Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2012, Article ID 396387, 15 pages
doi:10.1155/2012/396387

Research Article

A Cloud-Computing-Based Data Placement Strategy in High-Speed Railway

Hanning Wang,[1] Weixiang Xu,[2] Futian Wang,[1] and Chaolong Jia[1]

[1] *State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, Beijing 100044, China*
[2] *School of Traffic and Transportation, Beijing JiaoTong University, Beijing 100044, China*

Correspondence should be addressed to Hanning Wang, 06114158@bjtu.edu.cn

Received 28 August 2012; Revised 31 October 2012; Accepted 21 November 2012

Academic Editor: Wuhong Wang

As an important component of China's transportation data sharing system, high-speed railway data sharing is a typical application of data-intensive computing. Currently, most high-speed railway data is shared in cloud computing environment. Thus, there is an urgent need for an effective cloud-computing-based data placement strategy in high-speed railway. In this paper, a new data placement strategy named hierarchical structure data placement strategy is proposed. The proposed method combines the semidefinite programming algorithm with the dynamic interval mapping algorithm. The semi-definite programming algorithm is suitable for the placement of files with various replications, ensuring that different replications of a file are placed on different storage devices, while the dynamic interval mapping algorithm ensures better self-adaptability of the data storage system. A hierarchical data placement strategy is proposed for large-scale networks. In this paper, a new theoretical analysis is provided, which is put in comparison with several other previous data placement approaches, showing the efficacy of the new analysis in several experiments.

1. Introduction

With the development and popularity of information technology, internet is gradually growing into various computing platforms. Cloud computing is a typical network computing mode, which emphasizes the scalability and availability of running large-scale application in virtual computing environment [1]. A large-scale network application based on cloud computing demonstrates features of distribution, heterogeneousness, and the trend of data intensity [2]. In cloud computing environment, data storage and operation is provided as a service [3]. There are various types of data, including common files, large binary files such as virtual machine image file, formatted data like XML, and relational data in database. Thus, a distributed storage service of cloud computing has to take large-scale storage mechanism for various data types into account, as well as the performance, reliability, security, and simplicity of data operation. As an important component of China's transportation science data sharing system, high-speed railway data is the key to optimizing the operating organization. High-speed railway data sharing system has the characteristics of typical data-intensive application [4–6], to which the management of a large amount of distributed data is crucial. It is mainly reflected in the fact that the data size it processes is often up to TB or even PB level, which includes both existing input data source and the intermediate/final result data produced in the process.

In the process of implementing and executing data-intensive applications under the environment of cloud computing, as well as the process of establishing a large-scale storage system to meet the demand of a fast growing data storage volume, the main challenge is how to effectively distribute data at Petabyte level to hundreds of thousands of storage devices. Thus, an effective data placement algorithm is needed.

2. Goal of Designing High-Speed Railway Data Placement Strategy

The network storage system under cloud computing environment consists of thousands, and even ten thousands of storage devices. Different systems have different underlying devices, for example, the storage device set could be chunk device disk for SAN and GFPS, or OSD (object storage device) for object storage systems Lustre and ActiveScale, or PC for PVFS and P2P [7]. A data placement strategy mainly solves the problem of selecting storage device for data storage. An effective mechanism shall be adopted to establish the mapping relationship between data sets and storage device sets. Then, data sets generated by applications in the storage system are placed into different storage device sets. Meanwhile, certain particular goals need to be met, and different data placement strategies are designed for different purposes. For example, the stripping technology in RAID is mainly designed to acquire aggregated I/O bandwidth. The strategy of placing several replications of the data into different devices is mainly for the purpose of fault tolerance and data reliability improvement. Distributing data equally could realize a more balanced I/O load.

The high-speed railway data placement strategy under cloud computing environment is designed to meet the following goals.

2.1. Fairness

The size of the data stored in each device is proportional to the storage volume of that device [8].

2.2. Self-Adaptability

With the elapse of time, the volume of storage devices is dynamic and varied. Take the case of adding a new device and the case of deleting an existing device for example. When the scale of the storage system changes, a data placement strategy is applied to reorganize the data, making the data distributed to device sets satisfy the fairness criteria over again. Furthermore, it needs to be ensured that the migrated data volume is close to the optimal migration data volume. This would reduce the overheads of data migration. The optimal data volume to migrate is equal to the data volume that is acquired by the added device, or equal to the data volume on the deleted device. The self-adaptability of the data placement strategy is measured by the ratio of its actual migrated data volume to the optimal migration data volume. Therefore, the ratio value of 1.0 represents the optimal condition.

2.3. Redundancy

Getting several replications copied for the data, or enabling the data remain accessible through the use of erasure code when one of the replications is lost. So that fairness can balance the IO loads, self-adaptability can reensure fairness in accordance with storage scale change, and the data size migrated and the IO bandwidth occupied can also be decreased. Finally, the data reliability can be improved.

2.4. Availability

It is crucial that a system could be normally accessed in all cases. Once the system is unavailable, all functions would fail to perform normally. To improve system availability, it is necessary to regularly have the data location adjusted according the availability of storage devices, thus maximizing the system's availability [9].

2.5. Reliability

It indicates whether the system could be normally accessed during a certain period of time. As the large-scale storage system contains thousands of storage devices, the probability of disk failure is rather high. When applying a data placement strategy, indicators of reliability such as data size need to be used in designing parameters of the placement strategy. Thus, a storage system with higher reliability is obtained.

2.6. Space-Time Effectiveness

It means that few time and space is used in calculating data location along with the data placement strategy.

When designing the data placement strategy for large-scale network storage system, certain particular goals need to be met depending on different application demands. However, it is impossible to meet all goals at the same time.

3. Related Work

Some data management systems under the cloud computing environment have already been emerged currently, for example, Google File System [10] and Hadoop [11, 12], both of which have hidden the infrastructure used to store application data from the user. Google File System is mainly used for Web search application, but not for process application under the cloud computing environment. Hadoop is a more commonly distributed file system, which is used by many companies including Amazon [13] and Facebook. When Hadoop file system receives a file, the system will automatically separate the file into several chunks, each of which is randomly placed into a cluster. Cumulus project [14] has proposed a cloud architecture of single data center environment. However, the above-mentioned cloud data management systems did not study the data placement problem of data-intensive process applications under the cloud environment. Finally, let us look into several examples of existing popular large-scale data storage systems. Commercial Parallel File System (CPFS) [15, 16] divides a file into data chunks of the same size, which are stored on different disks of the file system in the form of rotation. Parallel Virtual File System (PVFS) [17] with open-source codes divides the file into strip and chunk and adopts the method of placing sliced data on multiple IO nodes in rotation. The data slice size of PVFS is a constant. PVFS data does not have any fault tolerance function. Panasas [18] is an object-oriented file system, where the data are allocated to underlying smart object storage device (OSD) in the unit of object [19]. A file is divided into strips, and each strip unit is stored on multiple OSD in the form of object. Upon initial placement, objects are fairly distributed between OSD devices using the random method.

PanFS, developed by the Panasas Company, is a Linux cluster file system based on object storage [20]. It is the core part of ActiveScale storage system. At first, these file systems divide the file into strips, and then allocate each strip to the underlying smart OSD in unit of object. The distribution of files across multiple OSDs is realized based on the round-robin algorithm. The size of the data object is random, and it could increase accordingly with the increase of file size without modifying the metadata mapping chart on the metadata server.

The object-oriented file system Lustre is a transparent global file system. The Lustre file system treats the file as an object that is located by the metadata server, which then directs the actual file I/O request to the corresponding object storage targets (OSTs). Since a technology is adopted where the metadata is separated from the storage data, the computing resources could be fully separated from storage resources [21]. Thus, the client could focus on the I/O request from users and applications. Meanwhile, the OST and metadata server could focus on data reading, transmitting, and writing.

All storage nodes in the COSMOS parallel file system [22] are divided into several strips. Each COSMOS file is stored in a certain strip. And the length and logic chunk length of the strip are related to the disk speed and file access mode of applications. This type of data placement strategy has features such as high performance, large file suitability, and high degree of parallelism. Through the stripped subfile, COSMOS is directly saved on local disks in the form of common JFS file. Thus, the expression direct management of disks is avoided while increases the overheads when entering into the core of VFS/Vnode for the second time.

4. Study and Analysis of Existing Data Placement Strategy

Here are some currently popular data placement algorithms. Standard hashing is the simplest homogeneous (indicating that all storage devices have the same volume) placement algorithm, which can ensure fairness. But when the storage scale varies, the locations of all the data have to be changed as well.

Consistent Hashing [23] uses the *hash* function to map a device to the continuum, and then the hash function $h_2(x)$ is used to evenly map the data to that continuum. Then data is allocated to that device represented by the node which is nearest to the data itself. Since devices are not evenly distributed on the continuum, each device is virtualized to $k \log |N|$ devices (where k is a constant) to ensure the fairness of data allocation. The data size of this device is equal to the total size of data allocated to virtual nodes. When a storage device is added to the system, only parts of the data located on the left and right neighbor nodes are to be migrated to that device. Consistent Hashing has a high degree of self-adaptability, and this mechanism takes up a space of $O(n^2 \log n)$.

As a matter of fact, the data storage under the cloud computing environment is heterogeneous, which means that there is great volume discrepancy between storage devices. Therefore, the consistent Hashing algorithm is improved as follows: the virtual nodes on the continuum are allocated based on the weight of a device. The device with greater weight covers more virtual nodes on the continuum. However, this approach would introduce large amount of virtual nodes in a heterogeneous storage system with extremely significant weight discrepancy, and this would increase the space complexity of the algorithm.

In order to solve the problem of space waste with consistent hashing, a segmentation method based on the unit interval is brought forward. In this method, the interval is divided into unit subintervals with identical length, and each device occupies an interval. When a device is added, part of the data on other devices is migrated to this new device. When a device is deleted, the data on the last device is equally migrated to remaining devices, and the data on the device to be deleted is migrated to the last device, and then the very device is finally deleted. In this way, the fairness could be guaranteed. During the device addition, the data migration volume is 1 time the optimal data migration volume. During the device deletion, the data migration volume is 2 times the optimal data migration volume. $O(\log n)$ steps are needed to locate a specific data, which takes longer time than locating data with consistent hashing, but only a space of $O(n \log n)$ digit is occupied here. Compared to the consistent hashing, this algorithm exchanges the time for space. It is not suitable for a storage system that has a demanding requirement for rapid data search. Furthermore, this algorithm's self-adaptability is not as high as consistent hashing.

In order to solve the problem of space waste resulting from the consistent hashing's introduction of virtual nodes, the linear method and logarithm method are proposed. In the linear method, the weight for a device is introduced similarly. Suppose w_i indicates the weight for device i, and $d_i(x)$ indicates the distance between the hash values of device i and data x. The linear method would select the device, which has the lowest value of $H_i = d_i(x)/w_i$, to store data x.

As the storage scale changes, the linear method could guarantee that data would only be migrated between the added/deleted device and other devices. There would be no data migration between other devices. The logarithm seeks to find a device that brings the the minimum value for the function $H_i = -(\ln(1-d_i(x)))/w_i$. In the absence of the virtual nodes, the logar ithm performs with better fairness than linear one, but it would take a longer time to locate data.

As a result, a data object placement algorithm based on the dynamic interval mapping is proposed [22]. The unit space is divided into multiple subintervals according to the device's weight. And then a mapping relationship between the device and subinterval is established. Based on the interval where the data falls in, data is allocated to the device corresponding to that interval. This approach has better fairness and self-adaptability, where the time consumption in locating data will increase with the expansion of the number of storage devices. But if the number of storage devices is extremely large, when a device is added or deleted, the system is required to communicate with all other storage devices for data migration, which will bring tremendous overheads. Furthermore, the time consumption in locating data will increase with the expansion of the number of storage devices as well.

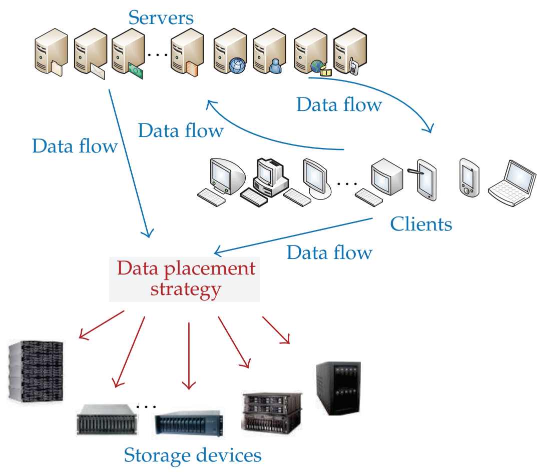

Figure 1: Direct management.

5. A Hierarchical Structure Based on Cloud Computing

With the expansion of network scale, the number of data storage devices keeps increasing. The existing data placement algorithm is insufficient to address the system's self-adaptability. Adding new or removing existing devices could lead to a new data placement over again, which will result in an increase of data migration overheads so that the occupation of IO bandwidth is inevitable [24, 25]. Therefore, the data reliability cannot be guaranteed, and the overheads are too large to use a duplicate copy for data reliability assurance [26]. Thus, a data placement strategy based on a hierarchical structure is brought forward in this paper for the purpose of making up for the shortage of existing data placement algorithm, addressing the system's self-adaptability, guaranteeing data reliability, and improving the efficiency of data access.

In the proposed approach, each single storage device is directly managed through a common data placement strategy, as shown in Figure 1.

The hierarchical structure could reduce the time consumption of data query and location. As a result, a data placement strategy of hierarchical structure is more suitable for data management under cloud computing environment, as shown in Figure 2.

It is assumed in this paper that large amount of storage devices are heterogeneous in the storage system under cloud computing environment. That is to say, the storage volume of every single device is different from one another. These storage devices are grouped into several device sets that count relatively less in number. When storing the file data, it is first located on a device set, and then the file data is stored inside the device set. So that the locality of file data within this device set is ensured, and this helps improve the speed for data reading and writing.

In the case of data placement for file with several replications, different replications of the same file should be placed onto different device sets as many as possible. So that when a certain storage device within a storage device set cannot function properly, the client could get the target file data located on other device sets as usual. Thus, it could improve the availability and reliability of files.

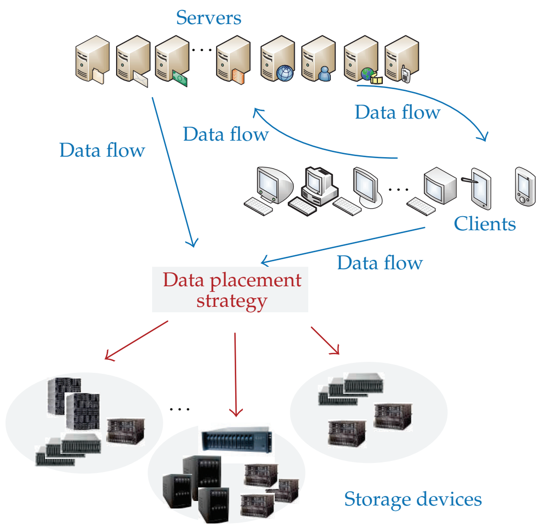

Figure 2: Hierarchical structure.

In the data placement strategy of hierarchical structure, when a storage device is to be added, it is designed to allocate the newly added storage device to a device set; when a storage device is to be deleted from a device set, the migration data could be constrained to other different storage devices within that device set. This would reduce the overheads in communication with large amount of storage devices within other storage device sets. The I/O bandwidth occupation would be reduced during data migration. When an aged storage device needs to be replaced with a new one, firstly the data on the original device is transferred to the new device. Since the new storage device in replacement outperforms the original one in both the storage volume and read/write performance, fairness is disrupted between each storage device within the device set in regard of data storage. Therefore, data is migrated between the new storage device and other ones within that device set in order to meet fairness criteria between each storage device within that set.

6. Algorithm Description

We would group larger amount of heterogeneous storage devices into less amount of device sets. The number of already grouped sets is to be kept unchanged. The total storage volume of different storage device sets should remain the same. Files and their various numbers of duplicate copies are to be mapped to different device sets for storage using an algorithm based on semidefinite programming. Files are sliced within the device set, and then the data slices are mapped to devices with different volumes in the set using a dynamic interval mapping method.

6.1. The Semidefinite Programming Algorithm

So that the problem of data copies placement is converted into a problem of seeking semidefinite programming, different copies of a file are placed on different storage device sets. Meanwhile, according to the algorithm, the file is located on a device set and stored into

```
Formal description of a semi-definite programming problem
Solution:
        min $\sum_{i \neq j}^{n} C(i,j) \cdot l_i \cdot l_j$
    Satisfying:
    $l_i, l_j$ are unit vectors; $l_{i \cdot i} = 1$
    $l_i$ and $l_j$ form the matrix $L = [l_i \cdot l_j]$ and all its
      eigenvalues are greater than or equal to 0,
     that is, matrix $L$ is semi-definite.
```

Algorithm 1: Formal description.

various devices within the set strip by strip; thus, the file locality is ensured. The file data could be immediately accessed upon one time of locating, so that the file access speed is improved.

Function $C(i,j) = (i,j) = 1$ is right only when i and j are two different copies of the same file, or when j stands for the copy of i. If not, $C(i,j) = 0$. Also, $C(i,j) = 0$ when i equals j. An associated matrix C is constructed using $C(i,j)$. C can represent the relationship between all files, that is, which file owns and which file copies. The Algorithm 1 converts the problem of data copy placement into the formalized description of a semidefinite programming problem.

Solution to the semidefinite programming problem can produce a semidefinite matrix $L = [l_i \cdot l_j]$. And further processing of the semidefinite matrix L can obtain the device sets, where each file copy is stored in the storage system.

6.2. Dynamic Interval Mapping Algorithm

Supposing some device set D contains n devices, that is, $d_1, d_2, \ldots, d_n$. All these n devices have different volumes, respectively, $D_1, D_2, \ldots, D_n$, so that the ratio of the weight of each device volume to the total volume within this device set is $r_i = D_i / D_1 + D_2 + \cdots + D_n$, where $i = 1, 2, \ldots, n$ and $r_0 = 0$. It is known that $\sum_{t=1}^{n} r_i = 1$. So then we segment a subinterval with the length of $w_i = [r_{i-1}, r_{i-1} + r_i]$ for each device D_i in the interval $[0,1]$. When the file is allocated into a device set, it is divided into data chunk sets $S : s_1, s_2, \ldots, s_m$ with the same size, and, then m data chunks are mapped to devices with different weight values in the set (Algorithm 2).

The hash function $h(S) :\rightarrow [0,1]$ is used to map the data chunks to the interval $[0,1]$. If $h(s_i) \in w_i$, then the data chunk s_i is allocated to the device mapped by the interval w_i.

7. Experiment and Analysis

In this paper, the two key algorithms in hierarchical data placement, namely, Semidefinite programming (SDP) algorithm and dynamic interval mapping algorithm, are implemented on Matlab platform. Matrix is the basic unit of Matlab language, which could be directly used in matrix calculation. Therefore, Matlab could be directly applied to solve complicated problems such as optimization or linear programming. The semidefinite programming problem we need to solve in this paper will be described in a mathematic formalized

```
  Pseudocode of the algorithm
Initialization:
  Device set $D$, Data set $S$, and subinterval set $w$;
Input: data chunk $s_i$
Program main: $k = h(s_i)$;
    for ($j = 1; j < n; j + +$)
    if ($k \in w_j$)
    Placing the data chunk $s_i$ on the device $d_j$
    Output: data volume stored on the device $d_j$
```

Algorithm 2: Formal description.

matrix. Furthermore, it is easy to formalize a dynamic interval mapping problem into a formalized matrix, which is suitable to implement in Matlab environment. Meanwhile, Matlab features an abundant toolbox and module set. In order to seek a solution to the semidefinite programming problem, a toolbox that provides support for Matlab to solve SDP problem should be installed.

7.1. Fairness Analysis on Semidefinite Programming Algorithm

Suppose that each file has 5 copies. Then, respectively, distribute 100, 200, 300, and 400 files into 10 device sets and 20 device sets using the semidefinite programming method. The deployment is shown in Figures 3 and 4. The experiment has shown that files can be fairly evenly distributed to multiple device sets using the semidefinite programming. It has been illustrated that this approach could ensure the fairness of file data layout.

7.2. Reliability Analysis on Semidefinite Programming Algorithm

Now let us further discuss the situation for placement of 5 copies of the same file, that is, the problem regarding whether all the 5 copies of the same file have been placed into different device sets. As shown in Table 1, when 400 files (with 2,000 copies) are distributed to 10 and 20 device sets, all the 5 copies of, respectively, 299 and 372 files are completely distributed to 5 different device sets. Other files which do not include these copies failed to do this. There are 2 of 5 copies of one file that are allocated to the same device set. As a result, the semidefinite programming algorithm has shown a better performance to allocate different copies of a file into different storage device sets. Thus, the probability of data loss due to device failure is reduced, and the data reliability is improved. Based on the principle of random function, it can be inferred that the probability for data allocated to each subinterval using the dynamic interval mapping algorithm is proportional to the length of each interval. Similarly, the data volume of all the devices inside the device set is proportional to its overheads. It has been proved that when the storage nodes inside the storage device set change, the dynamic interval mapping method can minimize the overheads of data migration under the condition that the number of storage nodes is not extremely high. So it eliminates the overheads of communicating and migrating data caused by the change of the number of storage nodes, when directly managing a very large amount of storage devices. As a new device is added, the subinterval occupied by each device within a device

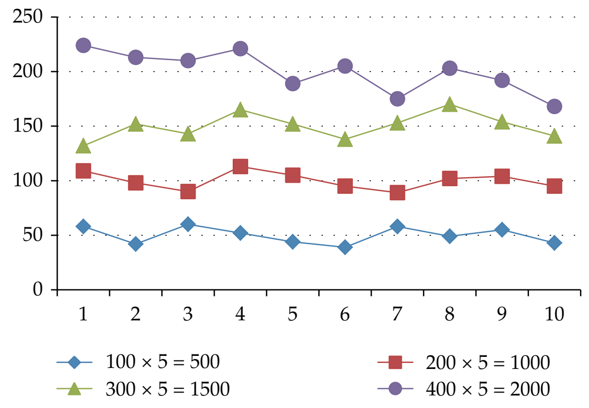

Figure 3: Files distribution over 10 device sets.

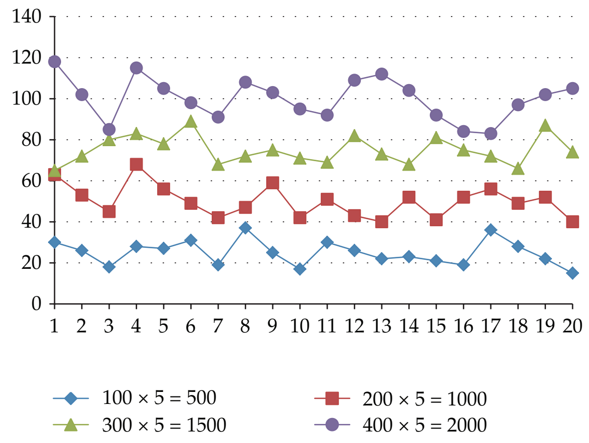

Figure 4: Files distribution over 10 device sets.

set changes correspondingly, reallocating the interval occupied by existing devices and the corresponding data chunk to the new device, in order to realize fairness over again. The overheads of communicating and transferring data are constrained to only those few devices inside the device set.

7.3. Fairness Analysis on Dynamic Interval Mapping Algorithm

Firstly, the fairness of dynamic interval mapping algorithm is tested. Let us take a look at the file data volume stored on each storage device within a device set. When 1,000 files are stored in 10 device sets, 100 files are stored in the no. 5 device set as indicated in Figure 3. Then, we assume that there are 10 storage devices inside the no. 5 device set. And the 100 files are stripped into 1,500 data strips, which are stored onto the 10 storage devices by means of dynamic interval mapping algorithm. For all these 10 storage devices, the percentage of each device's storage volume in their total storage volume, as well as the interval length (w_j) corresponding to that percentage are all shown in Table 2.

Table 1: Distribution situation of replications.

Replications distribution	10 device sets	20 device sets
100 files (500 replications)	79	93
200 files (1000 replications)	154	186
300 files (1500 replications)	237	266
400 files (2000 replications)	299	372

Table 2: Storage percentage of each device relative to the total storage volume.

Device code	No. 1	No. 2	No. 3	No. 4	No. 5	No. 6	No. 7	No. 8	No. 9	No. 10
%	5	10	16	11	20	9	8	10	6	5
Interval	0 ∼ 0.05	0.05 ∼ 0.15	0.15 ∼ 0.31	0.31 ∼ 0.42	0.42 ∼ 0.62	0.62 ∼ 0.71	0.71 ∼ 0.79	0.79 ∼ 0.89	0.89 ∼ 0.95	0.95 ∼ 1

Based on the dynamic interval mapping algorithm and above-mentioned volume of each storage device, the stripped 1,500 data strips are equally stored into these 10 storage devices. The theoretical allocation situation is shown in Table 3.

When implementing the dynamic interval mapping algorithm, the hash function $h(s_i)$ is used to map data chunk s_i to a random number between $(0, 1)$. If $h(s_i) \in w_j$, $(0 < j \leq 10)$, the data chunk s_i is placed into the storage device j. Consequently, all the 1,500 data strips are stored into the 10 storage devices. Comparison between the actual allocation situation and theoretical situation is shown in Figure 5.

7.4. Self-Adaptability Analysis on Dynamic Interval Mapping Algorithm

Let us test the self-adaptability of dynamic interval mapping algorithm. The cases of removing a storage device and adding a new storage device are, respectively, considered.

7.4.1. Removing a Storage Device

Let us examine the file data volume migrated between other storage devices when a storage device is deleted from a device set. For example, Table 4 shows the situation when no. 7 device is deleted from the device set. The percentage of each remaining device's storage volume, and the interval length corresponding to that percentage are shown in the Table 4.

When removing the no. 7 storage device, the data on that device is migrated to the remaining 9 storage devices. The situation of change relating to actual data migration is shown in Figure 6.

From Figure 7 above, we can see that after deletion of no. 7 device, those remaining storage devices in the device set can still store data according to the percentage of each one's storage volume in the total remaining storage volume.

Table 3: Theoretical data allocation on 10 storage devices.

Device code	No. 1	No. 2	No. 3	No. 4	No. 5	No. 6	No. 7	No. 8	No. 9	No. 10
Data allocation	75	150	240	165	300	135	120	150	90	75

Table 4

Device code	No. 1	No. 2	No. 3	No. 4	No. 5	No. 6	No. 7	No. 8	No. 9	No. 10
%	5.4	10.9	17.4	12	21.7	9.8	0	10.9	6.5	5.4
Interval	0 ~ 0.054	0.054 ~ 0.163	0.163 ~ 0.337	0.337 ~ 0.457	0.457 ~ 0.674	0.674 ~ 0.772	0 ~ 0	0.797 ~ 0.881	0.881 ~ 0.946	0.946 ~ 1

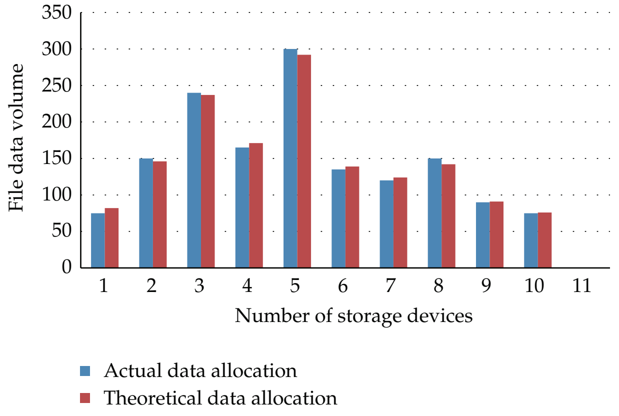

Figure 5: Comparison between actual and theoretical allocation.

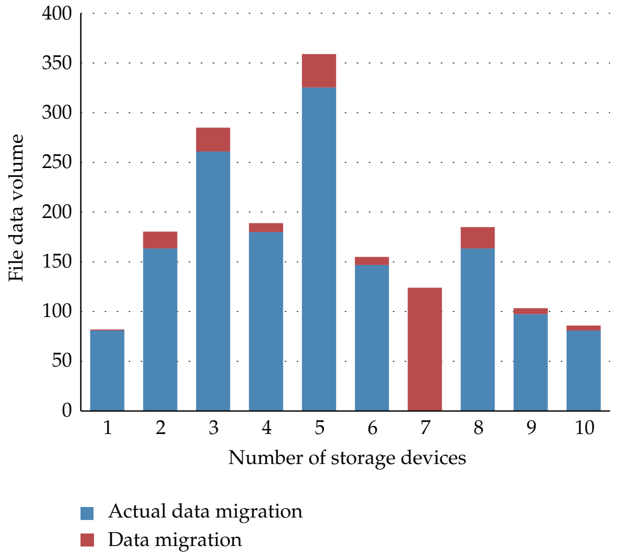

Figure 6: Data migration after deleting no. 7 device.

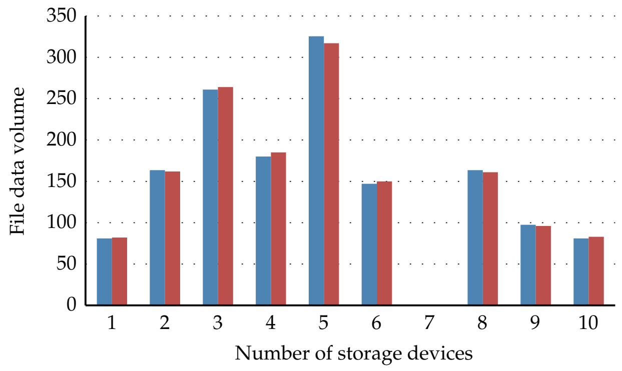

Figure 7: Comparison between the actual and theoretical allocation after deleting no. 7 device.

7.4.2. Adding a New Storage Device

Now let us examine the situation when a storage device is added to that device set. The case is similar to the above-mentioned situation when a storage device is removed. We would follow these steps below.

(1) First, when a new storage device is added to that device set, the percentage of each device relative to the total storage volume is recalculated. And the interval length (w_j) corresponding to that percentage is redefined as well.

(2) Then the difference between the original interval length and the revised one following the addition of a storage device is calculated. And the data corresponding to that length difference is what to be migrated into the new storage device.

(3) Dynamic interval mapping algorithm is used to place the migrated data into the newly added storage device. The Hash function $h(s_i)$ is used to map the data strip s_i to a random number between $(0, 1)$. If $h(s_i) \in w_j, (0 < j \leq 10)$, the data chunk s_i is placed onto device j.

8. Conclusion

A hierarchical structure data placement algorithm under the cloud computing environment is proposed in this paper. The proposed algorithm combines the semidefinite programming algorithm with the dynamic interval mapping method. The semidefinite programming method would distribute the data of a file with replications to grouped device sets. Experiments have demonstrated that this method could guarantee the data reliability and high-speed file accessibility. The dynamic interval mapping method could distribute data fairly to devices with different volumes inside the device set. The self-adaptability of this method is proved theoretically.

Acknowledgments

This research was supported by the National Natural Science Foundation of China (General Projects) (Grant no. 61272029), National Key Technology R&D Program (Grant No.: 2009BAG12A10), China Railway Ministry Major Program (2008G017-A), and State Key

Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, China (Contract no. RCS2009ZT007).

References

[1] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[2] C. Miceli, M. Miceli, S. Jha, H. Kaiser, and A. Merzky, "Programming abstractions for data intensive computing on clouds and grids," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09)*, pp. 478–483, Shanghai, China, May 2009.

[3] K. Liua and L. J. Dong, "Research on cloud data storage technology and its architecture implementation," *Procedia Engineering*, vol. 29, pp. 133–137, 2012.

[4] J. Ma, "Managing metadata for digital projects," *Library Collections, Acquisition and Technical Services*, vol. 30, no. 1-2, pp. 3–17, 2006.

[5] W. Wang, W. Zhang, H. Guo, H. Bubb, and K. Ikeuchi, "A safety-based approaching behavioural model with various driving characteristics," *Transportation Research C*, vol. 19, no. 6, pp. 1202–1214, 2011.

[6] W. Wang, *Vehicle's Man-Machine Interaction Safety and Driver ASSiStance*, China Communications Press, Beijing, China, 2012.

[7] V. T. Tran, G. Antoniu, B. Nicolae, L. Bougé, and O. Tatebe, "Towards a grid file system based on a large-scale BLOB management service," in *Grids, P2P and Services Computing*, pp. 7–19, 2010.

[8] L. Amsaleg, M. J. Franklin, A. Tomasic, and T. Urhan, "Improving responsiveness for wide-area data access," *Data Engineering*, vol. 20, pp. 3–11, 1997.

[9] A. Deshpande and Z. Ives, "Adaptive query processing," *Foundations and Trends in Databases*, vol. 1, no. 1, pp. 1–140, 2007.

[10] S. Ghemawat, H. Gobioff, and S. T. Leung, "The google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pp. 29–43, ACM, New York, NY, USA, October 2003.

[11] Apache Hadoop, http://hadoop.apache.org/.

[12] "Hadoop distributed file system," http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf.

[13] Amazon Elastic MapReduce, http://aws.amazon.com/elasticmapreduce/.

[14] J. Tao, M. Kunze, A. C. Castellanos, L. Wang, D. Kramer, and W. Karl, "Scientific cloud computing: early definition and experience," in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC'08)*, pp. 825–830, Dalian, China, September 2008.

[15] Y. Zhai, M. Liu, J. Zhai, and X. Ma, "Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications," in *Proceedings of the SC'11 State of the Practice Reports*, 2011.

[16] C. H. C. Evangelinos, "Cloud computing for parallel scientific HPC applications: feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2," in *Proceedings of the 1st Workshop on Cloud Computing and Its Applications (CCA'08)*, October 2008.

[17] P. H. Carns, I. W. Ligon, R. Ross, and R. Thakur, "PVFS: a parallel file system for linux clusters," in *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, Ga, USA, 2000.

[18] "Cloud computing with parallel storage," http://www.panasas.com/blog/cloud-computing-with-parallel-storage.

[19] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, 2010.

[20] N. Maheshwari, R. Nanduri, and V. Varma, "Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 119–127, 2012.

[21] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[22] R. Chaiken, B. Jenkins, P. A. Larson, and B. Ramsey, "SCOPE: easy and efficient parallel processing of massive data sets," in *Proceedings of the VLDB Endowment VLDB Endowment Hompage Archive*, vol. 1-2, pp. 1265–1276, 2008.

[23] D. Karger, E. Lehman, T. Leighton, and R. Panigrahy, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," in *Proceedings of the 29th*

annual ACM Symposium on Theory of Computing (STOC'97), pp. 654–663, ACM, New York, NY, USA, 1999.

[24] J. Dhok, N. Maheshwari, and V. Varma, "Learning based opportunistic admission control algorithm for MapReduce as a service," in *Proceedings of the 3rd India Software Engineering Conference (ISEC'10)*, pp. 153–160, ACM, Mysore, India, February 2010.

[25] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[26] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters," in *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, pp. 541–548, Rio De Janeiro, Brazil, May 2007.