Send Orders for Reprints to reprints@benthamscience.ae

The Open Cybernetics & Systemics Journal, **2014**, *8*, 957-963

Open Access

An Improved Data Placement Strategy in a Heterogeneous Hadoop Cluster

Wentao Zhao[1, 2], Lingjun Meng[1], Jiangfeng Sun[1,2,*], Yang Ding[1], Haohao Zhao[1] and Lina Wang[1,2]

[1]*School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo 454000, China;* [2]*Opening Project of Key Laboratory of Mine Informatization, Henan Polytechnic University, Jiaozuo 454000, Henan, China*

Abstract: Hadoop Distributed File System (HDFS) is designed to store big data reliably, and to stream these data at high bandwidth to user applications. However, the default HDFS block placement policy assumes that all nodes in the cluster are homogeneous, and randomly place blocks without considering any nodes' resource characteristics, which decreases self-adaptability of the system. In this paper, we take account nodes heterogeneities, such as utilization of nodes' disk space, and put forward an improved blocks placement strategy for solving some drawbacks in the default HDFS. The simulation experiments indicate that our improved strategy performs much better not only in the data distribution but also significantly saves more time than the default blocks placement.

Keywords: Data placement, disk space utilization, HDFS, network load, nodes heterogeneity.

1. INTRODUCTION

With the era of explosive information and data coming, more and more fields need to deal with massive, large scale data. Cloud computing is emerging as a powerful paradigm for dealing with big data, 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 [1]. One of the key consideration of cloud computing is absolutely transparent for upper layer users who don't have to know the internal structure to reduce the burden on user.

Google has been dedicated to promoting application engines based on the techniques of GFS [2] (Google File System), MapReduce [3], Big Table [4] and so on. Hadoop [5-7] is an open-source implementation of Google's MapReduce [8] programming model, which is regarded as a significant project hosted by the Apache Software Foundation. HDFS (Hadoop Distributed File System) is the file system component of Hadoop and provides a number of APIs for researchers and programmers. The architecture of HDFS is master/slave and an HDFS cluster has a Namenode and multiple Datanodes. Files are divided into multiple blocks and distributed across Datanodes. A typical block size used by HDFS is 64MB and the replications factor is three by default. Hence, HDFS is highly fault-tolerant through data redundancy. Specifically speaking, each block of file will be replicated to multiple nodes to prevent the failure of one node from losing all copies of file as Fig. (**1**) shows:

However, the default HDFS block placement policy ignores the tremendous heterogeneities and great volume discrepancy existed among computing nodes in a cluster, and tries to place blocks randomly, which may easily result in the load imbalance occurrence as well as poor parallelism and low performance of the system. What's worse, some serious phenomenon may emerge, such as some nodes are too busy to receive requests owing to their capacity constraints or network congestions, while other nodes are idle so much, which may accelerate load imbalance and traffic jams, even lead some nodes crash.

Aiming at the problems mentioned, we propose an improved data placement strategy in heterogeneous Hadoop clusters in this paper. The simulation experiments demonstrate that our strategy performs much better not only in the balance placement but also significantly saving time than the default blocks placement.

2. THE DEFAULT BLOCK PLACEMENT STRATEGY OF HDFS

"Rack Awareness" is one of the key concepts of HDFS. 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. The flowchart of the default block placement is as the following steps:

Firstly, the policy judges whether the client is a node in the cluster or not by contains function in Network Topology class. If it is, the policy will try to select the client as the 1st selected node by chooseLocalNode function in ReplicationTargetChooser class. If it fails, a random node in the client will be selected by chooseLocalRack function in ReplicationTargetChooser class. If the client is not in the cluster, a random node in the cluster will be taken as the 1st node by chooseRandom function in ReplicationTargetChooser class. Then the 1st selected node information will be saved in the results array in DatanodeDescriptor of ReplicationTargetChooser class.

1874-110X/14

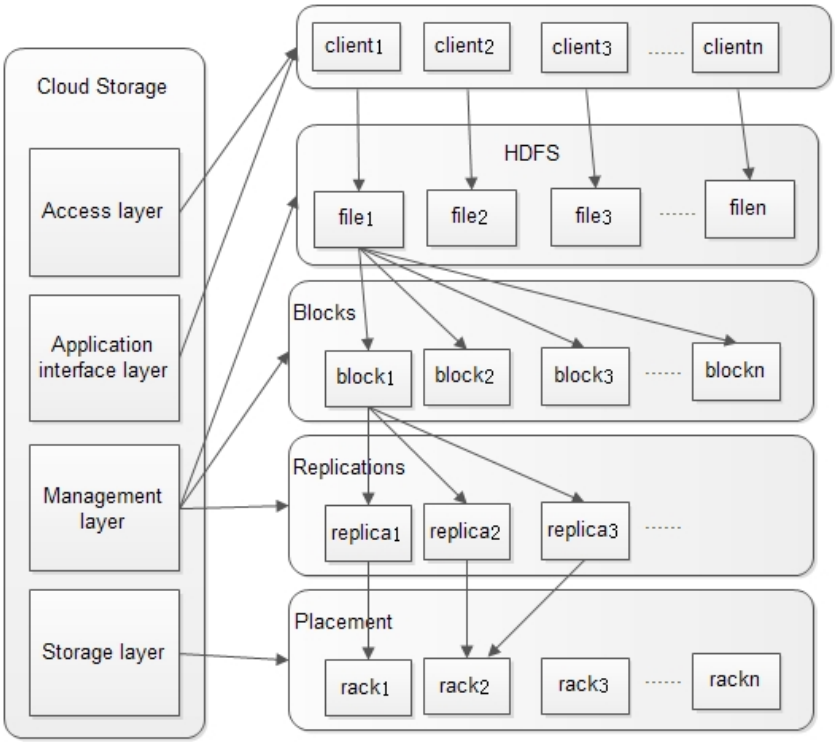

Fig. (1). The placement mechanism of replicas.

Secondly, a random node in the remote rack will be selected as the 2nd node by chooseRemoteRack in ReplicationTargetChooser class. If it fails, a random node in the same rack the 1st node will be chosen and saved in the results array.

Finally, the default policy will check whether the first two selected nodes are in the same rack by isOnSameRack function in NetworkTopology class, if they're, the policy tends to select a random node besides this rack by chooseRemoteRack in ReplicationTargetChooser class. A random node in the same rack as the 2nd node will be selected by chooseLocalRack function and regarded as the 3rd selected node. Then the 3 selected nodes in the array will be returned to the client.

3. PROBLEMS OF DEFAULT BLOCK PLACEMENT STRATEGY IN HDFS

The replicas placement is very important to HDFS performance. The default block placement strategy is highly fault-tolerant through data redundancy. Besides, it also reduces the inter-rack write traffic by the "replication pipeline" policy. What's more, 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. In a word, the default strategy not only places replicas efficiently, but also takes the network bandwidth and fault tolerance into consideration.

However, there're still many shortcomings to be made up for. Current block placement assumes all Datanodes in the cluster are homogenous, and Namenode select node just by "Rack Awareness" without taking any resource characteristics into account. Actually, tremendous heterogeneities and great volume discrepancies existed among computing nodes. The most serious aspect lies in the fact that the default policy ignores nodes' disk space utilization. Consequently, load imbalance may occur easily, that's to say, some nodes are fully occupied, whereas others are idle so much. Although HDFS provided a balancer procedure, which is deployed as an application program that can be run by the cluster administrator, that's to say, the default balancer procedure cannot work without the administrator manually calls (sh $HADOOP_HOME/bin/start-balancer.sh –t 'threshold'). Moreover, blocks migration would take up much network bandwidth once the procedure is called. What's worse, 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.

How to improve the default block load balancing strategy has become a critical subject, and a great number of scientists and researchers have been devoting themselves into improving performance of Mapreduce by designing optimal data placement for Hadoop cluster. The priority method given in reference [12] sets 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. Jiong *et al.* proposed a method to improve the performance of MapReduce by allocating data according to computing capacity of each node in reference [13], however, it does not take account disk space utilization and fails to support various applications at the same time. Besides, an evaluation of power-efficient data placement for HDFS was proposed in reference [14], it will increase the network overhead significantly, since it always has to migrate data from inactive node to active node.

4. AN IMPROVED BLOCK PLACEMENT STRATEGY IN HDFS

Hence, we take impact of nodes great volume discrepancies and network load heterogeneities into considerations, and put forward an improved data placement strategy in heterogeneous Hadoop clusters in this paper. We improve the default strategy based on the "Rack Awareness" of HDFS to guarantee that the strong fault tolerance properties of Hadoop are retained.

4.1. Principle of the Improved Strategy

The premier principle of this improved block placement strategy is as follows: we assort all nodes into two parts: high network load group and relatively small network load group. Provided the difference of network load between these two groups is not larger than the threshold we set, the nodes in the small network load group with the larger disk space can be selected in preference. Otherwise, the improved strategy tend to select nodes in the high network load group with the larger disk space, which realizes load balance as far as possible because this new placement strategy focuses on load balancing by selecting optimal node to place replica, instead of realizing balance by the default balancer procedure.

4.2. Algorithm Model

We would like to take account 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 during this period. *i.e.* $p(i,t)$, $p\left(i,t+\Delta t\right)$ respectively stand for packages number of the i th node at time t and ($t+\Delta t$). The detailed network load expression is represented as:

$$p\left(i,\Delta t\right)=p\left(i,t+\Delta t\right)p(i,t) \qquad i\in\left[1,N\right] \tag{1}$$

In Eq. (1), N represents the total number of nodes in the cluster.

Then, all nodes can be easily partitioned into two sets by nodes network load. If its network load is larger than the ts1, it will be taken as low network load node. Otherwise, it belongs to the high network load. The partition strategy can be evaluated as:

$$S=\left\{i \mid p(i,\Delta t)<ts_1, i\in\left[1,N\right]\right\} \tag{2}$$

$$L=\left\{j \mid p(j,\Delta t)\geq ts_1, j\in\left[1,N\right]\right\} \tag{3}$$

In Eq. (2) (3), S represents a set which contains small network load nodes, whereas L set contains large network load nodes. Both i and j range from 1 to N.

Next, denote as the critical factor to decide which node to select. It reflects nodes' performance difference overall. The detailed expression is given by:

$$\Delta\overline{G}=\frac{\sum_{\forall j\in L} G(j)}{card(L)}-\frac{\sum_{i\in S} G(i)}{card(S)} \qquad (i\in S, j\in L) \tag{4}$$

In Eq. (4), $\Delta\overline{G}$ is a function to work out the i th node available disk space (hereafter referred as G value), reflects average difference of G values between large network load nodes and relatively small ones on the whole.

In order to select an optimal node for replicas placement, $\Delta\overline{G}$ will directly decide how to choose an optimal node. The detailed expression can be formulated as:

$$P=\begin{cases} Max(S) & \Delta\overline{G}<ts_2 \\ Max(L) & \Delta\overline{G}\geq ts_2 \end{cases} \tag{5}$$

In Eq. (5), ts_2 is a threshold the administrator sets. Provided is smaller than ts_2, which illustrates that difference of G values between S set and L set is acceptable, we pick out node in the S set with the largest available disk space, since network load plays a more critical role for the selection. Otherwise, when the difference of G values between the two sets is too large, which proves that the small network load nodes are occupied intensively, we should select an optimal node from the L set with the largest available disk space, since disk space utilization plays a more critical role for the selection.

4.3. Algorithm Implementation

The proposed strategy in this paper has been conducted on Hadoop-0.20.2. The detail procedures are as follows:

(1) Firstly, we need to construct NetLoad class in the package of org.apache.hadoop.hdfs.server.datanode, by which can we compute nodes' network load. Then we create 3 functions in NetLoad class, including getPackages function, getNum function and getNetLoad function respectively. As its name suggest, getPackages function is used to calculate package number a node deals with during a certain period and the result is saved as string, getNum function can transfer the string result to integer format for mathematical operation. GetNetLoad function finally calculate network load by calling the two functions above twice and subtracting the two returned package number.

(2) Next, we need to rewrite heartbeat protocol, by which can we communicate with Namenode. The node network load is a novel concept we propose, so it can not to convey by the default heartbeat protocol. The functions we need to rewrite contain SendHeartbeat function in DataNode class, HandleHeartbeat function in FSNamesystem class, and UpdataHeartbeat function in DatanodeDescriptor class.

(3) Lastly but most significantly, we need to rewrite myChooseTarget function in ReplicationTargetChooser class, which implements blocks placement strategy directly, we need to create and rewrite functions as follows:

1) We need to define a dynamic array named chosenNodes in the function of myChooseTarget, by which can we save get most information of each node, *e.g.* 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 node.

2) We also need to create a function named Select in the NetworkTopology class, by which can we return all nodes. The core function in the improved algorithm is myChooseNetLoad function, by which can we change the default ran-

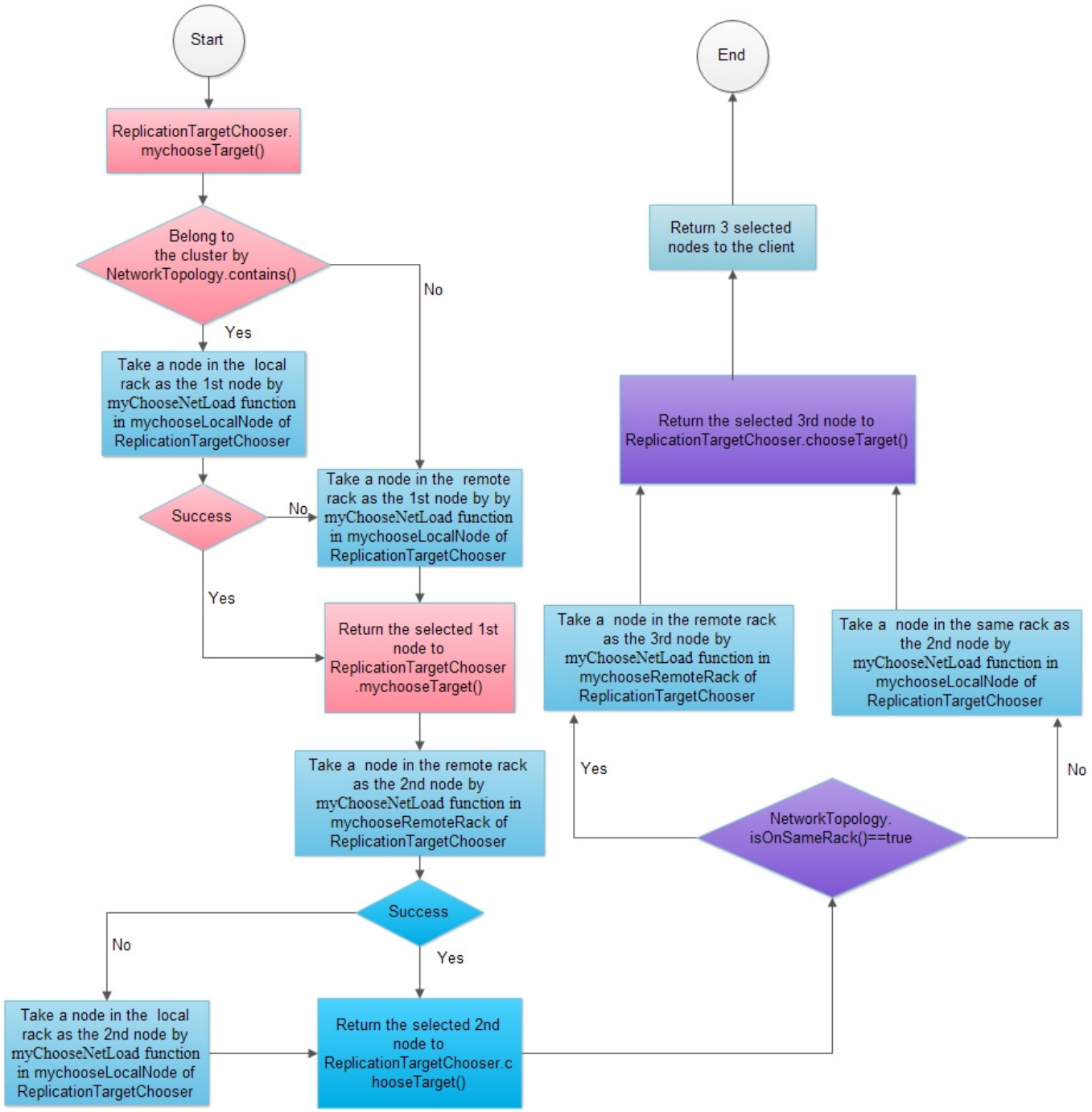

Fig. (2). The flowchart of the improved block placement.

dom mechanism and take both disk space utilization and network load into account and this core function will be called in the whole process of selecting nodes, (including 1st node, 2nd node and 3rd node).

3) We need to remove the illegal nodes from the dynamic array, since the array ChosenNodes saves all the information about nodes, including the illegal nodes in excludedNodes. Then the optimal node is returned by chooseMyNode function from the chosenNodes.

The total flowchart of our improved strategy is as following Fig. (**2**). the pink blocks, the blue blocks and the purple blocks coupled with the light blue blocks represent selection of the 1st, 2nd, and the 3rd nodes respectively:

5. EXPERIMENTAL RESULTS AND ANALYSIS

5.1. Experimental Environment

We implement our proposed strategy based on "Rack Awareness" concept of the default one, but we change the random way that it adopts, and take nodes disk space utilization and network load into consideration.

We present the experimental verification on Hadoop-0.20.2, and our experimental platform consists of 25 nodes.

Table 1. Node specification in a hadoop heterogeneous cluster.

	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

Table 2. Disk space used ratio by DBPS and IBPS.

Name	Disk Space used Ratio (%)		Name	Disk Space used Ratio (%)	
	DBPS	IBPS		DBPS	IBPS
node1	78.12	72.85	node13	75.76	75.74
node2	54.39	81.68	node14	79.48	77.48
node3	66.56	71.87	node15	79.49	79.86
node4	76.38	76.05	node16	79.26	79.4
node5	73.67	74.92	node17	79.51	79.33
node6	64.93	75.43	node18	79.43	79.29
node7	80.4	73.38	node19	75.63	75.61
node8	86.88	80.99	node20	78.28	75.68
node9	73.73	79.34	node21	79.22	79.3
node10	74.16	78.82	node22	79.73	78.45
node11	84.78	72.68	node23	78	79.41
node12	76.85	74.91	node24	79.34	79.41

We take node0 as Namenode, and nodes from node1 to node24 are regarded as Datanodes, and they are connected by a Gigabit network in a total rack. The environmental configuration of system nodes is shown in Table **1**.

We evaluate the effect by comparing data distribution after uploading big files to HDFS by these two different strategies, the file we intend to upload is around 750G, each replica of the file is the default 64M and total block number is 12195.

We write a script to increase some nodes' network load so that the final comparison of data distribution can be much more obvious. In our experiment, we accelerate network load of these 5 nodes: node1, node13, node17, node19 and node20.

5.2. Experimental Results and Analysis

We evaluate the effect of the data placement strategy on the data distribution by comparing the default block placement strategy (DBPS) and the improved block placement strategy (IBPS). We record the used disk space by DBPS and IBPS respectively and calculate the used ratio of each node, the detail results are as following Table **2**:

It can be obtained from Table **2** that the standard deviation of disk space used ratio by DBPS is 6.66, and the standard deviation of disk space used ratio by IBPS is 1.54, which demonstrates that IBPS performs much more balanced than DBPS. As Fig. (**3**) shown, our block placement strategy performs much more evenly and uniformly in the data distribution, and the platform keeps far more balanced if IBPS is adopted. What's more, once the balancer procedure is called, the investigated strategy can save much more time, because the unbalanced blocks that need to be migrated by the proposed policy are far less than that of the default one. This is because our IBPS mechanism maintains the load balance of the system efficiently, by taking the impact of network load and disk space utilization of each node into account.

Next, we investigate the effect of the time costs by comparing DBPS and IBPS. In our experiment, we record the time it costs as the size of the file from 50GB to 250GB. As shown in Fig. (**4**), the time that our block placement strategy costs is much less than the default one. The average time by using the default policy is 35257 s, whereas the average time by IBPS is 25270 s, which saves 28.32%. This is because our IBPS mechanism can prevent the heavy network load nodes from being occupied and tend to select optimal nodes with small network load and large disk space in preference.

CONCLUSION

In this paper, we take the impact of resource characteristics, *e.g.* disk space utilization and network load into consideration. Based on these considerations, we improve the default block placement strategy and put forward an improved

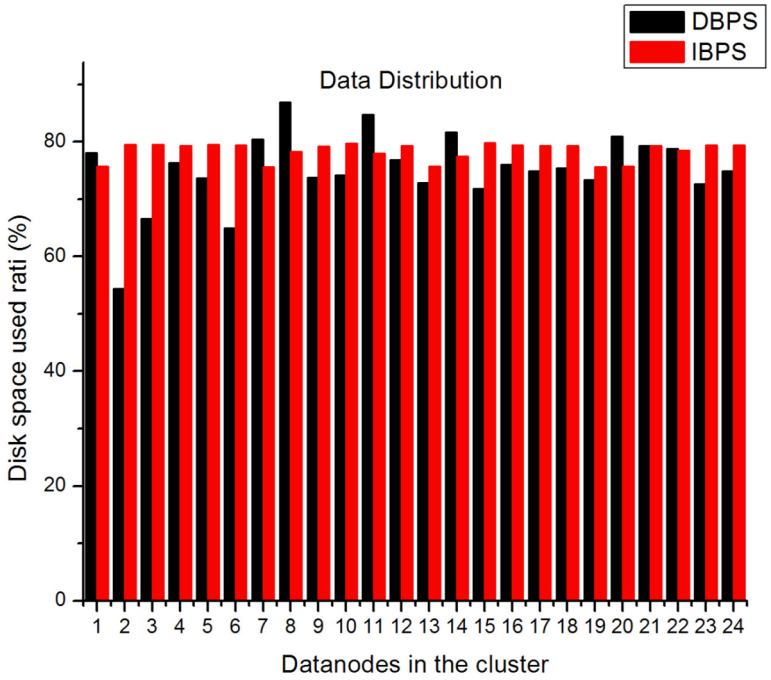

Fig. (3). Data distribution.

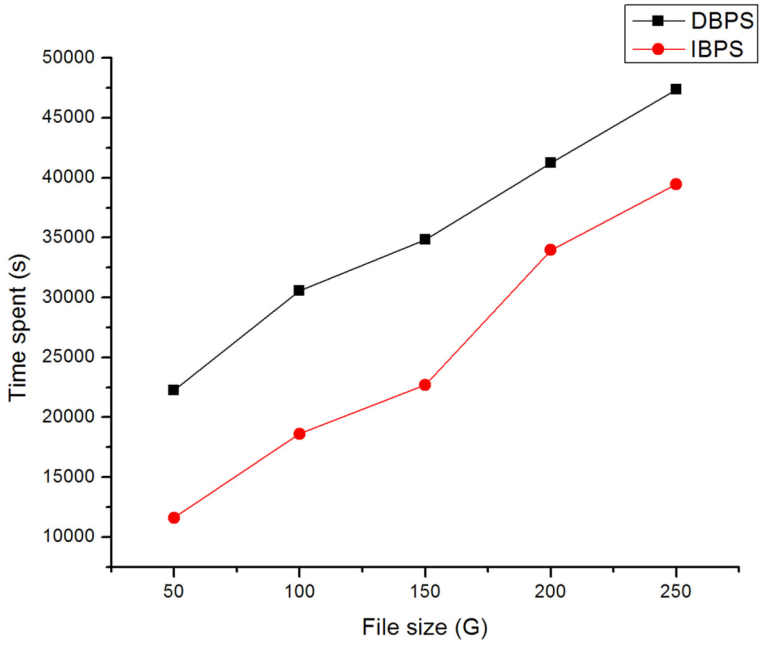

Fig. (4). Time spent on uploading files.

mechanism. Our approach is designed based on the principle that the strong fault tolerance properties of HDFS are retained. The experimental studies are conducted to demonstrate that the proposed strategy not only performs much more balanced (without balancer procedure) in the data distribution but also significantly saving time than the default strategy. Our future work will take account some other factors based on our current work, *e.g.* CPU load and real-time of each node to achieve a better performance.

CONFLICT OF INTEREST

No conflict of interest exits in the submission of this manuscript, and manuscript is approved by all authors for publication. I would like to declare on behalf of my co-authors that the work described was original research that has not been published previously, and not under consideration for publication elsewhere, in whole or in part.

ACKNOWLEDGEMENTS

This work is supported by the Provincial Key Technologies R & D Program of Henan under Grant No. 142402210435.

REFERENCES

[1] C.Lee, R.Buyya, P.Roe, "Future generation computer system", *Future Generation Computer Systems*. vol.18, pp.599-616, 2002.

[2] G. Sanjay, G. Howard and S-T. Leung, "The google file system",*Operating Systems Review (ACM)*, vol.37, pp. 29-43, 2003.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Proceedings of 6th Symposium on Operating System Design and Implementa-tion(OSDI)*, pp. 137–150, 2004.

[4] N. M. Patel, N.M Patel, M. I. Hasan, P. D. Shah and M. M. Patel, "Improving HDFS write performance using efficient replica placement", *Proceedings of the 5th International Conference on Confluence 2014*, pp. 36-39, 2014.

[5] Apache Hadoop, http://hadoop.apche.org/.

[6] X.L. Ye, M.X. Huang, D.H. Zhu and P. Xu, "A Novel Blocks Placement Strategy for Hadoop", *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, pp. 3-7, 2012.

[7] Z.D. Cheng, Z.Z. Luan, Y. Meng, Y.J. Xu and D.P. Qian. "ERMS: An Elastic Replication Management System for HDFS", *2012 IEEE International Conference on Cluster Computing Workshops*, pp. 32-40, 2012.

[8] 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," *2010 IEEE International Conference on Cluster Computing*, pp. 188-196, 2010.

[9] 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, pp. 489-496, 2013.

[10] 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", *Conference on File and Storage Technologies (FAST)*, pp. 1-14, 2012.

[11] 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, pp. 1013-1026, 2015.

[12] H. Rahmawan and Y. S. Gondokaryono, "The simulation of static load balancing algorithms", *Proceedings of the 2009 International Conference on Electrical Engineering and Informatics*, pp. 640-645, 2009.

[13] J. Xie, S. Yin, X.J. Ruan, Z.Y. Ding, Y. Tian and J. Majors, A. Manzanares, and X. Qin, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters", *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–9, 2010.

[14] K. Shvachko, H.R. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System", 2010 *IEEE 26th Symposium on MSST*, pp. 1-10, 2010.

Received: September 16, 2014 Revised: December 23, 2014 Accepted: December 31, 2014