Applied Mathematics & Information Sciences
An International Journal

Experimental Study on Block and Ratio Storage Strategy and Parallel Transmission Algorithms

Chun-mao Jiang[1,2], *Guo-yin Zhang*[1] *and Ming-cheng Qu*[3]

[1]School of Computer Science and Technology, Harbin Engineering University, Harbin, Heilongjiang 150001, P.R.China
[2]School of Computer Science Technology and Information Engineering, Harbin Normal University, Harbin, Heilongjiang 150001, P.R.China
[3]School of Computer Science and Technology, Harbin Institutue of Technology, Harbin, Heilongjiang 150001, P.R. China

Received: 22 Sep. 2012, Revised: 27 Nov. 2012, Accepted: 16 Dec. 2012
Published online: 1 May 2013

Abstract: Parallel transmission algorithms CLBA and DAS greatly improved the speed of transferring data files, but they cause a large overhead of storage space and network traffic for deployment of duplications of large files, in addition to their poor feasibility of deploying large complete replicas on nodes with limited storage space. In view of Google and Hadoop distributed file systems all adopted block storage mechanism, a parallel data transmission algorithm based on block storage strategy is proposed, but the interrelation between block storage and parallel transmission was not properly taken into consideration and so the nodes of slower bandwidth greatly affected the overall transmission performance. Therefore, we proposed in this paper block and ratio storage strategies for files of massive data in distributed systems and parallel transmission algorithms PTBM and PTRM for the two storage strategies. Experimental results indicate the proposed algorithms are better than CLBA, and close to DAS, and they save more than 50% storage space, and can adapt to the deployment of large data files.

Keywords: Distributed system, large data file, data block storage, parallel transmission

1 Introduction

GridFTP is a sub-project of Globus which expands the FTP protocol, and one of its important functions is to release one data file on multiple nodes so that the file can be partially transferred in parallel from multiple servers [1,2], the studies on applications based on parallel transmission and partial file access greatly improved the speed of data acquisition[3,4].

(1)Parallel Transmission Algorithms Based on Multiple Complete Replicas

Sudharshan and others[5,6] proposed in 2003 the Co-Allocation strategies which use partial file access function of GridFTP to transfer many parts of the whole data at the same time from multiple nodes who have the copies of the file and then merge these parts. As shown in Figure 1.1, the co-allocator is the core unit that allocates downloading data for each download thread. Sudharshan proposed the following 4 strategies: Brute-Force Co-Allocation, History-based Co-Allocation, Conservative Load Balancing and Aggressive Load Balancing.

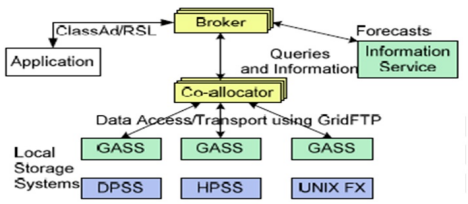


Figure 1.1 Co-allocation architecture

Brute-Force Co-Allocation (shown in Figure 1.2) allocates the same size of data block for each transmission channel; History-based Co-Allocation (shown in Figure 1.3) allocates the data size for each transmission channel proportionately on the basis of previous average

* Corresponding author e-mail: hsdrose@126.com

transmission speed of server nodes; Conservative Load Balancing (shown in Figure 1.4) divides the original duplication of the file into the blocks of equal size, each transmission channel is allocated to one data block, after it completes the current data block transmission, it applies for the next data block; Aggressive Load Balancing allocates more download data blocks for faster duplication nodes each time instead of only one block while it allocates data blocks. It is found through experiments that the performance of Aggressive Load Balancing is better than Conservative Load Balancing mainly due to its larger data block, and the performance of History-based Co-Allocation is poorer, and the performance of Brute-Force Co-Allocation is the poorest.

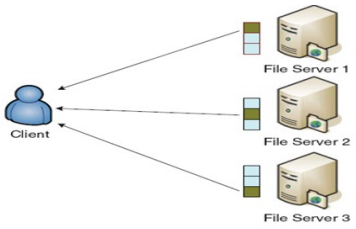


Figure 1.2 Brute-Force Co-Allocation

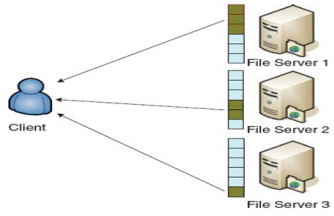


Figure 1.3 History-based Co-Allocation

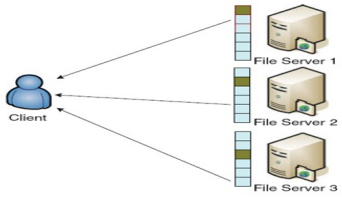


Figure 1.4 Conservative Load Balancing

Feng and others [7] also developed in 2004 a rFTP system based on partial data transfer function of GridFTP, and the essential characteristics of this system are the same as those of Sudharshan's thinking of cooperative transmission from multiple duplicates. The 5 cooperative transmission strategies in the system are Baseline, Static, Dynamic, SelfObserve and NoObserve.

Feng's Baseline and Sudharshan's Brute-Force Co-Allocation are exactly the same; his Static and Sudharshan's History-based Co-Allocation work to a similar principle, History-based Co-Allocation allocates download data based on the previous speed of each duplicate node while Feng's Static allocates download tasks based on the predicted speed of each duplicate node; Feng's Dynamic gets a file block U of fixed size every time, then allocates U proportionately on the basis of the predicted speed of each duplicate node, any task that finishes downloading will get a data block U1 of the same size as U from the unallocated remaining data file, and it will be transferred like U. SelfObserve is similar to Dynamic while SelfObserve allocates U based on the real speed of each duplicate node rather than the predicted speed; Feng's NoObserve is completely the same as Sudharshan's Conservative Load Balancing.

It can be seen from Feng's experiment that the performance of Dynamic, SelfObserve and NoObserve is very close to each other. The big problem Feng encountered is that when he wanted to deploy larger duplicates he had to observe further experimental results because the file system doesn't support large files in his experiment. Chao and others [8] came to the conclusion in 2005 that the final transmission completion time is subject to the slowest node after the last scheduling because allocating is based on the blocks of fixed size. So they proposed a recursive and dynamic data distribution algorithm, let the data file size be U, given the scale factor a(0¡a¡1), the first scheduling removes file fragment (section 1) of size U*a from U, the second scheduling removes file fragment (section 2) of size U*a*(1-a). Section 1 will be allocated in proportion based on the predicted speed of each node, when a download thread finishes its downloading (shown in Figure 1.5 as t1), then the allocation of section 2 begins with the unfinished parts on each node in section 1 taken into account. The same goes recursively until the remaining data that's not downloaded is less than a threshold value (threshold) which will be directly allocated (not multiplied by scale factor a).

Chao's method has a certain similarity to Feng's Dynamic except that Feng's study didn't specify how to deal with data that didn't download of each thread when current block U was allocated. Chao's study took this into account and gave the calculation formula, and the key of the formula is that the remaining data of each thread plus newly allocated data divided by the predicted speed of each thread must be equal, so that each thread finishes its transmission at the same time when the next scheduling begins shown t1 and t2 in Figure 1.5.

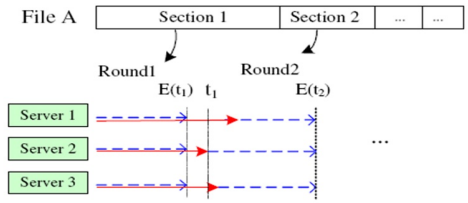


Figure 1.5 Process of Adjustment

It can be seen through comparison between Chao's experimental results and the result obtained using Sudharshan's three methods that Chao's algorithm is slightly better than Sudharshan's Conservative Load Balancing, but there is one precondition for Chao's experiments that the size of data block must be large, and his conclusion of the experiments also pointed out that the threshold value that's allocated at the last time played a key role, and so it's not certain that the performance of Sudharshan's Conservative Load Balancing is poorer than that of Chao's algorithm when the size of data block is smaller.

Bhuvaneswaran and others [9,10] also analyzed in 2005 and 2006 the impact of the last transmission block on the overall completion time in Sudharshan's Conservative Load Balancing and Aggressive Load Balancing, and he also took the impact of the uncertainty of network on the scheduling allocation into account. He gave a circular queue allocation method, and some blocks will be allocated multiple duplicate nodes in this case, so his algorithm has the ability to guard against failure. That is to say, if any duplicate node stops transmission, data block can be retransferred because other duplicate nodes also have the data block. And because each node has the full duplicate, no matter which node has a problem, data can still be transferred from other nodes. It can be seen from the description of the algorithm that a new block is allocated after a duplicate node finishes its current block downloading, this is similar to Sudharshan's Conservative Load Balancing.

Chao and others [11,12] further enriched and improved in 2007 and 2010 their algorithm based on the ideas they had proposed before, and developed a parallel transmission system. They took into consideration CPU, IO and network load of duplicate nodes, and gave a duplicate node selection model, but this didn't improve the performance of their previous parallel transmission algorithm.

(2)Review of the Study Above

The core idea of the study above is to make sure each thread finishes its download tasks at the same time through rationally allocating download tasks for each transmission thread. No matter it's based on fixed size block or dynamic size block, they all tried to shorten the completion time of the last block to optimize the overall time because the performance of the whole transmission process is basically the same except the last block. The study above makes full use of the advantage of parallel transmission to guarantee the transmission speed, but they all have some shortages. (1) It takes large storage space and network traffic overhead to deploy the full duplicate of the original file on multiple nodes to improve the collected bandwidth to support parallel transmission; (2) When the data file is too large, it's impossible to deploy the full duplicate of the file on a node with limited storage space of the node. The problem Feng met was the size of the files used by others was no larger than 4G.

(3)Block Storage and Parallel Transmission of Data Files

Since it's difficult to deploy the full duplicate when file is large, so people working in distributed file system research field proposed the block storage mechanism. For example: Google designed and implemented Google File System (GFS) to meet the increasing demand for data processing. The concept "file size" of GFS is different from that of a common file system. The file size of a common file system is usually counted by G bytes while the files of GFS are cut into fixes size Chunk and then deployed on different nodes.

Apache Foundation implemented Hadoop Distributed File System (HDFS), which is similar to GFS. It uses block storage strategy with high ability of fault-toleration and it's designed to be deployed on low-cost hardware. The parallel transmission studies above were all based on multiple full duplicates. In view of the difficulties in deploying duplicates of large files, Ruay-Shiung and others [16] proposed parallel transmission algorithm based on block storage strategy. It first assumes that the blocks of data file are randomly stored on different nodes of a distributed file system, a scheduling algorithm is given after the file information is obtained. But this algorithm is based on random block duplicate storage (it doesn't take network load into account), so it will meet the problem that the overall transmission time is too long because the faster node will wait for the slower node.

Similarly, Gaurav and others [16,17] established in 2008 a transmission optimization model based on the parallel transmission mechanism used for "Multi-source-Multi-target" data access situation. This model tried to minimize the overall completion time. In his research "Multi-source" doesn't mean the same data duplicate but different blocks of the same file. However, this research had the same shortages as Ruay-Shiung, it didn't take into account the process of data block storage (position, amount and so on), so it had the same problem that the overall transmission time is too long because the faster node will wait for the slower node[18].

(4) Problems and Main Contributions of This Paper

Problems: Parallel transmission based on multiple full duplicates has such shortages that on one hand it takes large storage space and network traffic overhead to deploy multiple full duplicates, on the other hand it's difficult to deploy the full duplicate when data file is too

large because of the limited storage space of a node. Although block storage solved the problems associated with deploying large files. can't avoid The impact of the slower nodes on the whole transmission time can't be avoided using download algorithm for data block random storage because it takes into account the process of data block storage and scheduling algorithm for later parallel access.

Contributions: In order to solve the problem above, a data block and fair storage strategy is proposed in this paper and a parallel transmission algorithm is given for this storage strategy. This storage strategy has a great advantage in space over the strategy based on full duplicate, and the parallel transmission algorithm can not only adapt to the dynamic change of network, but also it optimizes the transmission time of the last data block. Compared to Sudharshan's Conservative Load Balancing and Chao's algorithm, the performance of this paper's algorithm is better than Sudharshan's Conservative Load Balancing and close to Chao's algorithm in performance, but strategy proposed in this paper guarantees the parallel transmission speed, while it solves problems of deploying large data file and optimizing storage space and network traffic during the deploying process.

2 Data Block and Storage Strategy

Let data file be f with size S, k nodes fit for storing file $N_1 \sim N_k$.

(1)File Block

First divide the file into k equal parts, each part is denoted as $f_i(1 \le i \le k)$, the size of f_i is denoted as S_i, obviously $S_i = S/k$. Then divide f_i into $k-1$ equal parts, each part is denoted as $f_{ij}(1 \le i \le k, 1 \le j \le k-1)$, and the size of f_{ij} is denoted as $S_{ij}(1 \le i \le k, 1 \le j \le k-1)$. The total number of data blocks of file f is $k(k-1)$ as shown in Figure 2.1

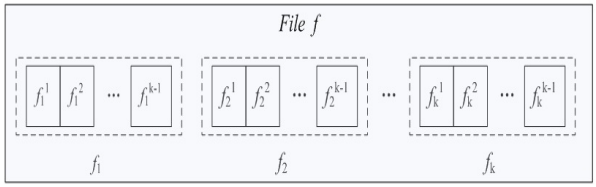


Figure 2.1 Diagram of File Block

(2)File Storage

There are two steps:(a)store f_i on node N_i, (b)Store $k-1$ files $f_{ij}(1 \le i \le k, 1 \le j \le k-1)$ of f_i on other $k-1$ nodes, one data block is stored on each node only. The storage is based on the number of nodes and number j of f_{ij} one by one in an increasing order (as shown in Figure 2.2). Other $f_i(1 \le i \le k)$ and its sub-files $f_{ij}(1 \le i \le k, 1 \le j \le k-1)$ are all processed in the same way. And the data storage structure is shown in Figure 2.2.We assume $k > 3$.

As shown in Figure 2.2 there are two types of data on each node. The blocks in the lower position of each node that's made bold are all from step (a), we call them LND data of the node (LND data of node i is donated as LNDi); The blocks in higher position of each node are all from step (b), we call them OND data of the node (OND data of node i is donated as ONDi).

Theory 1 It's obvious that LND data and OND data is equal in amount, so the total amount of storage is 2S. And it can be inferred from block storage mechanism that when any node can't be used, the union of the storage data of other nodes is equal to the complete data.

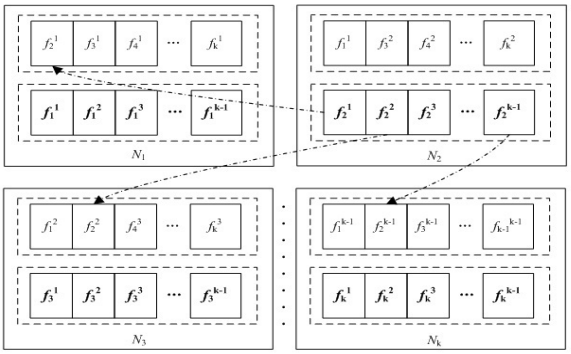


Figure 2.2 Diagram of Data Block Storage

3 Scheduler

3.1 Basic Ideas

Definition 1 (Local Data Surplus, S_i), let the average bandwidth of node N_i be V_i, the difference between the ideal data amount that's downloaded from server node N_i and local data LNDi in a parallel data acquiring is defined as local data surplus S_i, and

$$S_i = V_i/(\sum_{j=0}^{k-1} V_j) - LND_i$$

This indicator reflects the load of download tasks of each node, when S_i tends to or is equal to 0, it means each node can generally finish downloading at the same time. If S_i is positive, it means the download speed of the node is faster, it can share download tasks of nodes whose S_i is negative.

Example (1): use four nodes to store block data k=4, average bandwidths of each node are V1=12, V2=10,

V3=4, V4=28. The goal of scheduling algorithm is to make sure each node finishes download tasks at the same time without duplicate data. We give a scheduling result of the algorithm as shown in Table 1(a and b). When downloading data blocks in gray from node Ni, we can calculate that the ideal download data blocks of each block are 8, 6.7, 2.7 and 18.7, the actual scheduling result is 8, 7, 3 and 18. The load of each node before scheduling is (-1, -2, -6, 10), it turns to be (0, -0.3, -0.3, 0.7) after scheduling, the load has been basically balanced.

Because the number of blocks is an integer, it must be rounded off.

Table 1 Example of Block Storage and Scheduling

(a)

Node	***Speed***	S_i	***Ideal A***	***Actual B***	***A-B***
N_1	12	-1	8	8	0
N_2	10	-2	6.7	7	-0.3
N_3	4	-6	2.7	3	-0.3
N_4	28	10	18.7	18	0.7

(b)

LND Data of Node	*OND Data of Node*
(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)	(12)**(19)(20)**(21)(28)(29)(30)
(10)(11)(12)(13)(14)(15)(16)(17)(18)	(1)(2)(3)**(22)**(23)(24)(31)(32)(33)
(19)(20)**(21)**(22)**(23)(24)**(25)(26)(27)	(4)(5)(6)(13)(14)(15)(34)(35)(36)
(28)(29)(30)(31)(32)(33)(34)(35)(36)	**(7)(8)(9)(16)(17)(18)(25)(26)(27)**

It can be seen from Table 1 and Table 2 that the faster node shares the download task of slower node from OND data when it finishes its LND download task. For example, data blocks (25)(26)(27) are in LND of node N_3, but they are downloaded from OND data of node N_4.

3.2 Description of Scheduling Algorithm

Goal: Optimize Si of each node to make it close or tend to 0 to balance the load of each node.

Description: For any two nodes ZN and FN, if S_i of ZN is positive and LND data of FN is not shared by OND data of ZN, then ZN can share download tasks of FN. Every time ZN shares a data block, ZN.S_i–, FN.S_i++. If a data block is shared, then its download position will be marked (including the same data block on the other nodes), and the same data block on the other nodes won't be processed.

Input: k, p, V_i that meet the demand of the storage model.

Output: The scheduling schemes for download data blocks of each node.

(1) Construct linked list ZS_i_list with nodes whose S_i is positive, put ZSi_list in descending order.
(2) Construct linked list FS_i_list with nodes whose S_i is negative, put FS_i_list in ascending order.
(3) For every node FN in FS_i_list deal from first one by one.
(4) For every node ZN in ZS_i_list deal from first one by one // Let nods in ZS_i_list share download task of nodes in FS_i_list.
(5) IF FN.$S_i \geq 0$ THEN FSi_list.Move_to_Next, go to(3).
(6) IF ZN.$S_i > 0$ THEN Let OND data of current node ZN share LND data of FN, update ZN.S_i according to the data shared, each time one data block is shared, the following two conditions are always judged:
(a) IF FN.$S_i \geq 0$ THEN FS_i_list.Move_to_Next, go to (3).
(b) IF ZN.$S_i \leq 0$ THEN ZS_i_list.Move_to_Next, go to (4).
(7) In the steps (1) - (6) above, let the faster node share the download tasks of the slower node, as long as one of the two lists gets to the end, it jumps out.
(8) If there still are nodes FN with S_i negative in FS_i_list, then traverse ZS_i_list from first, if the unprocessed OND data blocks of node ZN in ZS_i_list and the unprocessed LND data blocks of FN intersect, then let ZN share download tasks of FN until they don't intersect regardless of whether the S_i of ZN is negative.
(9) At this time there are two types of nodes in FS_i_list: Si is positive or non-positive. Let OND data of nodes with S_i positive share LND data of nodes with S_i negative until it can't be shared.
(10) At this time there are two types of nodes in ZS_i_list: S_i is positive or non-positive. Repeat steps (1) - (6) for nodes in ZS_i_list. Then download based on the download marks of data blocks on each node after it's done.

4 Storage Strategy and Parallel Transmission Algorithm

Because scheduler only has functions that are similar to "History-based Co-Allocation", it schedules only once for every transmission, it has to schedule many times to adjust the data blocks that are downloaded from each node owing to the dynamic change of network transmission speed.

Definition 2(Space Occupied Ratio, SAV) the ratio between the storage space it takes to store data on i nodes based on this paper's storage strategy and the storage space it takes to store data on i nodes using full duplicate is defined as space occupied ratio (SAV).

4.1 Block Storage Strategy Model

If node k can be used to store data, then divide the original data into k equal parts, each block is denoted as B_i, i∈(1,k). Each block uses the storage strategy given in Section 2. An example of k=4 is given in Figure 4.1

The following conditions are true in Figure 4.1: size(B_1)=size(B_2)=size(B_3)=size(B_4),

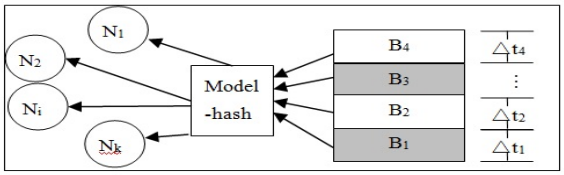


Figure 4.1 Example of Storage Strategy

Table 2 Example of Segmentation Process of Block Storage Strategy Model

B_i	B_1	B_2	B_3	B_4
Segmentation	↓	↓	↓	↓
Data	(1)(2)...(11)(12)	(13)(14)...(23)(24)	(25)(26)...(35)(36)	(37)(38)...(47)(48)

file=$B_1 \cup B_2 \cup B_3 \cup B_4$. Specific segmentation process is as shown in Table 2.

We can get the data allocation shown in Table 3 further based on the mechanism of block storage model.

Table 3 Example of Data Allocation of Block Storage Strategy Model

Node / B_i	N_1	N_2	N_3	N_4
B_1	(1)(2)(3)(4)(7)(10)	(4)(5)(6)(1)(8)(11)	(7)(8)(9)(1)(5)(12)	(10)(11)(12)(3)(6)(9)
B_2	(13)(14)(15)(16)(19)(22)	(16)(17)(18)(13)(20)(23)	(19)(20)(21)(17)(14)(24)	(22)(23)(24)(18)(15)(21)
B_3	(25)(26)(27)(28)(31)(34)	(28)(29)(30)(25)(32)(35)	(31)(32)(33)(26)(29)(36)	(34)(35)(36)(27)(30)(33)
B_4	(37)(38)(39)(40)(43)(46)	(40)(41)(42)(37)(44)(47)	(43)(44)(45)(38)(41)(48)	(46)(47)(48)(39)(42)(45)

4.2 Scale Model Storage Strategy

4.2.1 Storage Mechanism

In the section above, the whole data is divided into i sub-blocks B_i, then store the i sub-blocks based on the model. In this section, we will give another storage and segmentation strategy as shown in Figure 4.2, divide the complete data file into two parts, δ and $1-\delta$, $0<\delta<1$.

Figure 4.2 File Ratio Segmentation

If i node can be used to distribute file, the storage and segmentation process of the two parts above is as followed:

(1) Part δ is divided into i equal parts, then every sub-file block is stored on node i;

(2) Part $1-\delta$ is stored based on block storage model given in the above section.

The whole file storage strategy is shown as Figure 4.3.

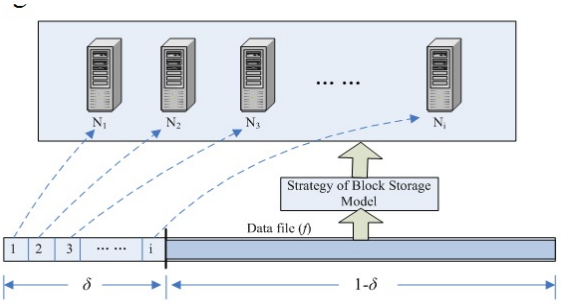


Figure 4.3 Example of File Scale Model Storage Strategy

If we use block model storage strategy in Section 4.1, then its space occupied ratio SAV with full data duplicate deploying is 2/k. If we use scale model storage strategy, the formula to calculate space occupied ratio SAV is:

$$SAV = \frac{a+2(1-a)}{k} = \frac{2-a}{k} \quad (0<a<1) \qquad (1)$$

It can be seen from the diagram that when δ equals 0, scale model storage strategy will change to block model storage strategy, and the corresponding space occupied ratio SAV will have the same trend as shown in formula (1).

4.2.2 Relationship between Average Bandwidth of Nodes and δ

It can be seen from 4.2.1 that part δ of the file is divided into i parts and then put on i nodes, so every data block will be downloaded if we want to get the complete data. The goal of parallel transmission is to make sure each node finishes download tasks at the same time, so for the slower nodes they must finish data blocks δ stored on them while for the faster nodes will try their best to transfer all the file blocks on them to the demanders. Theorem 1 Assuming each node finishes download tasks at the same time, let the maximum average bandwidth of nodes in a transmission be V_{max}, let the minor be V_{min}, then

V_{max}, V_{min} and δ must satisfy $\frac{V_{max}}{V_{min}} \leq \frac{2}{\delta} - 1$

Prove: Because each node finishes download tasks at the same time and data integrity must be guaranteed, the

fastest nodes must finish transferring all the data stored on them. Their total data amount $D_{\max} = \left(\frac{\delta}{i} + \frac{2(1-\delta)}{i}\right) \cdot M$ based on block model storage strategy and scale model storage strategy; the slowest nodes must finish transferring all the data of δ parts stored on them, that is $D_{min} = \frac{\delta}{i}M$. The transmission time is equal, so

$$\frac{V_{max}}{V_{min}} = \frac{D_{max}}{D_{min}} = \left(\frac{\delta}{i} + \frac{2(1-\delta)}{i}\right) \cdot M \Big/ \frac{\delta}{i}M = \frac{2}{\delta} - 1$$

If $\frac{V_{max}}{V_{min}} > \frac{2}{\delta} - 1$, then the bandwidth of slowest nodes will be less than Vmin or the bandwidth of the fastest nodes will be greater than Vmax, in this situation, none of them can finish data transmission at the same time, this contradicts the precondition of the theorem, so in order to make sure each node finishes download tasks at the same time, we must have $\frac{V_{max}}{V_{min}} \leq \frac{2}{\delta} - 1$.

4.3 Principle of Parallel Transmission Algorithm

Scheduling algorithm in this section is based on block model storage strategy, the process of δ part of file in scale model storage strategy is given in section 5.2.

Take Diagram and Table for example, because any data block B_i that satisfies the storage model is not full duplicate, if download B_i from four nodes at the same time, it must satisfy: (1) Download data from the four nodes continuously; (2) Download without repeated data blocks; (3) The union of downloaded data of each node after finishing downloading is equal to B_i.

Because transmission speeds of nodes dynamically change, let Δt_i be the interval of time when the first two nodes finish downloading B_{i-1} and B_i. For example, N_1 first finishes downloading B_{i-1} and the time is t_{i-1}, N_3 first finishes downloading B_i and the time is t_i, then $\Delta t_i = t_i - t_{i-1}$.

The data amount of Bi that each node is allocated is determined as follows: (1) Let V_i, the speeds of each node at the time $t_{i-1}(0 \leq i \leq k-1)$ be the input of scheduler; (2) Let the unfinished data amount of each node at the time t_{i-1} be m_i; (3) Let the data amount of B_i that each node is allocated be $x_i(0 \leq i \leq k-1)$; (4) They theoretically satisfy Formula 2 to make sure each node finishes download tasks at the same time.

$$\frac{V_0}{m_0 + x_0} = \frac{V_1}{m_1 + x_1} = \ldots = \frac{V_{k-1}}{m_{k-1} + x_{k-1}} \qquad (2)$$

Apply Equation Identically Equal Theorem on Formula 2, we deduce Formula 3, and from Formula 3 we deduce the method of calculating x_i shown as Formula 4. Let x_i be the input values of scheduler S_i.

$$\left(\frac{V_i}{m_i + x_i}\right)_{i=0}^{k-1} = \frac{\sum_{i=0}^{k-1} V_i}{\sum_{i=0}^{k-1}(m_i + x_i)} = \frac{\sum_{i=0}^{k-1} V_i}{\sum_{i=0}^{k-1} m_i + \sum_{i=0}^{k-1} x_i} = \frac{\sum_{i=0}^{k-1} V_i}{\sum_{i=0}^{k-1} m_i + M} = \lambda \qquad (3)$$

$$x_i = \frac{V_i}{\lambda} - m_i \ (0 \leq i \leq k-1) \qquad (4)$$

Based on the formulas above, we give the schematic diagram shown in Figure 4.1. The client is made up of download unit and scheduler unit, the download unit passes recent average speed V_i and unfinished data amount mi of each node to scheduler when a Δt ends, scheduler works out x_i, then calculate and output data blocks of B_i that should be downloaded from each node, and pass them to download unit. Download unit downloads data blocks from each node based on the output of scheduler unit, it also observes whether a new Δt is generated all the time along this processing loops.

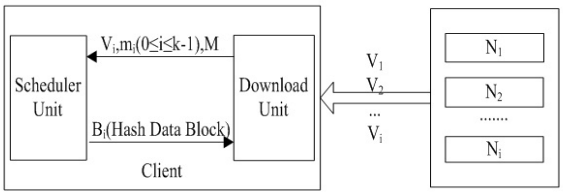


Figure 4.4 Schematic Diagram of Parallel Transmission

5 Experimental Environments

5.1 Constitution of Functions

As shown in Figure 5.1, the client of a prototype system mainly has the following 6 functional modules: data partition unit, data upload unit, storage unit, query unit and data download unit.

Model Process Unit Divide the original data according to the number of nodes (k) used to store and the number of blocks (h) of block storage based on the storage model and the block storage strategy. A prototype system uses all kinds of files as the original data, uses an open-source file segmentation unit to divide file, and it can control the size of B_1 when dividing.

Data Upload Unit Based on the IP address of servers, upload the segmentation data in batches according to the hash rules of model, and write the data and server storage information into database through the storage unit, which can be used to query during downloading.

Data Download Unit When download data, first let the query unit queries the database according to file name

(unique), get all the storage information and block information of the data. Then call the thread control unit to create threads based on the number of storage nodes, transfer data parallel with scheduler unit and parallel transmission algorithm unit to finish downloading data.

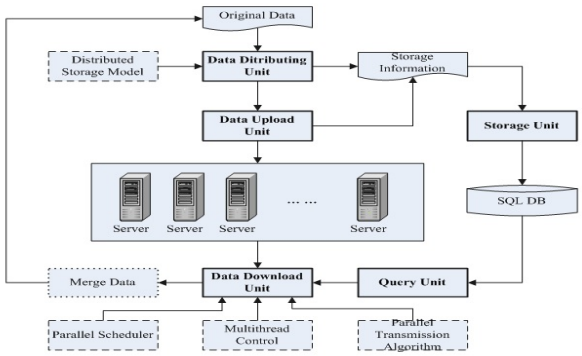


Figure 5.1 Constitution and Procedure of Prototype System

Because the original data blocks are effectively divided, all the blocks can be merged correctly after they are all downloaded.

5.2 Determination of Multithread Transmission Parameters

Determination of Average Download Speed of Each Thread At first, the master threads read out the storage list of all the file, create threads based on the value of k , and the default download speed of each node is the same, that is V1:V2:...Vk=1:1:...:1, this is the input of scheduler (11), so the download data amount of each node must be equal at the first scheduling. For the δ part data in scale model storage strategy, it just needs to download data blocks that have been divided equally. How to calculate continuous V_i? Since every data block file in the downloaded list of each thread contains the time when downloading is finished, use the time when the most recent downloading is finished to subtract the previous one shown as the coordinate axis in the right of Figure 5.2, we can get Δt, which is the time it takes to transfer data file x+1. Then we can work out the average speed $\overline{V_i}$ at which a specific thread downloads from a specific node because the size of each data block file has already been known in the stage of first segmentation, we can use this speed as the input speed of scheduler unit.

Determination of Scheduling Opportunity That is when to dispatch a new download task to threads. When a download list of any thread is null, the master thread gets out the next grouped data $B_i + 1$ based on the position of the dispatched grouped data B_i to begin scheduling.

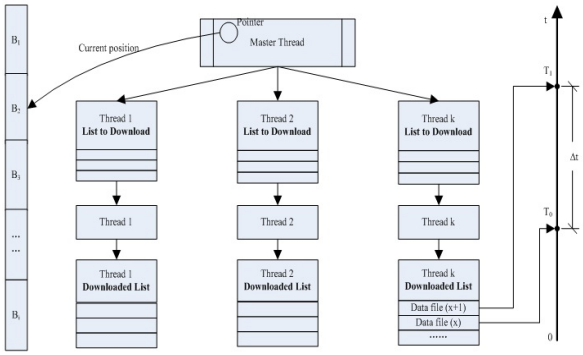


Figure 5.2 Schematic Diagram of Multithread Transmission

Determination of Unfinished Data Amount of Each Thread Use the number of surplus files in the list to download each thread as the mi of scheduler unit (Figure 4.4).

6 Performance Test

6.1 Introduction to Experiment

Since parallel transmission of this paper can be applied to block model storage strategy and scale model storage strategy, let parallel transmission algorithm based on block model storage strategy be PTBM(parallel transmission based on block model), let parallel transmission algorithm based on ratio model storage strategy be PTRM(parallel transmission based on ratio model), let conservative load balancing algorithm be CLBA, let Chao's algorithm be DAS(dynamic adjustment strategy). As for ratio model storage strategy, it changes to block model storage strategy when δ is 0. This experiment first verifies the impact of B_i on block model storage strategy; then detect the performance difference of ratio model storage strategy in certain network status.

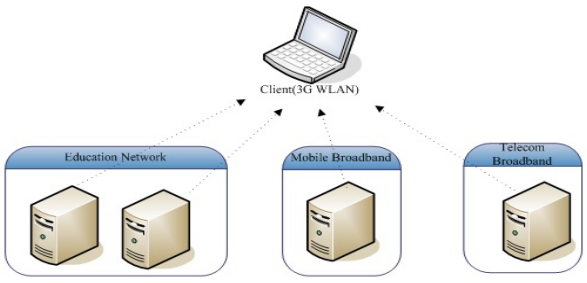


Figure 6.1 Network Environment of Experiment

We choose 3 kinds of network, education network, Mobile Broadband and Telecom Broadband, select a certain number of nodes from the 3 kinds of network as the data deployment nodes of the model as detailed below: Data demand node is connected through Telecom 3G WLAN; select two computers in education network, one computer in Mobile Broadband and one computer in Telecom Broadband as data deployment nodes, install IIS server on them, and the size of the data file is 960M. Use the same network connection and service nodes to guarantee the physical conditions are the same as shown in Figure 6.1.

Definition 2-23 (MLTT: minimum limited transmission time) If the total data amount is M, the number of nodes is k, the real speed of node i is V_i, the minimum limited transmission time is ζ, then $M = \sum_{i=1}^{k} \left(\int_0^{\zeta} V_i dv \right)$.

Time ζ is determined as follows: Record data amount mi that each node transfers (take no account of scheduling and merging file) when transfer data from duplicate node to client. Stop when $\sum m_i = M$, then we can get ζ.

Conclusion: The minimum transmission time of each algorithm can't surpass MLTT.

Experiment(A) Test performance differences between this paper's PTBM algorithm and CLBA algorithm, DAS algorithm under the condition of equal number of nodes.

Experiment(B) Test the impact of the size of B_i on the performance of PTBM algorithm.

Experiment(C) Test performance differences between PTBM algorithm and CLBA algorithm, DAS algorithm in the same network conditions when δ changes and in the different network conditions when δ is fixed.

6.2 Experiment A: PTBM Performance Comparison

Use two PCs in education network, one PC in Mobile Broadband and one PC in Telecom Broadband. Let $B_i = 40$, data file was divided into 480 blocks after being processed by the model. Every data block was approximately 2M. For Conservative Parallel Transmission, it's also divided into 480 blocks, and the partition method is the same. Observe the value of MLTT to compare the performance with other two algorithms. The test durations are 9-10(AM), 2-3(PM) and 5-6(PM), observe the time value of PTBM,CLBA,DAS and MLTT in each duration. Observe 3 times for every algorithm experiment in every duration because network conditions change in different periods of time. The results are shown as Table 4(a and b).

Process the data in the table, we can get Figure 6.2, it can be seen from the table that there is little difference between PTBM and CLBA observed at the same time, DAS algorithm always has some slight advantage over the others. Because the network is not stable, in the first and third observation, transmission time of CLBA is slightly less than that of PTBM while in the second observation, transmission time of PTBM is slightly less than that of CLBA.

Table 4 Transmission Time

(a)

Item	PTBM			CLBA		
Count	1	2	3	1	2	3
Time(Sec.)	2306	2226	1765	2250	2259	1666

(b)

Item	DAS			MLTT		
Count	1	2	3	1	2	3
Time(Sec.)	2206	2189	1658	2180	2152	1580

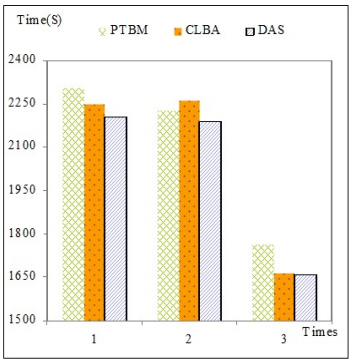


Figure 6.2 Comparison of Transmission Time

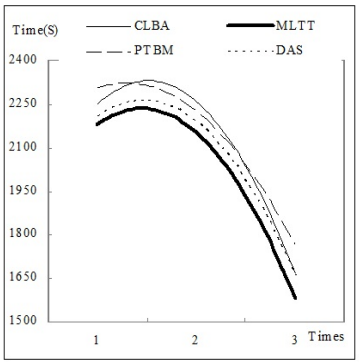


Figure 6.3 Polynomial Curve Fitting of Transmission Time

Use a quintic polynomial to fit the curves to three kinds of transmission time to further observe trends and performance differences. As shown in Figure 6.3, the MLTT curve is at the bottom. PTBM curve is basically consistent with CLBA curve, their trends are the same, they cross with each other in the middle, which means their performances are close to each other. DAS curve is always at their bottom, it has some slight advantages. Three curves are all above the MLTT curve.

6.3 Experiment B: Analysis of Impact of B_i in PTBM

Because B_i has a direct impact on the size of a file, the previous dynamic schedule algorithm mainly optimizes the transmission time of the last data block, the experiment in this section is mainly concerned with the impact of B_i on the algorithm performance when its values are different, and test duration is 8:30-11:30 (AM).

Table 5 List of metadata and B_i

B_i	10	20	30	40	50	60	70
metadata(M)	8	4	2.7	2	1.6	1.33	1.14
Transmission Time	2456.4	2394.3	2348.3	2226.86	2247.1	2226.4	2456.4

Let B_i=10,20,30,40,50,60,70, take out a block for the processing every time based on the model and download with parallel transmission. The corresponding minimum file block size metadata when the value of Bi is different is as shown in Table 5. The transmission time of downloading the same data under all kinds of conditions is shown in the third line of Table 5.

It can be seen from Figure 6.4 that when B_i is 10, the performance is the poorest, it's the same as the scheduling based on history in this situation. With the increasing of B_i, transmission time gradually decreases. It can be seen from the fitting curve quartic polynomial in Figure 6.5 that the performance tends to be stable when B_i is 40, 50, 60 and 70. That is to say the impact of the size of data block on the algorithm performance tends to be stable to a certain degree. And it can be seen from experiment A that the algorithm performance is close to the conservative scheduling algorithm when B_i is 40.

6.4 Experiment C: PTRM Performance Comparison

The experiments were mainly made to test: (1) differences in performance between PTBM algorithm and CLBA algorithm, DAS algorithm in the different network conditions when δ is fixed; (2) differences in performance between PTBM algorithm and CLBA algorithm, DAS algorithm in the same network conditions when δ changes.

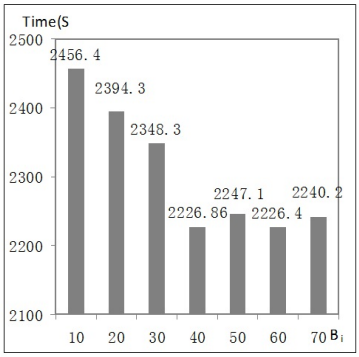


Figure 6.4 Comparison of Transmission Time

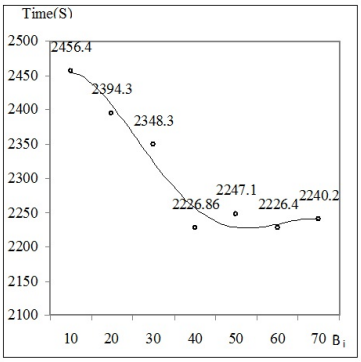


Figure 6.5 Polynomial Curve Fitting of Transmission Time

(1) Let $\delta = 0.3$, test durations are 9-10(AM), 2-3(PM) and 5-6(PM). Observe PTRM in the first day, CLBA in the second day and DAS in the third day. Observe 3 times for every algorithm experiment in every duration because network conditions change in different periods of time. The observed value of PTRM is shown in Table 6, use data in Table 4 as the result of CLBA and DAS.

Table 6 Observed Value of PTRM

(a)

Item	**PTRM($\delta = 0.3$)**			**CLBA**		
Count	1	2	3	1	2	3
Time(Sec.)	2243.88	2271.94	1752.6	2250.09	2259.75	1666.58

(b)

Item	**DAS**		
Count	1	2	3
Time(Sec.)	2206.85	2189.6	1658.3

We can get Figure 6.6 from the data of Table 6, it can be seen from figure 6.6 that the performances of the three algorithms are close to each other in the same duration every day without big difference, but DAS algorithm has some slight advantage over the others.

(2) Let δ=0.1, 0.2, 0.3, 0.4, 0.5, test time is 5 days, observe PTRM at 8-9(PM) every day when δ takes different values in order to ensure approximate consistent state of the network. The observed result is shown in Table 7.

Table 7 PTRM Performances for Different δ

δ	0.1(First Day)	0.2(Second Day)	0.3(Third Day)	0.4(Fourth Day)	0.5(Fifth Day)
Time(Sec.)	1989	1970	1981	2021	2040

We can get Figure 6.7 from the data of Table 7, it can be seen from Figure 6.7 that the performances are close to each other when δ=0.1,0.2,0.3. Experiment of 6.6.3-(1) shows the performance of PTRM is close to those of the other two algorithms when $\delta = 0.3$ while performance of PTRM decreases obviously when δ=0.4,0.5.

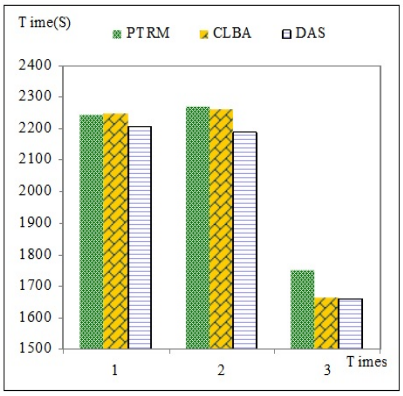


Figure 6.6 Comparison of Transmission Time

The maximum and minimum average bandwidths of each experiment are shown in Table 8. It can be seen from Table 8 that the actual V_{max} and V_{min} are 4.6 and 4.5 respectively when δ is 0.4 and 0.5, they are all greater than theoretical values 4 and 3. It can be seen from the observed results that the increasing of time is because the nodes with narrow average bandwidth delay the download time of part of file δ. So the nodes with wide average bandwidth have better performance when δ is smaller. For example, the actual V_{max}/V_{min} is 4.4 when δ=0.3, but the performance is still good.

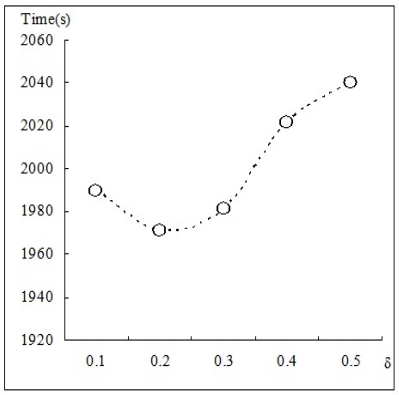


Figure 6.7 Transmission Time When δ Changes

Table 8 Ratio of V_{max} and V_{min}

δ	0.1	0.2	0.3	0.4	0.5
Theoretical V_{max}/V_{min}	19	9	5.7	4	3
Actual Average V_{max}**(kbyte/s)**	212	221	209.0	218	208
Actual Average V_{min}**(kbyte/s)**	44	43	48	47	46
Actual V_{max}/V_{min}	4.8	5.1	4.4	4.6	4.5

6.5 Summery of Experiments

Experiment (A) shows this paper's PTBM algorithm has a performance similar to those of CLBA and DAS, but it saves 50% storage space.

Experiment (B) shows B_i has an obvious influence on PTBM algorithm, but when size of metadata is less than 2M, the enlargement of Bi has no influence on the algorithm.

Experiment (C) shows the performance of this paper's PTRM algorithm is close to those of CLBA and DAS when $\delta < 0.3$ in the given network environment. At this time, V_{max}/V_{min} is 4.4, bandwidth differs greatly, which means PTRM can adapt to the poor network environment. It can save 57% storage space in the 4 node environment of this paper. (See section 6.6 for details)

6.6 Comparison of Storage Space (SAV)

We can work out space occupied ratio (SAV) in different experimental conditions based on Theory 1, Formula 1 and the definition of SAV shown in Table 9 and 10. As shown in Table 9, SAV is 0.425 when $\delta = 0.3$, and it can be seen from the experiment that the performance of algorithm is still good. As shown in Table 10, the corresponding SAV is 0.28 when k=6.

Table 9 Comparison of SAV When k=4

Item	PTBM	PTRM		
		$\delta = 0.1$	$\delta = 0.2$	$\delta = 0.3$
SAV	0.5	0.475	0.45	0.425

Table 10 SAV Comparison of PTRM When $\delta = 0.3$

k	3	4	5	6
SAV	0.57	0.425	0.34	0.28

7 Conclusion

Parallel transmission based on deploying multiple duplicate plays an important role in the practical use of distributed systems. Previous parallel transmission algorithm based on multiple full duplicates has made good progress. But complete multiple duplicate deployment has some shortages, on one hand it takes large storage space and network traffic overhead to deploy multiple full duplicates, on the other hand it's difficult to deploy large duplicate on nodes with limited storage space. So Google and Hadoop distributed systems completely adopt block storage strategy. Some others studied on parallel transmission algorithm are based on random block storage strategy, but it takes into account duplicate storage and parallel transmission, so that the nodes with narrow bandwidth will greatly delay the overall transmission time. This paper presents two storage strategies and parallel transmission algorithms, which effectively solve the problems of deploying large duplicate and large overhead of storage space. Compared to the algorithm based on full duplicates, PTBM and PTRM have better performance and save more than 50% storage space at the same time.

Acknowledgement

This work is partially supported by the National Science Foundation of China under Grant No.61073042 and Heilongjiang Province Science Foundation under Grant No.F201139. Thanks for the help.

References

[1] William Allcock, John Bresnahan, Rajkumar Kettimuthu etal. *The Globus Striped GridFTP Framework and Server*. Proceedings of the 2005 ACM/IEEE SC—05 Conference. November 12-18, 2005, Seattle, Washington, USA: 1-11.

[2] Jun Feng, Lingling Cui, Glenn Wasson. *Toward Seamless Grid Data Access: Design and Implementation of GridFTP on .NET*. Grid Computing Workshop, 2005. IEEE computer society: 164-171.

[3] Wantao Liu, Rajkumar Kettimuthu, Brian Tieman. *GridFTP GUI: An Easy and Efficient Way to Transfer Data in Grid*. GridNets 2009, LNICST **25**. 2010, 57-66.

[4] Branimir Radic, Vedran Kajic, and Emir Imamagic. *Optimization of Data Transfer for Grid Using GridFTP*. Journal of Computing and Information Technology, 2007, **4(15)**: 347-353.

[5] Sudharshan, Vazhkudai. *Enabling the Co-Allocation of Grid Data Transmissions*[A]. Proceedings of the Fourth International Workshop on Grid Computing[C], IEEE Computer Society, 2003: 1-8.

[6] Sudharshan Vazhkudai. *Distributed Downloads of Bulk, Replicated Grid Data*. Journal of Grid Computing, 2004, **2**: 31-42.

[7] Jun Feng, Marty Humphrey. *Eliminating Replica Selection - Using Multiple Replicas to Accelerate Data Transfer on Grids*. Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04), IEEE computer society: 1-8.

[8] Chao-Tung Yang, I-Hsien Yang, Kuan-Ching Li. *A Recursive-Adjustment Co-allocation Scheme in Data Grid Environments*. ICA3PP 2005, LNCS **3719**. 2005: 1-10.

[9] R.S. Bhuvaneswaran, Yoshiaki Katayama, and Naohisa Takahashi. *Coordinated Co-allocator Model for Data Grid in Multi-sender Environment*. ICSOC 2006, LNCS **4294**. 2006: 66-77.

[10] R.S. Bhuvaneswaran, Yoshiaki Katayama, and Naohisa Takahashi. *Dynamic Co-allocation Scheme for Parallel Data Transfer in Grid Environment*. Semantics, Knowledge and Grid, 2005. SKG'**05**. First International Conference on, IEEE, 2005: 1-6.

[11] Chao-Tung Yang, I-Hsien Yang and Kuan-Ching Li. *Improvements on dynamic adjustment mechanism in co-allocation data grid environments*. The Journal of Supercomputing. 2007, **40**: 269-280.

[12] Chao-Tung Yang, Shih-YuWang, William Cheng-Chung Chu. *Implementation of a dynamic adjustment strategy for parallel file transfer in co-allocation data grids*. J Supercompute, 2010, **54**: 180-205.

[13] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. *The Google File System*. SOSP'**03**, October 19-22, Bolton Landing, New York, USA, 2003: 1-15.

[14] Konstantin Shvachko, Hairong Kuang, Sanjay Radia et al. *The Hadoop Distributed File System*. IEEE 2010: 1-10.

[15] Shafer, J.; Rixner, S.; Cox, A.L. etal. *The Hadoop Distributed Filesystem: Balancing Portability and Performance*. 2010 IEEE: 122-133.

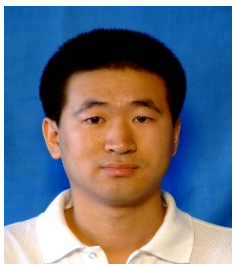
[16] Ruay-Shiung Chang, Po-Hung Chen. *Complete and fragmented replica selection and retrieval in Data Grids*. Future Generation Computer Systems, 2007, **23**: 536-546.

[17] Gaurav K, Umit C, Tahsin K, et al. *A Dynamic Scheduling Approach for Coordinated Wide-Area Data Transfers using GridFTP*. Parallel and distributed system., 2008, 1-12.

[18] Ji-Yi W, Jian-Lin Z, Tong W, et al. *Study on Redundant Strategies in Peer to Peer Cloud Storage Systems*[J]. Applied mathematics & information sciences. 2011, **5(2)**: 235-242.

JIANG Chun-Mao received the MS degree in software engineer from Harbin Institute of Technology in 2004. He is a Ph.D. candidate in school of computer science and technology of Harbin Engineering University. His research interests include embedded computing and P2P etc.

ZHANG Guo-Yin, born in 1962, is a professor and Ph.D. supervisor in school of computer science and technology of Harbin Engineering University. His research interests include embedded computing and P2P etc.



Qu Ming-Cheng, born in 1980, is a Ph.D. in school of computer science and technology of Harbin Institute of Technology. His research interests include embedded computing and P2P etc.