A Reliable And Efficient Data Migration Method Based On GlusterFS

Li Li, Gang Yin, Tao Wang

Computer Science, National University of Defense Technology,Changsha, China
dylan-lili@foxmail.com,jack_nudt@163.com,taowang.2005@outlook.com

Abstract— Big data storage and high speed data access have become the main performance bottleneck for many big data applications. The higher speed data access and lower cost for storage must be required than some applications having small-scale data. Distributed hierarchical storage provides a good storage way to speed data access and lower cost. But it a data migration method which you choose decide the performance of distributed hierarchical storage system because data migration occurs frequently in hierarchical storage systems. There are many data migration methods, which most of those cannot ensure data utterly integrity after data migration. In this paper, we invent a reliable and efficient data migration method to ensure the utterly integrity of migrated data by MD5 checksum and improve performance of data migration by the pipeline technology. Through adjusting parameters, we get the best performance of data migration by using pipeline in our storage system which is a hierarchical storage system based on GlusterFS.

1 Introduction

In recent years, with the rapid development and popularity of computer science and information technology, the scale of the industry application system expands rapidly and the industry application data is generated by explosive growth. For example, the US New York Stock Exchange generates about 1TB transaction data every day, the Internet archives store about 2PB data, and increasing at a rate of at least 20TB every mouth [1]. Different periods of data have different meanings for application having massive data. The data of newly generated have the much higher frequency to access than the older data [2]. According to statistics, 80% of the disk data is not often accessed, but which is very important and must be completely stored. How to store these data in order to achieve the requirements of higher speed data access and lower cost.

At present, SSD (Solid State Disks) and HDD (Hard Disk Drives) are the main storage devices. The SSD have high data access speed, but the price of unit storage is very expensive and the lifespan is short. Though the price of unit storage is much cheaper and the lifespan is longer for the HDD disks, they cannot meet performance requirements because of the low speed of the data access for the application having massive data. Hierarchical storage [3] can be a good balance of storage costs and the speed of data access as application needed. The data is not accessed for a long period of time which is stored in the HDD and the frequently accessed data is stored in the SSD. Since only a small amount of data needs to be accessed frequently over a period of time, a small amount of SSD are required and the most of data is stored HDD. The hierarchical storage strategy have a good performance in accessing data with almost cost of HDD.

For the application having massive data, it's necessary to use the distributed storage to store massive data because a single disk storage capacity should be insufficient.

GlusterFS [4] proposed by Z RESEARCH Company is an open source distributed file system, which is widely used in cloud storage system. GlusterFS has PB class file cluster storage capacity by linking to many kinds of cheap x86 hosts with Infiniband RDMA [5] or TCP/ IP [6] protocol to a large-scale parallel network file system. In view of the large differences in data access frequency between different data, GlusterFS is used to build a hierarchical distributed storage cluster. A few SSD and a lot of HDD with different access speeds disks are connected into a cluster. The online volume having high speed of data access is created on the SSD and the offline volume having low unit storage costs is created on the HDD disks. In this way, the storage cluster provide good performance of data access under constraints of storage capacity and hardware costs.

In this distributed hierarchical storage system, data access frequency is dynamic. In order to enable the system to provide better performance all the time, we need to migrate frequently data between different storage devices. There are many methods to migrate data between offline volume and online volume, the most commonly used way is

to migrate data file for offline database. But statistics show that large data file migration maybe fail or migrated data is not available because of various types of problems. For example, unexpected computer downtime during migration, or varying degrees of data loss will result in inconsistent data [7]. Therefore, in order to ensure data availability before and after large data file migration, data file must be verified after data migration.

At present, there are two main aspects to ensure the data availability after migration. One is comparing some specific data consistence before and after data migration by using spot checks and partial statistics [8]. It is a sampling approach, which can only ensure the sampling part of the data consistency, not all data. Another way is ensuring the data consistency in transfer process by using SHA1 [9], MD5 checksum [10] or other data verification method. This approach can ensure all data integrity after data migration because each bit of all data is checked in data transfer. However, it takes much more time to verify the data compared to the only migration data.

In this paper, we using a parallel method to verify and migrate data with pipelining. We want to take a little extra time to finish verify data available and integrity. Under the condition of ensuring data available and integrity, our method can significantly enhance the migration efficiency.

2 Related Work

2.1 GlusterFS

GlusterFS is mainly composed of storage server (brick server), client and NFS/Samba storage gateway. Obviously, there is no metadata server component in the GlusterFS architecture, which is largest design advantage than other file system. It is a decisive significance for improving the performance, reliability and stability of the entire system. GlusterFS builds a storage cluster through TCP/IP and InfiniBand RDMA high-speed network interconnection. The client can access data through the native Glusterfs protocol by using the FUSE (File System in User Space) module mount GlusterFS storage cluster into local file system [11]. So it is easy to access data stored in GlusterFS cluster by mount point in local file system.

2.2 MD5 Checksum

The MD5 checksum checks the correctness of the data by performing a hash operation on the received transmission data because arbitrary two strings should not have the same hash value (i.e., it is strong possibility that hash value is not the same, and it is difficult to create artificially two hash values of the same string). An MD5 checksum verifys the correctness of the data by performing a hash operation on the received transmission data. The hash value is calculated by received data is compared to the hash value transmitted with the data. If the two values are the same, which indicates that the transmitted data is intact and not been tampered (The hash value has not been tampered on the premise); if not, which indicates that the migration is fail and remove the destination file. Therefore, MD5 checksum can be used in data files migration to ensure the consistency of data.

2.3 The Traditional Method of Verifying data

There are two main aspects to ensure the data consistency and availability after data migration. Someone verify data consistency based on guard [12], which has three main steps. Firstly, some specific data is added into data as the guard data. Then, we check the consistency of the guard data after data migration. We regard as this data migration is successful if the guard data is complete. If not, this process of migration must result in some data was lost. But it is not always right this data migration is successful if the guard data is complete, because we verify the specific data not the whole data and some data maybe be lost aside from guard data. Therefore, we do not use this way to verify the migrated data for our migration method.

The people named Bin Wei Tennyson X. Chen [13] verify every bytes of migrated data by MD5 checksum. They use the popular MD5 algorithm to generate the hash codes with every bytes of source data before it be migrated. And then, the data migration is started. The MD5 checksum of migrated data is generated and compared with source data hash codes when the process of data migration is finished. They regard as this data migration is successful if their MD5 checksum were same. If not, this process of migration must be failed because the data is inconsistency. This way certainly ensure the data consistency, but the cost is very high and it is almost three times higher than only data migration. If we need to migrate the very large data, the time spent cannot be accepted.

From the above, we need to a data migration method which not only ensure the data consistency after data migration but also have a better efficiency.

3 Our method

In this section, we design a reliable and efficient data migration method based on distributed hierarchical storage system with GlusterFS. We improve the method of data migration, in order to greatly increase efficiency of data migration under the condition of ensuring the data available and integrity after migration.

3.1 A reliable data migration frame

Our method is used to big data frequent migration which has a strict requirements about data consistency from one storage cluster to another storage cluster. Our purpose is to ensure data consistency and provide a good performance. In order to

achieve that aim, the source data is split into many pieces to execute the migration and verification in parallel.
Firstly, we split the source data into many pieces, and then a piece of source data is migrated and another is verified at the same time. We will try to design parallel steps as much as possible to improve efficiency.

3.2 Mathematical expression of the improved method

The traditional method of data migration from source data files to destination path with MD5 checksum verifying data, which can be divided into the following three serial steps:

- Step 1: Obtain the MD5 checksum of the source data file;
- Step 2: Copy the source data file to the destination path;
- Step 3: Obtain the MD5 checksum of the destination data file, then compare it with the MD5 checksum of the source data file in step one. If they are the same, the migration succeeded and the destination data is available. Otherwise, the data is invalid, delete the destination data file to avoid produce dirty data.

For large data files, time consumption mainly includes reading the source file I/O time T1 and the CPU time T2 of calculating the MD5 checksum of the source file contents in step 1, the time T1 of reading the source file and writing destination file I/O time T3 in step 2, read the destination file I/O time T4 and the CPU time T5 of calculating the MD5 checksum of the destination file contents in step 3, so the total time consumption is:

$$All\_Time=T1+T2+T1+T3+T4+T5 \quad (1)$$

According to the study of the serial migration method mentioned above, we found that step 2 which is copying the source data file to the destination path contains reading the source file into buffer and then writing buffer contents into the destination file, the step 3 which is obtain the MD5 checksum of the destination data file contains reading the destination file into buffer and then calculating the value of destination file MD5 checksum. It is worse for efficiency that the contents of the destination file exists in the buffer twice in step 2 and step3, so we can optimize this method by calculating the value of destination file MD5 checksum when it is in buffer in step 2 and removing the step 3. The step 2 is changed to step 2' which contains reading the source file into buffer, writing buffer contents into the destination file and then calculating the value of buffer contents MD5 checksum. This approach can reduce the time consumption of T3 in step 3, which is the time of reading the destination file, so the total consumption of improved method becomes:

$$All\_Time=T1+T2+T1+T3+T5 \quad (2)$$

T2 and T5 are the time of calculating the value of source file and destination file MD5 checksum. Normally, the source file and the destination file are the same, so T2 = T5 and this formula can be written as:

$$All\_Time =T1+T2+T1+T3+T2 \quad (3)$$

We cannot read all the contents of large file into buffer because it will result in memory overflow and performance degradation. We should read and process a segment contents of large file until the end of the file. Definition t1 is a part of T1 which is the time of reading a segment file into buffer, similarly, t2 is a part of T2 which is the time of calculating the value of buffer contents MD5 checksum and t3 is a part of T3 which is the time of writing buffer contents into file. Therefore, the time consumption of step 1 should be written as:

$$T1+T2=t1+t2+t1+t2+\cdots \quad (4)$$

Similarly, the time consumption of step 2 ' should be written as:

$$T1+T3+T2=t1+t3+t2+t1+t3+t2+\cdots \quad (5)$$

According to the improved method, we found that reading a segment source file into buffer and calculating the value of buffer contents MD5 checksum use different computer resource the disk and CPU in step 1. So, we can makes two-stage pipelining [14] optimization step 1. Similarly in step 2', the three substeps that reading the contents of the source file into buffer with I/O resource, calculating the value of buffer contents MD5 checksum with CPU resource and writing the buffer contents into the destination file with I/O resource. Because migration occurs between different speed volumes of data access in this storage cluster, reading file and writing file are executed on different disks, we can makes the three-stage pipelining optimization in step 2'. Therefore, the total time with pipelining is:

$$All\_Time=\max(T1, T2)+\max(T1, T3, T2)+\vartheta \quad (6)$$

The max (T1, T2) represents the maximum value of T1 and T2, max (T1, T3, T2) represents the maximum between T1, T3 and T2, and ϑ represents the additional time between processes switch.

3.3 Software implementation

The data migration method have two parts which one is calculating the source data checksum and the other part is data migration and verify the consistency of data. In order to migrate data with pipelining, we split the source data into many parts to make the migration and verification parallel.

Figure 1 shows the flow diagram of the reliable migration software, the main steps are as follows.

Symbol definition:

- Q1, Q2: The symbol of Q1 and Q2 denote two fixed size buffer queue to cache the unit buffer. Q1 store the

available buffer objects. Q2 store the some parts of source data read into memory.

- Bi (B1, B2, B3 …): Bi denotes the unit buffer object to store the piece of data. It is one of Q1 or Q2 elements.
- P1, P2: P1 and P2 denote two process of reading the source data into buffer and calculating their checksum.
- Similarly, the symbol of q1, q2, bi, p1, p2, p3 denote the same things in second part of migration process.
- S: S denotes the source data.
- D: D denotes the destination data.
- Bi<-Qi or bi<-qi: These denote polling an item from queue named Qi or qi as Bi or bi.
- Bi->Qi or bi->qi: These denote add the item Bi or bi into the Qi or qi queue.
- Pi~: The symbol Pi~ indicates that the process Pi has ended.
- Qi=0: This indicates that the queue Qi is empty.

Step 1: Obtaining the value of source data file MD5 checksum with pipelining.

The fixed size of the two buffer queues Q1 and Q2 is initialized and then Q1 is filled with unit buffer object Bi of constant size. The buffer queues is used for recycling the buffer object to avoid requesting and releasing frequently buffer resulting in performance degradation. The two threads P1 and P2 are executed in parallel to obtain the MD5 checksum of source data file. The thread P1 is a loop which reads the segment contents of source data file into buffer object B1 fetched from the Q1 queue until the end of source data file. The thread P2 is also a loop which calculates the value of the MD5 checksum of the contents of the buffer object B2 fetched from the Q2 queue until the end of thread P1.

Step 2: Copying source data file to destination and obtaining the MD5 checksum of destination data file.

Similarly, the fixed size of the three buffer queues q1,q2 and q3 is initialized and then q1 is filled with unit buffer object bi f constant size which is denoted as q1 (b1, b2, b3 ...). The three threads p1, p2 and p3 are executed in parallel to copy file and obtain MD5 checksum of destination file. The thread p1 and p3 are similar function to P1 and P2 in step 1. The thread p2 is a loop which writes the contents of buffer object b2 fetched from the Q2 queue into destination file until the end of thread p1.

3.4 Parameter Optimization

Buffer is used to improve performance by realizing the parallel optimization. Several buffer queues is used to avoid requesting and releasing frequently buffer resulting in performance degradation in software implementation. There are two parameters impacting obviously performance which are the unit buffer size and queue size. The unit buffer size represents how much contents is processed at a time, such as how much file contents is read into the buffer once and how much contents of buffer is handle to calculate MD5 checksum once. The unit buffer size too small to improve performance because it needs too many times loop for large files. If the unit buffer size increases to a threshold, the performance is not improved as the unit buffer size get bigger because it is limited by other conditions. Meanwhile it not only wastes computer resources, but also decline a little performance because of dealing with large buffer.

The size of the buffer queue is the maximum times of one thread executed continuously. A single process can be executed several times without switching process, to avoid thread switching frequently caused performance degradation. For example, if the buffer queue size is equal to 1, all threads are executed serially. So, if it is too small, the processes switch frequently resulting in time of switch processes increase and performance degradation because of the resource blocking. The time of processes switch reduces as the buffer queue size get bigger. However, if the buffer queue size increases to a threshold, the performance is not improved as it get bigger, because it is limited by other conditions. Therefore, in order to achieve the best performance, these two parameters that unit buffer size and buffer queue size must be adjusted according to the environment.

4 Experiments Design

In this section, we describe the experiment questions, experiment dataset and environment as well as the corresponding evaluation metrics.

4.1 Experiment Questions

In order to explore the efficiency of our improved data migration method. We focus on the following three experiment questions.

- Q1: How to adjust the parameter of buffer queue size and unit buffer size to optimize our data migration method in our specific experiment environment?
- Q2: How much efficiency of our method is improved than original method under migrating data from online volume to offline volume?
- Q3: How much efficiency of our method is improved than original method under migrating data from offline volume to online volume?

For experiment question Q1, we use the way of controlling variables to optimal the two parameters. At the same time, we fix one parameter and change another step by step to get the optimum value when the migration speed get best. For Q2 and Q3, under the optimal parameters conditions, we compare migration speed of our method with original method in different size file. So we can analyse the efficiency of our data migration method.

4.2 Dataset and experiment environment

To address the above research questions and validate our approach, we build the experiment environment which

include four computers with different hardware and choose some different size files ranging from 1M to 20G as dataset.

1) Experiment Dataset: We create some different files ranging from 1M to 20G as dataset, because we need to validate our method efficient in different size files. In order to get the best parameters of buffer queue size and unit buffer size, we set the buffer queue size value from 1 to 100 and unit buffer size from 1K to 16M.

2) Experimental Environment:

The experimental environment is a GlusterFS cluster which consists of two servers configured as shown in Table 1 and four servers configured as shown in Table 2 by Gigabit Ethernet network connection. The high-speed data volume is created which contains two nodes configured in Table 1 and named "online volume". The low-speed data volume is created which contains the four machines configured in Table 2 and named "offline volume".

TABLE I. high-speed data volume server configuration

Hardware	Configuration
CPU	8 Core Intel(R) Xeon(R) CPU E5-1620 0 @ 3.60GHz
Memory	DDR3 1600 MHz 48G
Hard Disk	Samsung SSD 840 EVO 250GB
Network Card	Intel Corporation Ethernet Controller 10-Gigabit X540-AT2
Software	Ubuntu 14.04 LTS, GlusterFS 3.8.1

TABLE II. low-speed data volume server configuration

Hardware	Configuration
CPU	8 Core Intel(R) Xeon(R) CPU E5-1620 0 @ 3.60GHz
Memory	DDR3 1600 MHz 48G
Hard Disk	Western Digital Caviar Green (AF, SATA 6Gb/s) 2TB
Network Card	Intel Corporation Ethernet Controller 10-Gigabit X540-AT2
Software	Ubuntu 14.04 LTS, GlusterFS 3.8.1

4.3 Evaluation Metrics

Our method verify the every bytes of data with MD5 checksum, so the migrated data must be available and integrity. We evaluate the method with efficiency which is migration speed.

5 Experiment Result

In this section, we first build the experiment environment which it creates a GlusterFS storage cluster and then creates two data volume as online data volume and offline data volume to migrate data from one to another. We adjust the parameters of unit buffer size and buffer queue size until our migration method is the fastest speed. Then we compare the efficiency of our migration method with traditional method and analyze the experiment result.

Figure 2 shows the speed of migrating the file that size is equal to 2GB with different unit buffer size and buffer queue size in my experimental environment plotted by the speed of data migration on the horizontal axis and the buffer queue size on the vertical. In the case of constant unit buffer size and the buffer queue size being less than 55, as the buffer queue size increase, the process switching times will reduce and the speed of migration will increase. But when the buffer queue size exceeds 55, as the buffer queue size increase, the speed of migration will decrease slowly. So, in the case of constant unit buffer size, when the buffer queues size is equal to 55, the speed of migration will get fastest.

Similarly, in the case of constant buffer queue, as the unit buffer size increase, the speed of migration will increase, but the increase rate of migration speed decreases. As we can see in Figure 3, the migration speed is roughly same when the unit buffer size equals 8M and 16M. However, the unit buffer size equals 16M that is used much more computer memory resources than the unit buffer size being 8M. So, in the case of constant buffer queue size, when the unit buffer size is equal to 8M, the speed of migration will get fastest. To sum up, under current environment, when the unit buffer size equals to 8M and the buffer queue size is equals to 55, the migration software will get the best performance.

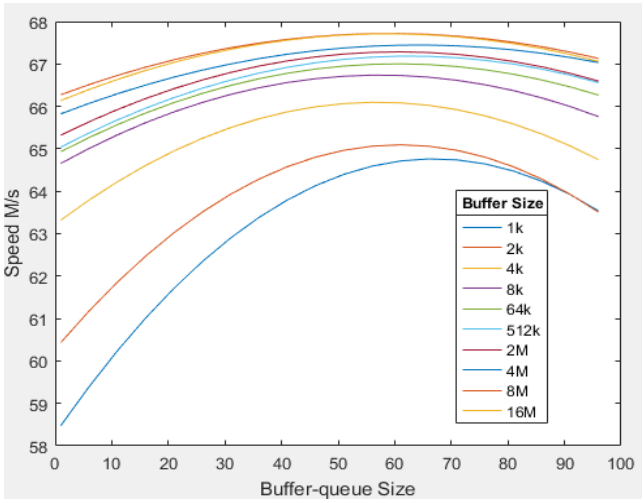

Figure 2. Optimal Parameter

Data migration mainly occurs when the data access frequency changes significantly; for example, data will be frequently accessed and then the data will migrate from the Offline-Volume to Online-Volume, if the Online-Volume storage space is insufficient and then the data that is accessed unlikely for the next period of time is migrated to the Offline-Volume to ensure that Online-Volume has enough space.

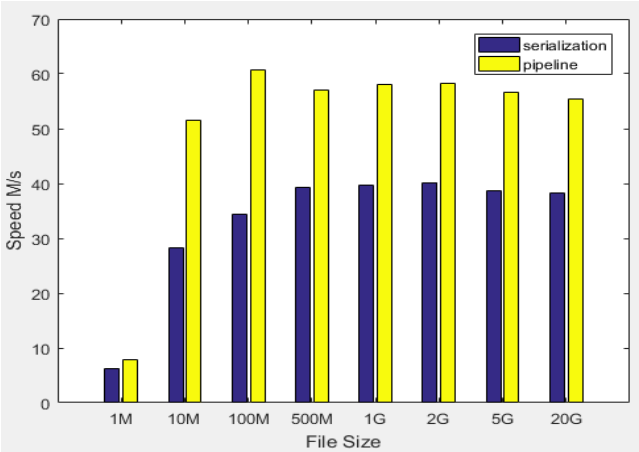

Figure 3. Data of Online-volume migrate to Offline-volume

Figure 3 shows the comparison of the efficiency of serial migration and pipeline migration which migrate data files of different size from online data volumes to offline data volumes and Figure 4 shows the comparison of the efficiency of serial migration and pipeline migration which migrate data files of different size from offline data volumes to online data volumes, which plotted by the speed of data migration on the horizontal axis and the file size on the vertical.

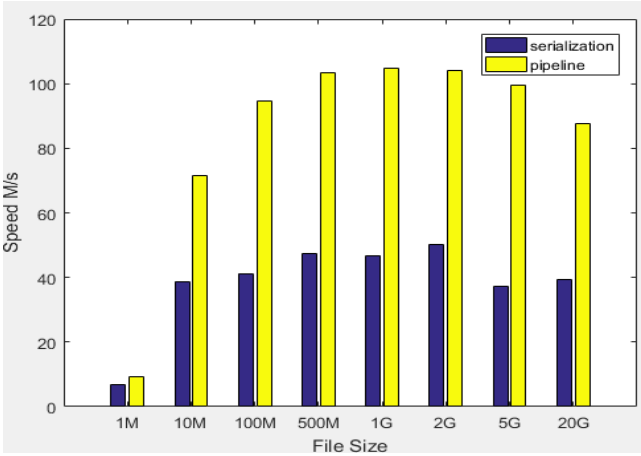

Figure 4. Data of Offline-volume migrate to Online–volume

Conclusion

This reliable data migration method is very important for data migration. It can be used to judge the integrity and availability of the migrated data accurately and quickly. It has a good application prospect for the application of high data integrity. At the same time, the pipeline migration method can greatly enhance the efficiency of data migration. In this experimental environment, the pipeline migration method efficiency of migrating the large data files over 500M from high-speed data volumes to low-speed data volumes can always is higher 2.5 times than the serial migration method from Figure 3. The migration files over 500M can also is higher 1.5 times than the serial migration method even from low-speed data volumes to high-speed data volumes from Figure 4.

References

[1] White T. Hadoop: The definitive guide[M]. 2st edition. " O'Reilly Media, Inc.", 2012.
[2] Song L, Dai H, Ren Y. A Learning Method of Hot-Spot Extent in Multi-Tiered Storage Medium Based on Huge Data Storage File System[J]. Journal of Computer Research & Development, 2012.
[3] Ao L, Yu D, Shu J, et al. A tiered storage system for massive data: TH-TS[J]. Journal of Computer Research & Development, 2011, 48(6):1089-1100.
[4] Gluster Inc. An Introuduction to Gluster Architecture. 2011.
[5] Liu J, Jiang W, Wyckoff P, et al. Design and Implementation of MPICH2 over InfiniBand with RDMA Support[M]. arXiv, 2003.
[6] Socolofsky T J, Kale C J. TCP/IP tutorial[J]. Technology, 1991.
[7] Wang G, Wang D, Li W, et al. Data Migration Technology Research Based on Big Data Environment[J]. Microcomputer Applications, 2013.
[8] DING Jian hua YAO Pei fu DONG Wei Yunnan Copper Industry Group. Technique and Implementation of Data Migration and Its on Storage Systems[J]. Computer Development & Applications, 2014.
[9] Eastlake Rd D, Jones P. US Secure Hash Algorithm 1 (SHA1)[M]. RFC Editor, 2001.
[10] Rivest R L. The MD5 message-digest algorithm. RFC 1321[J]. 1992.
[11] LengZhiJiang. The Research and Implementation of a Scal-out Cloud Storage Based on Distributed File System GlusterFS [D]. Fudan university, 2014.
[12] Schneider W A. Integral Formulation for Migration in Two and Three Dimensions[J]. Geophysics, 1978, 43(43):49.
[13] Wei, Chen, T.X. Verifying data migration correctness: The checksum principle[J]. 2014.
[14] Stone H S. High-performance computer architecture[J]. 1987.

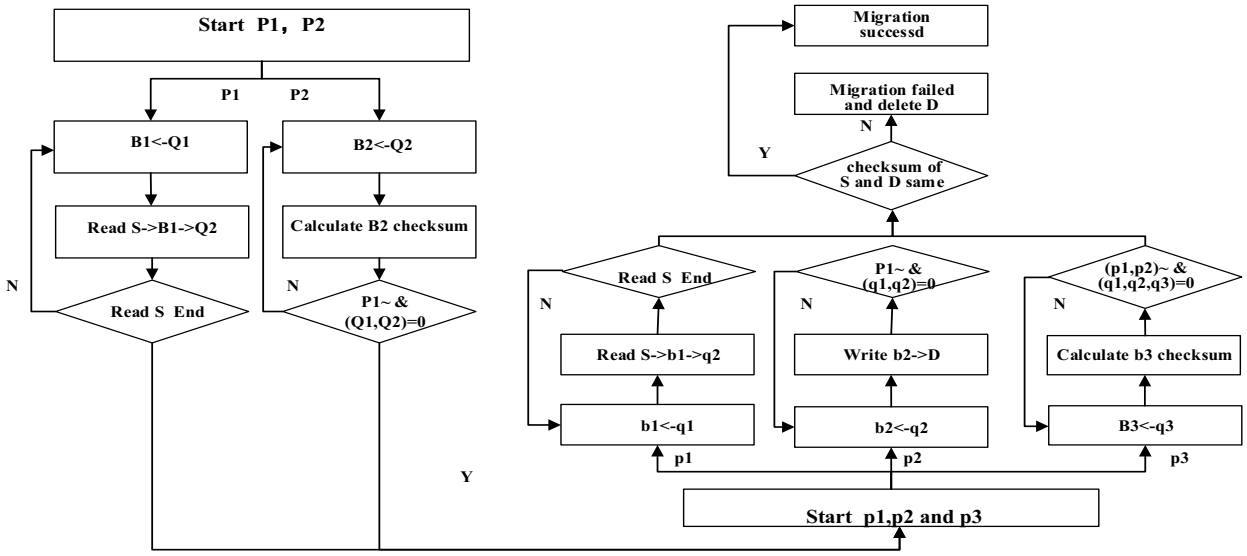

Figure 1. Flow chart of reliable migration method with pipelining