A Network Load Sensitive Block Placement Strategy of HDFS

Lingjun Meng[1], Wentao Zhao[1,2], Haohao Zhao[1] and Yang Ding[1]
[1]School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo,454000-China
[e-mail: menglingjun1109@126.com]
[2] Opening Project of Key Laboratory of Mine Information, Henan Polytechnic University, Jiaozuo,454000-China
[e-mail: zwt@hpu.edu.cn]
*Corresponding author: Wentao Zhao

Received November 14, 2014; revised July 12, 2015; accepted August 22, 2015; published September 30, 2015

Abstract

This paper investigates and analyzes the default block placement strategy of HDFS. HDFS is a typical representative distributed file system to stream vast amount of data effectively at high bandwidth to user applications. However, the default HDFS block placement policy assumes that all nodes in the cluster are homogeneous, and places blocks with a simple RoundRobin strategy without considering any nodes' resource characteristics, which decreases self-adaptability of the system. The primary contribution of this paper is the proposition of a network load sensitive block placement strategy. We have implemented our algorithm and justify it through extensive simulations and comparison with similar existing studies. The results indicate that our work not only performs much better in the data distribution but also improves write performance more significantly than the others.

Keywords: HDFS, block placement, network load, imbalance, load balance

A preliminary version of this paper appeared in KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS 2014, Oct. X. This version includes a concrete analysis and supporting implementation results on improved block placement strategy of HDFS. This research was supported by Provincial Key Technologies R & D Progaram of Henan province(142402210435).

http://dx.doi.org/10.3837/tiis.2015.09.014 **ISSN : 1976-7277**

1. Introduction

Arrival of big data brings unprecedented opportunities and challenges for data -intensive applications such as Internet services, cloud computing and social networking [1]. Traditional storage solutions fail to satisfy these applications due to the rapidly growing size of datasets ranging from a few gigabytes to several terabytes or even petabytes. Cloud computing is emerging as a powerful paradigm for dealing with big data[2], which is developed from distributed processing, parallel processing and grid computing and is deemed as the next generation of IT platforms that can deliver computing as a kind of utility. And traditional infrastructure has been superseded by cloud computing, due to its cost-effective and ubiquitous computing model [3]. Google adopts GFS [4] to access the ever-expanding datasets and leverages MapReduce model [5] for massive data parallel processing. Hadoop- a significant project hosted by Apache Software Foundation is an open-source implementation of the Google's MapReduce model developed by Yahoo and has been regarded as the first choice to deploy large-scale and data-intensive applications. It is mainly composed of HDFS (Hadoop Distributed File System) and MapReduce. HDFS is a typical robust and scalable distributed file system, which provides reliable storage and high throughput access to application data and a great deal of APIs for programmers. What's more, HDFS is highly fault-tolerant through data redundancy. It has had many successful applications so far, such as Yahoo!, Facebook, Last.fm, Amazon etc. All of these well-known enterprises deploy HDFS for data storage widely. MapReduce is a popular distributed processing model for applications generating big data since it can simplify the complexity of distributed data processing functions across multiple nodes in a cluster. Especially, it is fault tolerant and completely transparent to programmers.

HDFS plays a prominent role in Hadoop ecosystem as data locality is a key factor for the HDFS reliability and MapReduce performance. HDFS achieves high performance, reliability by data replication, so replica placement is critical to HDFS in which failure is normal instead of exceptional. HDFS take triplication measure to improve data locality in the event of node failures and achieve load balance by distributing each replica based on disk utilization [6]**.** The default block placement strategy is very useful in a homogeneous environment and does well in high fault tolerance, while HDFS is heterogeneous and complicated actually. Hence, some inherent problems with the default policy need to be analyzed and solved.

The default policy places replicas with the restriction that no more than one replica is placed at any one node and no more than two replicas are placed in the same rack when the number of replicas is less than twice the number of racks. It ignores the tremendous heterogeneities load existed among computing nodes in the cluster, such as network load, bandwidth, real-time state, CPU and memory etc. As a result, it may lead to load imbalance as well as decrease parallelism performance when the clients are concentrated on visiting the system. Some other serious phenomenon may occur, for instance, some nodes can be too fully occupied to respond clients' requests due to their capacity constraints or network congestions, while others may be idle very much, resulting in time-consuming or even nodes' breakdown easily. What's worse, node that dynamically join and leave may aggravate load imbalance in HDFS owing to data migration. The overload nodes can be rebalanced by the Balancer [7], which is a default load balance procedure provided by HDFS. But it's hysteretic obviously since it need to be operated manually by the administrator when the unbalance state arrives at the threshold

(default value is 10%).Worse, data migration caused by Balancer may occupy too much resources, including bandwidth, memory etc.

The rest of the work on replica placement is organized as follows. In section 2, we briefly summarize the latest work related to HDFS placement strategy. Section 3 describes the default block placement method and the default access to rebalance the cluster. Section 4 is dedicated to the proposed block placement principle and system model with mathematical formulation. While in section 5 we focus on the experiment results and analysis. Finally, section 6 concludes the work.

2. Related Work

According to the significant deficiency discussed above, how to place blocks effectively to the distributed file system has become a hot topic. And a better placement strategy can perform much better in the event of nodes' load imbalance, network congestion bandwidth constraints in the cluster.

In order to overcome these drawbacks, more and more experts and researchers contribute to the study of replicas placement for HDFS. Y.W. Wang et al. propose Zport supplementary mechanism in [1] and prove that Zput not only accelerate the local data uploading procedure significantly but also meaningfully boost the remote block distribution. J. Xie et al. denote that ignoring data locality issue in heterogeneous environment can noticeably reduce the MapReduce performance and define two algorithms to implement its mechanism. One algorithm is used to initially distribute file fragments to heterogeneous nodes, and another one reorganize file fragments to solve the data skew problem[6], however, it fails to support various applications. W.W. Lin et al. describe an improved approach that dynamically allocates network bandwidth to achieve data load balance by introducing a control variable [7]. X.L. Ye et al. take real-time situation of node into consideration and demonstrate that the strategy behaves much better than the HDFS blocks placement when the situation is coherent to time [8]. N. M. Patel et al. [9] present an alternative parallel approach for efficient replica placement in HDFS to improve throughput. Z.D. Cheng et al. divide data into hot data and cold data to get better performance and higher disk utilization and put forward ERMS replication placement [10]. Q.S. Wei et al. propose a novel model to capture the relationship between availability and replica number and present CDRM to calculate and maintain minimal replica number for a given availability requirement based on the model[11]. X.L. Shao et al. take disk utilization into consideration and prefer to select the node with low storage [12]. O. Khan et al. present an algorithm that finds the optimal number of codeword symbols needed for recovery for any XOR-based erasure code and produces recovery schedules that use a minimum amount of data. They differentiate popular erasure codes based on this criterion and demonstrate that the differences improve I/O performance in practice for the large block sizes used in cloud file systems, such as HDFS [13]. H.H. Le et al. propose an evaluation of power-efficient data placement for HDFS in [14], but it will increase the network overhead significantly, due to data has to migrate from inactive node to active node.

In terms of the default Balancer procedure, K. Liu et al. deem that the default method cannot balance the overload rack preferentially, and it is likely to cause the breakdown of overload machines, so they focus on the overload machines and propose an improved algorithm for balancing the overload racks preferentially to reduce the possibility of breakdown of racks [15]. The priority method given by H. Rahmawan et al. in [16] is to set priorities to DataNodes so that it can distribute network load according to the priority, but network load is always

changing, so the fixed priority is not that suitable in terms of nodes' real-time and self-adaptability of HDFS.

In this work, we put forward a network load sensitive block placement strategy in heterogeneous Hadoop clusters. The improved strategy we propose considers both network load and disk utilization to ensure blocks can be placed on the node with relatively smaller network load if the difference of average utilization over the threshold we set. The simulation experiments demonstrate that our strategy performs much better not only in the balance placement but also significantly improves write throughput and saves more time than both the default blocks placement and the improved strategy in [8].

3. Default Block Placement Strategy and Balancer of HDFS

3.1 Default Block Placement Strategy of HDFS

"Rack Awareness" is one of the critical concepts of HDFS and the key iron rule of default block placement is that, for every block of data, two copies will be stored in one rack, another copy in a different rack.

When a HDFS client wants to write data to the system, the input file can be initially divided into a number of even-sized fragments. Then, NameNode allocates the blocks with a unique block ID and returns a list of nodes for placement in accordance with the default placement algorithm based on the nodes' disk utilization. As replica for each block is written into the cluster and all target nodes are selected, a pipeline from node-to-node is formed between the 3 DataNodes (or however many you have configured in dfs.replication) according to the distance between a DataNode and the client[15],[17]. When the length of bytes stream in the buffer reaches 512B, a Packet can be created and it won't be pushed into dataQueue until it arrives at 64K, consequently, bytes are pushed into the pipeline as a sequence of packets. Then DataStreamer pushes packets in the dataQueue to the first DataNode in the pipeline and the first DataNode forwards the packets to the second one, and so on. (i.e. as a DataNode is receiving block data, it will at the same time push a copy of that data to the next node in the pipeline). The next block will not begin until this block is successfully written to all three nodes. Furthermore, acknowledgement of data written is also received in pipeline. In the end , the client calls close( ) on the stream which flushes all remaining packets to the node and waits for acknowledgments before contacting with NameNode to signal that the file is complete. Here is a primary example of data written in HDFS[18]. Suppose a client uploads a file named "File.txt" to HDFS, and the file is divided into 3 blocks, named Blk A, Blk B, and Blk C respectively and each block has three copies, the possible access is shown as following **Fig. 1**:

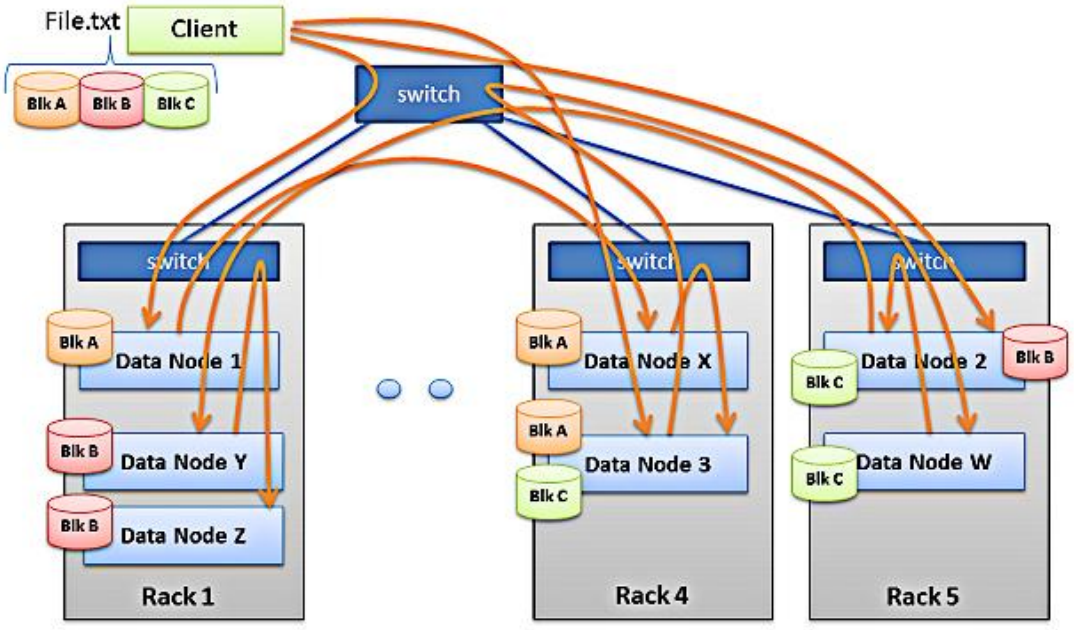


Fig. 1. HDFS multi-block replication pipeline

In the procedure above, how to select the optimal nodes to place replicas is an significant topic. According to the default placement strategy, local rack must be chosen and other racks and DataNodes are selected randomly. Actually, round-robin queue strategy is applied to the node selection in Hadoop 1.x version to achieve load balance and the source code in FSVolumeSet class is given as below:

Algorithm I: round-robin strategy in Hadoop 1.x

```
1.    synchronized FSVolume getNextVolume(long blockSize) throws IOException {
2.        int startVolume = curVolume;
3.        while (true) {
4.          FSVolume volume = volumes[curVolume];
5.          curVolume = (curVolume + 1) % volumes.length;
6.          //to check whether the remaining available space is enough for the block
7.          if (volume.getAvailable() > blockSize) {
8.    return volume;
9.           }
10.         if (curVolume == startVolume) {
11.          throw new DiskOutOfSpaceException("Insufficient space for an additional
      block");
12.         }
13.       }
14.     }
```

Specifically, HDFS places the 1st on the node where the client is located if the local node belongs to the cluster, otherwise, the replica would be placed on a random node in the cluster; the 2nd and the 3rd replicas on two different nodes in a different rack randomly, and the rest (if replication factor you set is larger than 3) are placed on random nodes with restrictions that no more than one replica is placed at one node and no more than two replicas are placed in the same rack when the number of replicas is less than twice the number of racks. If the first two replicas were placed on the same rack, for any file, two-thirds of its block replicas would be on the same rack.

3.2 Default Balancer Procedure of HDFS

Round-robin queue strategy used in Hadoop 1.x ensure all nodes in the cluster can be used to storage data, but this simple method always leads to imbalance due to the tremendous heterogeneities exist. Aiming at solving this problem, a Balancer program is designed to rebalance load and guarantee the cluster can achieve a well state of load balance. According to the average rate of all nodes disk space, DataNodes are divided into 4 types, including AvgUtilized DataNodes, over Utilized DataNodes, below AvgUtilized DataNodes and under Utilized DataNodes.

The basic idea of the default balance policy is to move replicas from DataNodes with higher utilization to DataNodes with lower utilization iteratively, and the move principle is [14]: firstly, replica is moved from over Utilized DataNodes to under Utilized DataNodes preferentially; next, replica is moved from over Utilized DataNodes to below AvgUtilized DataNodes; finally, data is moved from above AvgUtilized DataNodes to below AvgUtilized DataNodes. The Balancer adopts the principle of first balancing within one rack and then balancing among different racks to realize load balancing.

3.3 Problems of the Default Strategy in HDFS

The block placement is very important to HDFS throughput and MapReduce performance. The default strategy is highly fault-tolerant through data redundancy and it also reduces the inter-rack write traffic by the "replication pipeline" policy. Besides, the default strategy benefits from the "Rack Awareness" concept of HDFS as well, since the chance of a rack failure is far less than that of a node failure. More important, Hadoop provides a Balancer procedure to overcome the frequent imbalance load in the cluster. In a word, the default strategy places blocks efficiently and a balance procudure is designed to guarantee the cluster maintain balance.

However, there're still many shortcomings to be made up for. For instance, Current block placement assumes all nodes in the cluster are homogenous, actually, tremendous heterogeneities and great volume discrepancies existed among computing nodes. NameNode selects node just by "Rack Awareness" and a simple round-robin queue strategy without taking any resource characteristics into account. Consequently, load imbalance (some nodes are fully occupied, whereas others are idle very much) may occur frequently and the Balancer procedure would be called time and time again. Imbalance also occurs when new nodes are added to the cluster, which is likely to cause nodes' saturation and breakdown as soon as possible without balancing by the default Balancer. Moreover, the default placement strategy fails to adapt the changing of HDFS, since it's designed without considering the nodes' real-time state.

As to the default Balancer procedure, Balancer cannot work without the administrator manually calls (execute the command "sh $HADOOP_HOME/bin/start-balancer.sh –t 'threshold'"), and blocks migration would take up mounts of resources, such as network

bandwidth (the administrator can set new bandwidth by the command "hdfs dfsadmin -setBalanacerBandwidth newbandwidth") once the procedure is called. Furthermore, the principle of the default balancer procedure fails to optimize the overload racks preferentially, since it firstly balances within the rack and then balances among racks [15]. Even worse, the default Balancer procedure is designed to choose node from list in order. Node in the end of list cannot be rebalanced preferentially, despite its available disk space is used up or it crashes at once.

4. Proposed Network Load Sensitive Block Placement Strategy

We take impact of nodes great volume discrepancies and network load heterogeneities into considerations, and present a network load sensitive replica placement strategy in this paper. The proposed strategy is designed based on the "Rack Awareness" and pipeline mechanism of HDFS to guarantee that the strong fault tolerance and the inter-rack write traffic reduction properties of Hadoop are retained. By the improved policy, HDFS can maintain a well balance status during replica placement, rather than wasting large amounts of resources to rebalance the cluster by Balancer procedure.

4.1 Principle of the Improved Strategy

All nodes can be assorted into two parts including high group and low group in accordance with nodes' network load. If its network load exceeds ts_1 (a threshold we set), it will be assigned to low group, and otherwise it belongs to high group. Provided the difference of available storage space between the two groups is no larger than the ts_2 (a threshold we set), the nodes in the small group can be selected randomly so that it avoids placing replicas on the heavy network load nodes and saves much time .Otherwise, the improved strategy tend to select nodes with the largest remaining disk space in the cluster to guarantee the difference of available storage space among all nodes to stay in the range. Consequently, the cluster can maintain balance as far as possible.

4.2 System Model

We would like to take account of nodes' heterogeneities, including network load and disk space utilization, and at the same time avoid leaving nodes disk space unbalanced. According to this notion, we model our algorithm as follows:

Firstly, we define the packages number that the i th node deals with during a fixed period as its network load, i.e. $\mathrm{P}(i,t)$, $\mathrm{P}(i,t+\Delta t)$ respectively stand for packages number of the i th node deals at time t and $(t+\Delta t)$.The detailed network load expression is represented as:

$$\mathrm{P}(i,\Delta t)=\mathrm{P}(i,t+\Delta t)-\mathrm{P}(i,t) \qquad (i\in[1,N]) \tag{1}$$

where N stands for total number of nodes in the cluster.

Then, suppose there were two groups, named G_1 set and G_2 set, collecting nodes by rules. All nodes can be easily partitioned into these two sets by nodes' network load. The partition strategy can be evaluated as:

$$\mathrm{G}_1=\{\mathrm{j}|\mathrm{P}(i,\Delta t)<ts_1,\ (i\in[1,N])\} \tag{2}$$

$$\mathrm{G}_2=\{\mathrm{j}|\mathrm{P}(j,\Delta t)\geq ts_1,(j\in[1,N])\} \tag{3}$$

Here, ts_1 is a threshold the administrator sets .If network load of a node is smaller than ts_1, it would be gathered inG_1. Otherwise, it would be collected inG_2. Both i and j range from 1 to N. Accordingly, the whole set is regarded as G and expressed as:

$$\mathrm{G}=\{G_1\cup G_2\}=\{i\cup j|\mathrm{P}(i,\Delta t)<ts_1,\mathrm{P}(j,\Delta t)\geq ts_1,i\neq j,i,j\in[1,N]\} \tag{4}$$

In Eq.4, N represents the same meaning as we assumed above, i.e. $card(G) = N$ ($card$ is a mathematical symbol, which denotes the total number of elements in a certain set).

Next, denote $\Delta\overline{G}$ as average remaining available disk space, a critical factor to decide how to select an optimal node. It reflects nodes' performance difference overall. The detailed expression is given by:

$$\Delta\overline{G} = \frac{\sum_{\forall i \in G_1} G(i)}{card(G_1)} - \frac{\sum_{\forall j \in G_2} G(j)}{card(G_2)} \qquad (i \in G_1, j \in G_2) \tag{5}$$

Here, $G(i)$ is a function to work out the i th node available disk space, it's also called the i th node G value in this paper. $\sum_{\forall i \in G_1} G(i)$ indicates the sum of G values of all nodes in G_1. Similarly, $\sum_{\forall j \in G_2} G(j)$ means that of all nodes in G_2. Consequently, $\Delta\overline{G}$ reflects average difference of available remaining disk space between two groups.

Our aim is to apply the important factor $\Delta\overline{G}$ we calculate to achieve load-balancing replica placement. The detailed expression can be formulated as:

$$P = \begin{cases} RoundRobin(G_1) & \Delta\overline{G} < ts_2 \\ MAX(G(N)) & \Delta\overline{G} \geq ts_2 \end{cases} \tag{6}$$

where ts_2 is a threshold the administrator sets. P is a piecewise function including two parts. $RoundRobin(G_1)$ is a part that can select a node in G_1 set as Round-Robin queue strategy. $MAX(G(N))$ is another part ,which can pick out the node with the largest available disk space. If $\Delta\overline{G}$ we calculate as Eq.5 is smaller than ts_2, the strategy will select a node according the default policy of HDFS. Otherwise, the node with the largest available remaining storage space will be chosen to place the blocks.

Based on the principle and algorithm model above, the pseudo code of network load sensitive block placement strategy is given as follows:

Algorithm II: nodes selection algorithm

```
1.  {  for(int i = 0;i<n;i++)// To get each node network load by traversing all nodes
2.     NetworkLoad_i = n.getNetworkLoad();
3.     if(NetworkLoad_i < ts1)// To assort DataNodes as the threshold we set
4.     {  sum_low += n. getRemaining();
5.        i_low++;}
6.     else
7.     {  sum_high += n. getRemaining();
8.        i_high++;}
9.     avg_low = sum_low/i_low;// To get average available disk space
10.    avg_high = sum_high/i_high;//Ditto
11.    if(avg_high - avg_low < ts2)
12.       chooseNode = low_DataNodes.RoundRobin(nodes);
13. //Select a small network load node randomly ,if the difference is smaller than ts2
14.    else{ flag = max(rs1,rs2,…rsn);
15.       chooseNode = choosenNodes.get(flag);}
16. // Select the node with largest available disk space , if the difference is no larger than ts2
```

The explanations of pseudo code above are as follows:

(1) Line1-Line8: Get all the nodes' information, then calculate their network load, assort them and calculate the whole available disk space of heavy and small network load nodes respectively.

(2) Line9-Line10: Calculate the average available disk space of heavy and small network load nodes.

(3) Line11-Line16: Subtract the two average available disk space to get the network load overall. Then select the optimal node for block placement as the policy introduced above.

4.3 Design and Implementation

The proposed strategy in this paper has been conducted on Hadoop-1.2.1. The overall design is as follows:

(1)Firstly, construct a class named NetworkLoad in the package of org.apache.hadoop.hdfs.server.datanode, by which can we get all nodes' information, including network load;

(2)Secondly, rewrite heartbeat protocol of HDFS, so as to network load information can be sent and received by the rewritten heartbeat protocol. The functions need rewriting mainly include sendHeartbeat function in both NameNode class and DataNode class, handleHeartbeat function in FSNamesystem class;

(3)Thirdly, rewrite ChooseTarget function in BlockPlacementPolicyDefault class in the package of org.apache.hadoop.hdfs.server.namenode, which implements network load replica placement strategy.

Next, we will discuss how to implement the network load strategy as the design above. **Fig. 2** shows collaboration diagram of the network load sensitive replica placement algorithm.

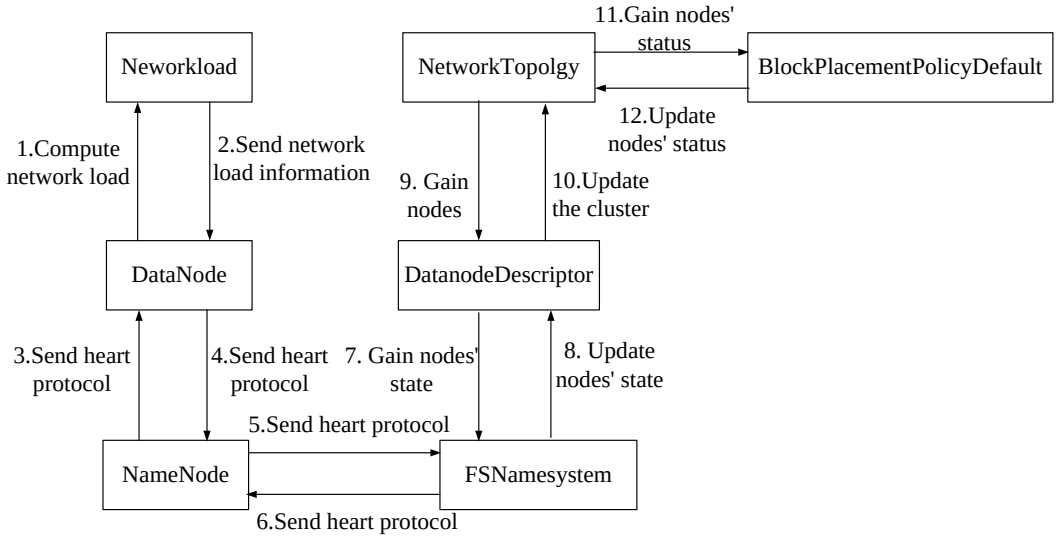


Fig. 2. Collaboration diagram of the proposed replica placement strategy

4.3.1 Compute Network Load

Construct NetworkLoad class in the package of org.apache.hadoop.hdfs.server.datanode and it's used to compute network load. It mainly includes getPackages function, getNum function and getNetworkLoad function, and they are implemented respectively as follows:

(1) GetPackages function is used to read each line of the information in the content of /proc/net/dev and it represents the cumulative number of sending and receiving packets.

Input the command of "cat /proc/net/dev" in the terminal, we can get information as **Fig. 3** shows:

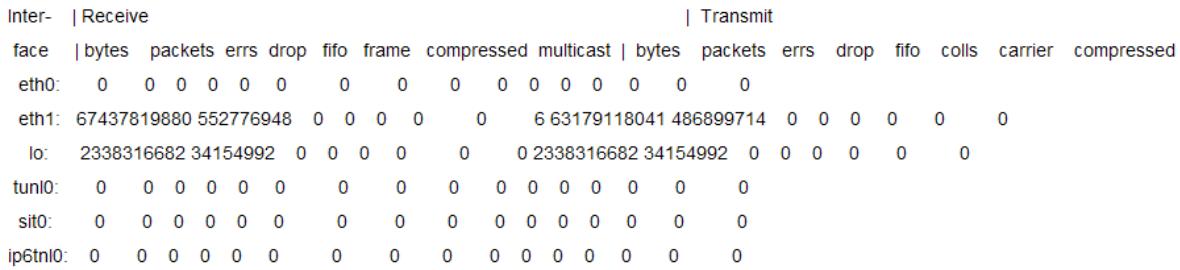


Fig. 3. Detailed information of network under path of "/proc/net/dev"

Some important parameters list as follows: Receive represents the number of receiving packets; Transmit means the number of sending packets; bytes depicts the whole bytes, receiving and sending included; packets describes the right packets number, err refers to the wrong packets number; drop points out the dropping packets number.

(2) GetNum function is used to convert the information saved by the function of getPackages to integer format and save the integer data.

(3) GetNetworkLoad function is used to get number of network load. It is called twice, respectively to calculate the number of sending or receiving data packets in a period of time, then subtract them to get the sending or receiving data packets during this period, the result is the so called the node's network load.

4.3.2 Rewrite Heartbeat Protocol

In HDFS, state of each DataNode is sent to NameNode by heartbeat protocol, but the default heartbeat protocol cannot transmit network load information we introduce. So we need to rewrite the default heartbeat protocol functions by adding network load variables to the formal parameter lists. The functions need rewriting in heartbeat protocol mainly include sendHeartbeat function in NameNode class, handleHeartbeat function in FSNamesystem class and updateHeartbeat function in DatanodeDescriptor class, all of these three classes lies under the package of org.apache.hadoop.hdfs.server.namenode. The purposes of these functions are as follows:

(1) SendHeartbeat function is to send the information of DataNode to NameNode.

(2) HandleHeartbeat function is to receive heartbeat sent by DataNode for NameNode.

(3) UpdateHeartbeat function is to update the information of the DataNode.

4.3.3 Implement the Model

Network load sensitive replica placement algorithm considers both network load and remaining available disk space, which is implemented by BlockPlacementPolicyDefault class under the package of org.apache.hadoop.hdfs.server.namenode .This improved class is responsible for mainly implementing replica placement by calling ChooseTarget function ,the premier work we need to do is :

(1)Define a dynamic array named chosenNodes, whose element is Node type, in the function of myChooseTarget we customize; it saves all nodes' information. Node is an interface which describes nodes' information in the HDFS, and implemented by DatanodeInfo class. We can get most information about node, such as its name, its total disk space and

its available disk space, etc. In addition, we also need to define a result array in myChooseTarget function, by which can we save the available disk space of each DataNode.

(2)Create a function named Select in the NetworkTopology class, which can return all nodes. Firstly Select function uses getNumOfLeaves function in the NetworkTopology class to get the number of the total nodes n; Then Select function circularly calls the function of getLeaf in this class and use index value from 0 to n to get corresponding DataNode; Finally Select function returns all nodes' information to the defined dynamic array chosenNodes.

(3) Save all the information about nodes in the dynamic array ChosenNodes, including the illegal nodes in excludedNodes. We need to remove these illegal nodes from the dynamic array. If the size of excludedNodes is larger than 0, remove DataNodes in excludedNodes circularly and delete them in chosenNodes accordingly.

(4) Return the optimal Datanode by chooseMyNode function from the chosenNodes. The pseudo code of the function of chooseNode has been given above, and its flow chart is as following **Fig. 4**:

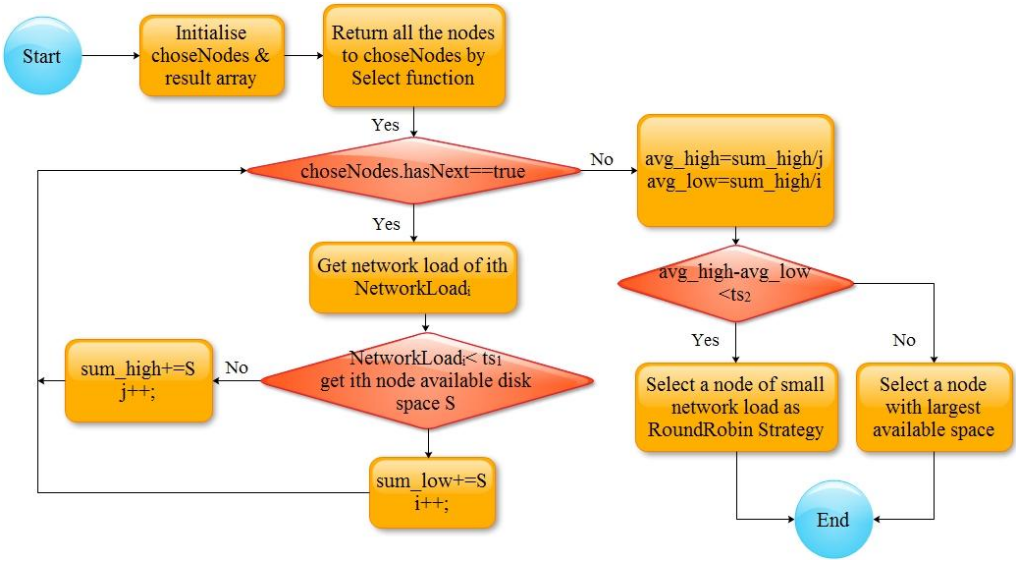


Fig. 4. Flow chart of the chooseNode policy

If the average remaining available disk space difference between high network load nodes and low network load nodes is smaller than ts_2 (we set 5G), NameNode will prefer to choose a node in low network load list as RoundRobin strategy; Otherwise, the DataNode with the largest available disk space will be selected for replica placement; The specific steps of the improved strategy are:

Step1: The selection of DataNode for placing the first replica is implemented by myChooseLocalNode function we customize in BlockPlacementPolicyDefault class. We judge whether the client node belongs to the cluster by contains function of NetworkTopology. If it does, we take priority to place replica on this node if possible.

Step2: The selection of DataNode for placing the second replica is implemented by myChooseRemoteRack function we customize in BlockPlacementPolicyDefault class. MyChooseRemoteRack tries to select optimal node on the rack but the rack the first replica placed on. If there is none, the function throws NotEnoughReplicasException and places the second replica on the same rack as the first one.

Step3: The improved strategy checks whether the first and second replica are placed on the same rack before the selection of DataNode for placing the third replica. If they are, the

strategy calls myChooseRemoteRack function as step2; else, myChooseLocalRack is called to place the third replica on the same rack as the second one.

5. Experiments and Analysis

The objective of this section is to determine the effectiveness of our implementation. We conducted following experiments to evaluate the effect data placement strategy on the data distribution: data written with different strategies, including the default block placement strategy (called DBPS), the policy considering nodes' real-time situation proposed in [8] (called RSBPS) and network load sensitive block placement represented in this paper (called NSBPS).

5.1 Experimental Environment

The experimental Hadoop is a commodity cluster consisted of 25 servers in a rack, node0 servers as Namenode, and nodes from node1 to node24 serve as DataNodes, and they are connected by a Gigabit network. In all experiments, Hadoop framework 1.2.1 and JDK 1.7.0_45 are used. The environmental configuration of system nodes is shown in **Table 1**.

Table 1. Environmental configuration of system nodes

	Number	Total Space(G)	Memory(G)	CPU	OS
NameNode	1	130.54G	18G	Xeon(R)2.40GHz	SUSE Server 10
DataNode	24	130.54G	18G	Xeon(R)2.40GHz	SUSE Server 10

The data we need to write includes 5 files of different sizes and the details of these files are shown in **Table 2**.

Table 2. Details of data information

File Name	Approximate Size(G)	Block(M)	Block Number
f0	50	64	813
f1	100	64	1626
f2	150	64	2439
f3	200	64	3252
f4	250	64	4065

5.2 Experimental Results and Analysis

We increase the network load of node1, node2, node3, node4 and node5 manually, and then upload files to HDFS. The experiments can be examined by two indicators. On the one hand, examine whether the 5 heavy network load nodes can be kept from being placed block and which placement strategy can make the cluster achieve balance status without Balancer. On the other hand, check which policy spent less time on uploading or see the impact of these strategies on HDFS write throughput.

Upload these 5 files and monitor nodes' remaining available disk space and the time spent on uploading, respectively. HDFS write performance is highly dependent on hardware and network environment (such as replication factor and block size). Results may vary as different cluster configuration environment varies.

The changes of remaining available disk space when uploading files with 3 different strategies are recorded as following **Table 3(a-c)**, in the table, column "Name" means the DataNode's name, column "Available Space" represents the available disk space before uploading files, and columns "Remaining Space After Uploading fx" records the remaining

space after uploading file fx; in the Used Space column, Space and ratio stand for node's total used disk space after all files written and proportion it accounts, respectively.

Table 3(a). Available disk space information by DBPS

Name	Available Space(G)	Remaining Space After Uploading fx (G)					Used Space	
		f0	f1	f2	f3	f4	Space(G)	Ratio (%)
node1	119.62	114.07	102.65	85.79	60.73	27.33	92.29	77.15
node2	119.57	112.76	100.01	81.29	54.96	23.51	96.06	80.34
node3	118.79	112.28	100.48	82.18	57.15	27.01	91.78	77.26
node4	119.37	112.34	101.49	84.16	54.64	21.19	98.18	82.25
node5	119.24	112.3	100.44	82.29	57.43	28.91	90.33	75.75
node6	119.27	112.24	98.99	78.71	50.79	23.91	95.36	79.95
node7	119.33	113.15	99.26	82.67	57.51	24.73	94.6	79.28
node8	119.68	112.42	101.74	85.17	55.01	27.72	91.96	76.84
node9	119.81	112.87	100.82	79.96	56.18	21.44	98.37	82.10
node10	119.74	112.93	99.69	84.64	54.52	22.14	97.6	81.51
node11	118.87	112.68	99.44	77.72	54.07	15.3	103.57	87.13
node12	119.36	114.44	103.19	84.34	60.99	34.13	85.23	71.41
node13	119.46	113.02	101.16	85.34	61.18	29.29	90.17	75.48
node14	119.38	115.59	102.46	83.11	59.06	22.51	96.87	81.14
node15	119.15	113.96	105.31	89.99	71.38	47.09	72.06	60.48
node16	119.39	114.02	105.11	84.65	64.7	36.25	83.14	69.64
node17	118.05	113	101.08	83.49	56.21	27.12	90.93	77.03
node18	119.01	113.14	101.85	83.13	59.28	25.19	93.82	78.83
node19	119.18	113.12	102.95	86.63	66.96	38.62	80.56	67.60
node20	119.44	114.27	100.45	80.65	55.48	24.49	94.95	79.50
node21	119.45	113.92	100.61	76.84	50.2	20.43	99.02	82.90
node22	119.46	113.4	102.11	81.83	60.41	29.88	89.58	74.99
node23	119.45	114.07	100.44	79.01	54.52	22.76	96.69	80.95
node24	119.82	114.21	101.65	80.15	51.84	19.09	100.73	84.07

Table 3(b). Available disk space information by RSBPS proposed in [8]

Name	Total Space(G)	Remaining Space After Uploading fx (G)					Used Space	
		f0	f1	f2	f3	f4	Space(G)	Ratio (%)
node1	119.62	113.38	101.34	81.86	57.72	26.47	93.15	77.87
node2	119.57	113.43	101.09	84.26	56.11	26.27	93.3	78.03
node3	118.79	112.74	100.86	81.06	57.21	25.02	93.77	78.94
node4	119.37	114.2	101.33	81.93	57.44	26.64	92.73	77.68
node5	119.24	112.9	101.15	83.91	57.31	25.91	93.33	78.27
node6	119.27	112.14	102.69	81.29	60.48	27.79	91.48	76.70
node7	119.33	113.04	101.9	84.18	58.37	27.33	92	77.10
node8	119.68	113.45	101.05	84.5	55.68	26.84	92.84	77.57
node9	119.81	112.41	102.12	81	60.83	30.96	88.85	74.16
node10	119.74	113.6	100.92	84.02	56.4	23.7	96.04	80.21
node11	118.87	113.28	101.33	82.09	57.39	26.79	92.08	77.46
node12	119.36	113.32	101.34	81.9	57.73	27.86	91.5	76.66
node13	119.46	113.72	101.3	82.92	61.12	28.93	90.53	75.78
node14	119.38	114.23	101.43	83.44	57.65	27.05	92.33	77.34
node15	119.15	113.96	101.18	82.98	57.81	26.19	92.96	78.02
node16	119.39	113.52	102.76	85.89	57.77	28	91.39	76.55
node17	118.05	112.5	100.02	79.54	54.61	22.04	96.01	81.33

node18	119.01	112.62	102.04	81.81	57.01	24.6	94.41	79.33
node19	119.18	114.12	101.29	82.56	57.69	26.63	92.55	77.66
node20	119.44	114.27	100.5	84.35	56.04	26.5	92.94	77.81
node21	119.45	112.63	101.71	81.91	57.47	26.26	93.19	78.02
node22	119.46	113.25	101.37	81.67	57.86	26.02	93.44	78.22
node23	119.45	113.42	100.68	81.51	56.74	26.47	92.98	77.84
node24	119.82	114.07	101.98	83.16	60.76	29.77	90.05	75.15

Table 3(c). Available disk space information by NSBPS in this paper

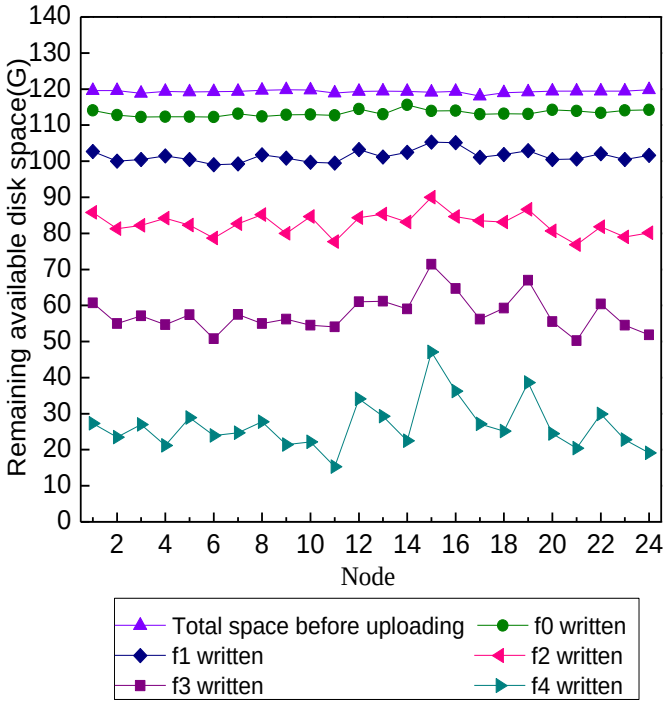
Name	Total Space(G)	Remaining Space After Uploading fx (G)					Used Space	
		f0	f1	f2	f3	f4	Space(G)	Ratio (%)
node1	119.62	114.51	102.58	85.4	61.07	30.03	89.59	74.90
node2	119.57	114.49	102.57	85.35	61.17	30.27	89.3	74.68
node3	118.79	113.67	101.73	85.26	61.2	29.94	88.85	74.80
node4	119.37	114.32	102.39	84.56	60.12	29.04	90.33	75.67
node5	119.24	114.05	102.12	84.61	59.81	29.05	90.19	75.64
node6	119.27	113.04	101.09	82.21	57.37	25.49	93.78	78.63
node7	119.33	112.99	101.05	81.86	57.06	25.16	94.17	78.92
node8	119.68	113.5	101.51	82.25	57.06	25.44	94.24	78.74
node9	119.81	113.53	101.56	81.98	56.72	25.66	94.15	78.58
node10	119.74	113.48	101.68	82.47	57.29	26.82	92.92	77.60
node11	118.87	112.63	101.15	82.01	56.74	26.08	92.79	78.06
node12	119.36	112.95	101.49	82.2	56.79	25.77	93.59	78.41
node13	119.46	113.17	101.21	82.02	56.65	25.85	93.61	78.36
node14	119.38	113.07	101.12	82.25	57.08	26.54	92.84	77.77
node15	119.15	113.1	101.09	81.65	56.81	25.48	93.67	78.62
node16	119.39	113.18	101.16	82.22	57.65	26.64	92.75	77.69
node17	118.05	111.94	99.97	80.94	56.04	25.55	92.5	78.36
node18	119.01	112.84	100.75	81.5	56.53	26.24	92.77	77.95
node19	119.18	113.13	100.95	81.05	56.23	25.1	94.08	78.94
node20	119.44	113.27	101.23	82.26	57.04	25.36	94.08	78.77
node21	119.45	113.19	101.07	81.62	56.6	26.01	93.44	78.23
node22	119.46	113.37	101.32	82.18	56.91	26.1	93.36	78.15
node23	119.45	113.12	101.07	83.1	57.88	26.08	93.37	78.17
node24	119.82	113.66	101.52	82.79	57.38	26.34	93.48	78.02

It's not difficult to calculate the standard deviation of the used space ratio with 3 different methods and the mathematical results are 0.0568, 0.0146, and 0.0135, respectively. According to the statistical theory, we know that the standard deviation represents the overall stability. The smaller the standard deviation, the much steadier the data presents. Compared with the values, the standard deviation of our placement is the smallest. Hence, NSBPS proposed in this paper performs much better than another two strategies, including DBPS and RSBPS. Next, we can depict the results in **Table 3(a-c)** into changing curves shown as following **Fig. 5**.

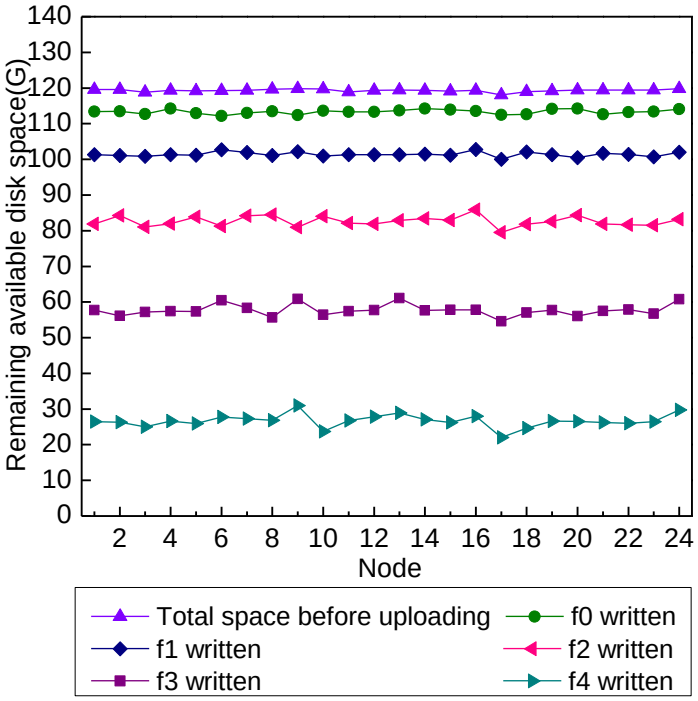
Fig. 5(a-c) respectively present the changing curves of remaining available disk space when uploading files with different placement policies. The horizontal axis coordinate represents each DataNode and the vertical axis coordinate denotes available disk space. Different lines mean uploading different files.

It's obvious to observe that curves in **Fig. 5(b) (c)** are much smoother than curve in **Fig. 5(a)**, while the first five nodes in **Fig. 5(c)** are a little higher than other nodes and all nodes

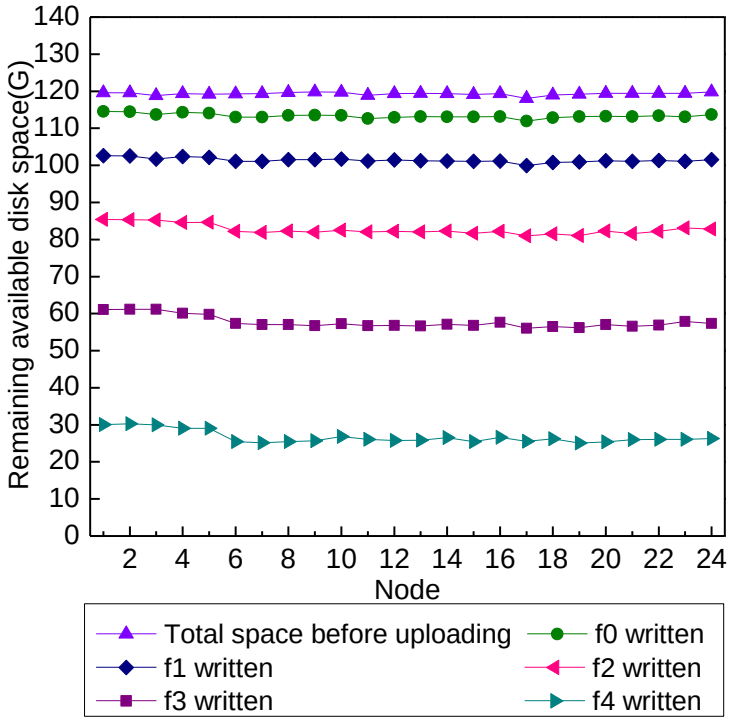
maintain the most balance. This is due to the impact of our strategy that all heavy network load nodes can be kept from being placed too much blocks based on nodes' balance on the whole, which reduces nodes' storage overhead, especially when confronted with heavy network load.



(a) Changing curve of remaining available disk space with DBPS



(b) Changing curve of remaining available disk space with RSBPS



(c) Changing curve of remaining available disk space with NSBPS

Fig. 5(a-c). Changing curve of remaining available disk space with different strategies

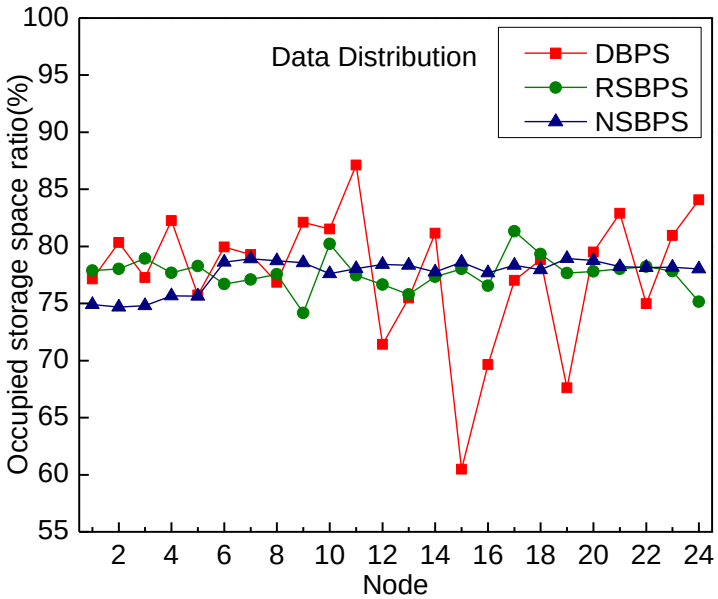


Fig. 6. Occupied storage space ratio with 3 different strategies

Fig. 6 reflects data distribution by different strategies over the cluster for block size of 64MB overall. It can be obtained from **Fig. 6** that the red curve representing the used disk space with the default strategy fluctuates wildly and many DataNodes need to be rebalanced by Balancer, while nodes' used space with RSBPS and NSBPS can be balance and it's obvious that the blue curve with NSBPS is much smoother than another two curves. This is because our replication management mechanism efficiently maintains load balance of the system.

What's more, as **Fig. 6** shown, the first five nodes' occupied space with our strategy are always lower than others, it strongly proves that those 5 heavy network load nodes, including node1, node2, node3, node4 and node5, are kept from being placed block in case of their risk of being saturated.

It can be concluded that, the improved replica placement policy in [8] and this paper can achieve much better balance than the default one, and the placement proposed in this paper not only performs the best balance but also prevents heavy network load nodes from being placed replica to reduce the possibility of network traffic jam and likelihood of nodes' breakdown significantly.

Now we are positioned to evaluate the impacts of data placement decisions on the execution time and throughput. In our experiment, we record the time that different strategies cost as writing each file.

Fig. 7. Execution time when uploading files with 3 different strategies

Fig. 7 shows that the execution time is very close when uploading f0, while with increase of the file size, our strategy behaves better and better. This is because when the file size is larger, more blocks and replicas will produce, which may accelerate imbalance and heavy network load. While our mechanism takes both disk space and network load into account so that it can prevent the heavy network load nodes from being occupied and tends to select optimal nodes with small network load or large disk space in preference in accordance with nodes' real-time status.

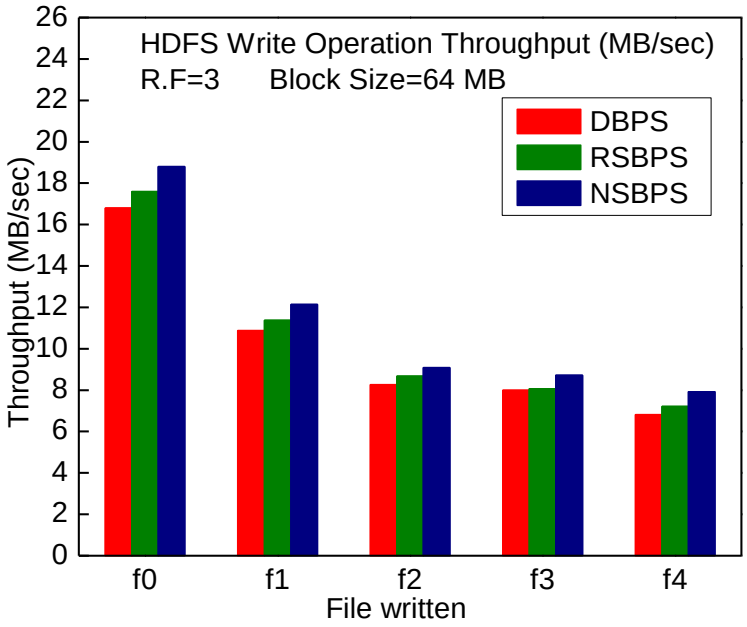


Fig. 8. Throughput when uploading files by 3 different strategies

Throughput of different approaches is shown in **Fig. 7**. And it can be observed that our strategy improves the throughput much more significant than RSBPS proposed in [8]. Besides, it also can be noticed that throughput decreases with file size increases in all approaches. Here, the main reason is that more blocks will produce when file size is larger which will increase total number of request from HDFS clients to NameNode and it increases network overhead heavily.

6. Conclusion

In this paper we take impact of resource characteristics, e.g. remaining available disk space and network load into consideration and propose a network load sensitive block placement strategy to achieve better performance than the default and another improved policy in [8]. Our approach is designed based on the principle that the strong fault tolerance properties of HDFS are retained. The experimental results demonstrate that the investigated strategy balances replica placement automatically and inteligently (without balancer procedure) so as to the blocks can be placed much more uniformly and it saves much more time and improves throughput significantly. Furthermore, the heavy network load nodes can be kept from being placed blocks while another two poclies try to place blocks on heavy network load nodes in spite of the risk of network traffic jam and nodes' saturation or breakdown. Consequently,the proposed replica placement strategy in this paper not only guarantees the reliability and scalability but it also improves performance of HDFS , especially when confronted with heavy network load.

References

[1] Y. W. Wang, C. Ma, W. P. Wang and D. Meng, "An approach of fast data manipulation in HDFS with supplementary mechanisms," *Journal of Supercomputing*, vol. 71, no. 5, pp. 1736-1753, May, 2015. Article (CrossRef Link).

[2] H. Zhuo, Z. Sheng and N. H. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *IEEE Transactions on Knowledge and Data Engineering*, vol.23, no.9, pp. 1432-1437, March,2011. Article (CrossRef Link) .

[3] P. P. Hung, M. A and E-N. H, "CTaG: An innovative approach for optimizing recovery time in cloud environment," *KSII Transactions on Internet and Information Systems*, vol.9, no.4, pp. 1282-1301, April, 2015. Article (CrossRef Link) .

[4] G. Sanjay, G. Howard and S-T. Leung, "The google file system," *Operating Systems Review (ACM)*, vol.37, no.5, pp. 29-43, October, 2003. Article (CrossRef Link) .

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *in Proc. of 6th Symposium on Operating System Design and Implementation (OSDI)*, pp. 137-150, December, 2004. Article (CrossRef Link) .

[6] J. Xie, S. Yin, X. J. Ruan, Z. Y. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters," *in Proc. of the 2010 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–9, April 19-23 ,2010. Article (CrossRef Link) .

[7] W. W. Lin and B. Liu, "Hadoop data load balancing method based on dynamic bandwidth allocation," *Journal of South China University of Technology (Natural Science)*, vol.40, no.9, pp. 42-47, September ,2012. Article (CrossRef Link) .

[8] X. L. Ye, M. X. Huang, D. H. Zhu and P. Xu, "A Novel Blocks Placement Strategy for Hadoop," *in Proc. of 2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, pp. 3-7, May 30-June 1,2012. Article (CrossRef Link) .

[9] N. M. Patel, N. M. Patel, M. I. Hasan, P. D. Shah and M. M. Patel, "Improving HDFS write performance using efficient replica placement," *in Proc. of the 5th International Conference on Confluence 2014*, pp. 36-39, September 25-26,2014. Article (CrossRef Link) .

[10] Z. D. Cheng, Z. Z. Luan, Y. Meng, Y. J. Xu and D. P. Qian, "ERMS: An Elastic Replication Management System for HDFS," *in Proc. of 2012 IEEE International Conference on Cluster Computing Workshops*, pp. 32-40, September 24-28, 2012. Article (CrossRef Link) .

[11] Q. S. Wei, B. Veeravalli, B. Z. Gong, L. F. Zeng and D. Feng, "CDRM: A Cost-effective Dynamic Replication Management Scheme for Cloud Storage Cluster," *in Proc. of 2010 IEEE International Conference on Cluster Computing*, pp. 188-196, September 20-24, 2010. Article (CrossRef Link) .

[12] X. L. Shao, Y. G. Wang, Y. L. Li and Y. W. Liu, "Replication Placement Strategy of Hadoop," *CAAI Transactions on Intelligent Systems*, vol.8, no.6, pp.489-496, January, 2013. Article (CrossRef Link) .

[13] O. Khan, R. Burns, J. Plank, W. Pierce and C. Huang, "Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads," *in Proc. of Conference on File and Storage Technologies (FAST)*, pp. 1-14, February 14-17,2012. Article (CrossRef Link) .

[14] H. H. Le, S. Hikida and H.Yokota, "Accordion: An efficient gear-shifting for a power-proportional distributed data-placement method," *IEICE Transactions on Information and Systems*, vol.98, no.5, pp. 1013-1026, May, 2015. Article (CrossRef Link) .

[15] K. Liu, G. C. Xu, and J. Yuan, "An Improved Hadoop Data Load Balancing Algorithm," *JOURNAL OF NETWORKS*, vol.8, no.12, pp. 2816-2822, December, 2013. Article (CrossRef Link) .

[16] H. Rahmawan and Y. S. Gondokaryono, "The simulation of static load balancing algorithms," *in Proc. of the 2009 International Conference on Electrical Engineering and Informatics*, pp. 640-645, August 5-7, 2009. Article (CrossRef Link) .

[17] K. Shvachko, H. R. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System," *in Proc. of 2010 IEEE 26th Symposium on MSST*, pp. 1-10, May 6-7, 2010. Article (CrossRef Link) .

[18] Understanding Hadoop Clusters and the Network, http://bradhedlund.com/.

Lingjun Meng received his B.E. degree from Luoyang Institute of Science and Technology, Luoyang, China in 2013. He is currently working towards his M.S. degree in Henan Polytechnic University, Jiaozuo, China. He got National Scholarship for Graduate in 2014. His research interests include cloud computing and data mining.
Email: menglingjun1109@126.com

Wentao Zhao received his B.E. degree from Henan Polytechnic University, Jiaozuo, China. He is a professor, M.S. advisor in School of Computer Science and Technology, Henan Polytechnic University and serves as director of Opening Project of Key Laboratory of Mine Information. His research interests include database for big data, networks, software systems and mine information.
Email: zwt@hpu.edu.cn

Haohao Zhao received her B.E. degree from North China University of Water Resources and Electric Power, Zhengzhou, China in 2012. She is currently working towards her M.S. degree in Henan Polytechnic University, Jiaozuo, China. Her research interest is data mining.
Email: vbvb4404@163.com

Yang Ding received his M.S. degree in Henan Polytechnic University, Jiaozuo, China, in 2014. His research interest is cloud storage.
Email: 379526563@qq.com