. For I/O workload imbalance, hot data will be migrated preferably. The proportion of data migrated can be adjusted according to the data scale of applications (Algorithm DMII line 5-11). For server disk saturation, cold data will be migrated preferably so that the impact to the normal file access will be minimized. We can compute the optimal quantity of migrated data with reference to the OSD's current disk usage, the average disk usage of the whole system and the destination OSD's disk usage ratio (Algorithm DMDS line 24-31).
4) Data Migration Manner. That is how the data will be migrated. For I/O workload imbalance, we have determined the one-to-one relationship of the source OSD and the destination OSD. Data migration will take place among every pair of these OSDs in parallel. And within every OSD, multi-thread will be used to move object files. For server disk saturation, the source OSD will use multi-thread to migrate some object files from itself to other OSDs.

```
1: Algorithm: data migration
2: collect_osd_load(osd_disk_usage[], osd_ioload[]); //osd_disk_usage[] is ascendant
3: call Algorithm DMDS    // to handle the situation of disk saturation
4: call Algorithm DMII    // to handle the situation of I/O workload imbalance

1: Algorithm: Data Migration of Disk Saturation (DMDS)
2: osd_disk_saturation[] = Φ ;
3: for each osd osd_i in system do
4:       if osd_i.disk_usage >= SATURATION_FLAG then
5:                add osd_i to osd_disk_saturation;
6:       endif
7: endfor
8: for each osd osd_i in osd_disk_saturation do
9:       data_migration_quantity=
10:               osd_i. disk_capacity*(osd_i.disk_usage–system_average_disk_usage);
11:      index = temp = 0;
12:      initilize dest_osd_data[];
13:      for each osd osd_iter in osd_disk_usage[] do
14:               avail_space=
15:               osd_iter.disk_capacity*(system_average_disk_usage-osd_iter.disk_usage);
16:               temp += avail_space;
17:               if temp <= data_migration_quantity then
18:                        dest_osd_data[index ++] = avail_space;
19:               else
20:                        dest_osd_data[index ++] = temp - data_migration_quantity;
21:                        break;
22:               endif
23:      endfor
24:      total_migration_size = 0;
25:      for osd_iter = 0 to index do
26:               migrate cold object files preferably to the osd_iter;
27:               update total_migration_size;
28:               if total_migration_size >= dest_osd_data[osd_iter] then
29:                        break;
30:               endif
31:      endfor
32: endfor

1: Algorithm: Data Migration of Ioload Imbalance (DMII)
2: for each group osd_group_i in system do
3:    if fl_i ∉[μ(f_l)−3×σ(f_l), μ(f_l)+3×σ(f_l)], ∀i=1,2,⋯,group_num then
4:       find the osd group with the lowest ioload: dest_osd_group ;
5:       for osd_index=0 to nr_osd_in_group in osd_group_i do
6:          total_migration_size = 0;
7:          migrate hot files from osd_group_i[osd_index] to dest_osd_group[osd_index];
8:          update total_migration_size;
9:          if total_migration_size >= osd_group_i[osd_index].total_size*5% then
10:            break;
11:         endif
12:      endfor
13:   endif
14: endfor
```

Fig. 3. Pseudo-code of data migration

5 Performance Evaluation

This section evaluates the performance of our load-aware data placement policy. The system configuration of DCFS3 consists of one MDS, 3 OSD groups with 2 OSDs in each group and six Clients. MDS is configured with two AMD Opteron 2.2 GHz

processors and 2GB RAM. All OSDs and Clients are configured of virtual servers created by VMware which include one Intel Xeon Processor (2.0GHz) and 1 GB RAM. All of the servers have one 146GB, 10k rpm Seagate SCSI disk and are connected by Gigabit Ethernet.

5.1 Load Balance Interval Constant

We have shown that constant *C* of the confidential interval is set to 3 when determining the I/O load balance. We will compare different constant set (*C*=1, 2, 3, 4) from running time, data migration frequency and data migration quantity. We simulate the burst I/O requests in some occasion of scientific computing applications [9] to evaluate the impact of constant *C*. File number that will be created is 500. The file size distribution is suitable for the lognormal distribution [9]. The total size of each Client is about 22GB. Table 1 shows the test results. The interval range of constant 4 is so broad that almost all servers I/O workload belong to the interval even if the I/O workload disparity is extremely large and running time increases by 6%. The running time of constant 1 and 2 increases by 8% and 5% because there are so many data migration operations and a large amount of data has been migrated. And there are some data thrashing that consumes the system disk bandwidth. Therefore, 3 is the optimal value of constant *C*.

Table 1. Test results of constant *C*

C	Running time	Migration frequency	Migration quantity
1	+8%	8/10	4.8GB
2	+5%	5/10	2.9GB
3	0	1/10	402MB
4	+6%	0/10	0

5.2 Throughput

We simulate three scenarios of system I/O workload balance, imbalance [9] and no I/O workload to compare the system throughput between random data placement (RDP) and load-aware data placement policy with (LADP-with migration) or without data migration (LADP-without migration). For load imbalance, we make the following scene: the average disk bandwidth utilization of the first two OSDs is 100%, the middle two OSDs is 50%, and the last two OSDs has no I/O workload. For load balance, we make the following scene: the average disk bandwidth utilization of all OSDs is 40-50%. The test will be taken under 1, 2, 4, 6 client configurations and all clients execute the test example as in section 5.1 concurrently.

Figure 4 shows throughput under circumstance of no I/O workload. We can see from the figure that throughput of LADP-without migration improves by 3.3%, 5.02%, 5.12% and 5.4% compared with RDP. Figure 5 shows throughput under circumstances of balanced load. We can see from the figure that throughput of LADP-without migration improves by 2.5%, 3.0%, 3.7% and 5.6% compared with

RDP. Figure 6 shows throughput under of load imbalance. We can see from the figure that throughput of LADP-with migration improves by 2.3%, 17.5%, 18.1% and 21.2% compared with RDP and throughput of LADP-without migration improves by 1.9%, 9.1%, 10.3% and 11.2% compared with RDP.

In the circumstance of no load and load balance, the system will undertake some I/O burden because several clients are writing files concurrently. LADP-without migration can consider the real-time load distribution of the system and choose the suitable servers to store the newly created files. Hence, its throughput will be improved for uniformly utilizing the system aggregate disk bandwidth. In circumstance of load imbalance, the system will undertake mixed I/O burden, not only because several clients are writing files concurrently but also other applications have brought out the current load imbalance. LADP-without migration can consider the current load distribution of the system and choose the suitable file storage servers. In addition to this, LADP-with migration can fully utilize the system aggregate disk bandwidth and improve throughput by data migration.

Therefore, the system resource contention on data servers can significantly degrade the overall I/O performance, and skipping hot-spots can substantially improve the I/O performance when load on storage servers is highly imbalanced.

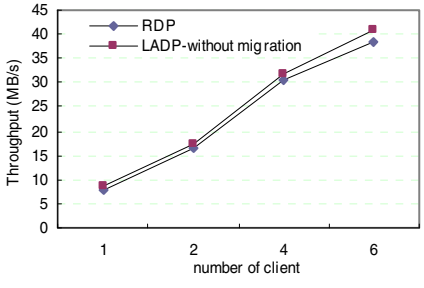

Fig. 4. Throughput when the current system has no workload

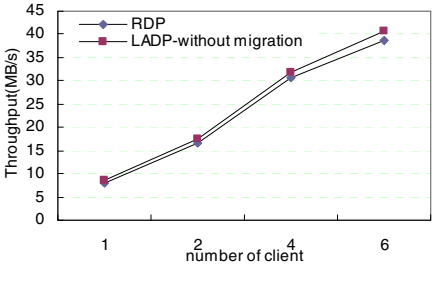

Fig. 5. Throughput when the current system has some workload and the load is balance

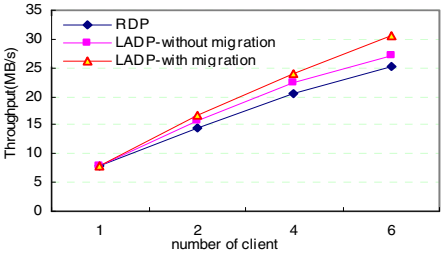

Fig. 6. Throughput when the current system has some workload but the load is not balance

5.3 mpiBLAST Application

The mpiBLAST [25] is the parallel implementation of the biological sequence search tool BLAST. By utilizing distributed computational resources through database fragmentation, query fragmentation and parallel I/O, mpiBLAST improves performance of BLAST by several orders of magnitudes. Since DCFS3 provides cluster-wide shared name space and mpiBLAST runs directly on them through parallel I/O interfaces, the worker does not need the copying procedure in the parallel I/O implementations. It is separated into two steps. The first one is to divide the whole database into multiple segments, that is *mpiformatdb*. The second one is *mpiexec*, which is each worker searches one database fragment using the entire query. Previous research has shown that the length of 90% of the query sequences used by biologists is within the range of 300-600 characters [26] and the second step usually takes a few minutes. But the first step will take dozens of minutes. Therefore, our focus is to decrease the run time of the first step *mpiformatdb*.

We use the sequence database month.est_others and est_mouse, which are the nucleotide sequence databases in non-redundant form, freely available from download at NCBI web site. We compare the running time between RDP and LADP-with migration. The fragmentation number is 25, 50 and 100, and Figure 7 and 8 show the experiment results. In these three fragmentation number, the fragmentation time decreases by 10.2%, 13.9%, and 17.1% for month.est_others and the fragmentation time decreases by 7.5%, 9.1%, and 9.2%s for est_mouse. Our experiments show that with the load-aware data placement, mpiBLAST greatly outperforms random data placement.

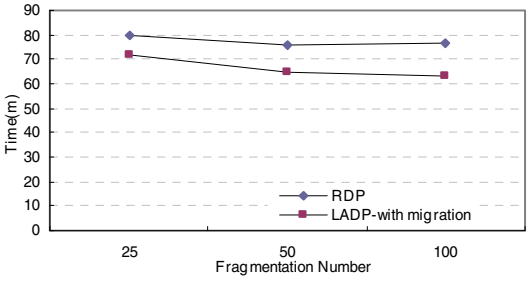

Fig. 7. mpiBLAST fragmentation time of month.est_others

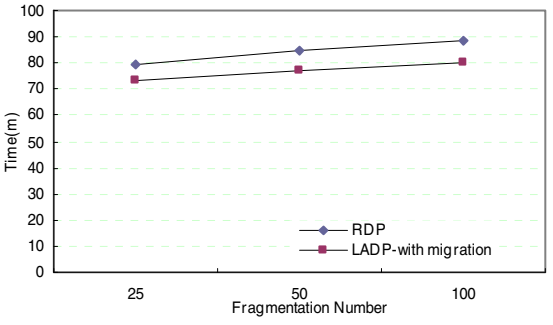

Fig. 8. mpiBLAST fragmentation time of est_mouse

6 Conclusion

In this paper, we present a load-aware data placement policy which is implemented on a large-scale cluster file system DCFS3. In order to avoid performance bottleneck, both I/O workload and storage utilization should be considered when measuring the load of storage system. Data migration is triggered to balance the system-wide load when it is imbalanced. We distinguish two different scenes of load imbalance and take different actions. We propose a probability-based load-aware data placement method. When creating a new file, its location is chosen based on I/O workload first, then the storage utilization. Our experiments show that by considering these two load factors and data migration, disk bandwidth of all servers can be fully utilized and the throughput increases obviously.

In this work, we only evaluate the influence of disk resource contention. Clearly, the load conditions of CPU, memory and network can also influence the throughput and response time. We will study the impact of contention of these resources in the future work.

Acknowledgments. This work is supported by the National High-Tech Research and Development Program of China under grant no. 2006AA01A102 and grant no. 2009AA01Z139.

References

1. http://www.top500.org/
2. DOE National Nuclear Security Administration and the DOE National Security Agency. SGS file system (April 2001)
3. Kramer, W.T.C., Shoshani, A., Agarwal, D.A., Draney, B.R., Jin, G., Butler, G.F., Hules, J.A.: Deep scientific computing requires deep data. IBM J. Res. Dev. 48(2), 209–232 (2004)
4. http://public.web.cern.ch/public/
5. Wang, F., Xin, Q., Hong, B., Brandt, S.A., Miller, E.L., Long, D.D.E., McLarty, T.T.: File system workload analysis for large scale scientific computing applications. In:

Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies, College Park, MD (April 2004)
6. Leung, A.W., Pasupathy, S., Goodson, G., Miller, E.L.: Measurement and analysis of large scale network file system workloads. In: Proceedings of the 2008 USENIX Annual Technical Conference (June 2008)
7. Evans, K.M., Kuenning, G.H.: Irregularities in file-size distributions. In: Proceedings of the 2nd International Symposiusm on Performance Evaluation of Computer and Telecommunication Systems, SPECTS (July 2002)
8. McKusick, M., Joy, W., Leffler, S., Fabry, R.: A Fast File System for UNIX. ACM Trans. on Computer Systems 2(3), 181–197 (1984)
9. Lawrence Livermore National Laboratory. IOR software, `http://www.llnl.gov/icc/lc/siop/downloads/download.html`
10. Zhu, Y., Jiang, H., Qin, X., Swanson, D.: A Case Study of Parallel I/O for Biological Sequence Analysis on Linux Clusters. In: Proceedings of Cluster 2003, Hong Kong, December 1-4 (2003)
11. Karger, D., Lehman, E., Leighton, T., Levine, M., Levin, D., Panigraphy, R.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In: Proceedings of ACM Symposium on Theory of Computing (STOC 1997), pp. 654–663 (1997)
12. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001), San Diego, CA (August 2001)
13. Choy, D.M., Fagin, R., Stockmeyer, L.: Efficiently extendible mappings for balanced data distribution. Algorithmica 16, 215–232 (1996)
14. Honicky, R.J., Miller, E.L.: Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In: Proceedings of the 18th International Parallel & Distributed Processing Symposium (IPDPS 2004), Santa Fe, NM (April 2004)
15. Weil, S.A., Brandt, S.A., Miller, E.L., Maltzahn, C.: CRUSH: Controlled, scalable, decentralized placement of replicated data. In: Proc. of the 2006 ACM/IEEE Conference on Supercomputing, Tampa, FL (November 2006)
16. Brinkmann, A., Salzwedel, K., Scheideler, C.: Efficient, distributed data placement strategies for storage area networks. In: In Proceedings of the 12th ACM Symposium on Parallel Algorithms and Architectures SPAA (2000)
17. Wu, C., Lau, F.: Load Balancing in Parallel Computers: Theory and Practice. Kluwer Academic Publishers, Boston (1997)
18. Zhou, S., Wang, J., Zheng, X., Delisle, P.: Utopia: A load-sharing facility for large heterogeneous distributed computing systems. Software Practice and Experience 23(12) (1993)
19. Zhu, H., Yang, T., Zheng, Q., Watson, D., Ibarra, O.H.: Adaptive load sharing for clustered digital library servers. International Journal on Digital Libraries 2(4) (2000)
20. Lee, L., Scheauermann, P., Vingralek, R.: File Assignment in Parallel I/O Systems with Minimal Variance of Service time. IEEE Trans. on Computers 49
21. Xiao, Q., Jiang, H., Zhu, Y., Swanson, D.: Toward load balancing support for I/O intensive parallel jobs in a cluster of workstation. In: Proc. of the 5th IEEE International Conference Cluster Computing, Hong Kong (December 14, 2003)
22. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: “Dynamo: Amazon’s Highly Available Key-Value Store”. In: ACM SOSP (2007)

23. Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T.: Bigtable: A Distributed Storage System for Structured Data. In: Proc. of OSDI 2006 (2006)
24. Tang, H., Gulbeden, A., Chu, L.: A Self-Organizing Storage Cluster for Parallel Data-Intensive Applications. In: Proc. of SC 2004, PA (November 2004)
25. http://www.mpiblast.org/
26. Pedretti, K.T., Casavant, T.L., Roberts, C.A.: Three complementary approaches to parallelization of local BLAST service on workstation clusters. In: Malyshkin, V.E. (ed.) PaCT 1999. LNCS, vol. 1662, Springer, Heidelberg (1999)
27. Xing, J., Xiong, J., Ma, J., Sun, N.: Main Memory Metadata Server for Large Distributed File Systems. In: GCC 2008 (October 2008)
28. Gulati, A., Kumar, C., Ahmad, I., Kumar, K.: BASIL: automated IO load balancing across storage devices. In: Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST 2010), pp. 13–13. USENIX Association, Berkeley, CA, USA (2010)