ICIC Express Letters
Volume **10**, Number **2**, February **2016**

ICIC International ©2016 ISSN 1881-803X
pp. 425–431

AN ENERGY-EFFICIENT DATA REPLICA DYNAMIC PLACEMENT STRATEGY FOR CLOUD STORAGE

Qingying Lin[1,2] and Yuelong Zhao[2]

[1]Department of Computer Science and Engineering
Hanshan Normal University
Qiaodong Street, Chaozhou 521041, P. R. China
lqying@hstc.edu.cn

[2]School of Computer Science and Engineering
South China University of Technology
Panyu District, Guangzhou 510006, P. R. China
ylzhao1@scut.edu.cn

Received August 2015; accepted November 2015

Abstract. *Cloud storage system as an important component of cloud computing is the foundation of all kinds of cloud computing services. However, with the expansion of its scale and the energy consumption factors being ignored by its designers, the problem of high energy consumption and low efficiency is exposed. To solve energy saving of cloud storage system, an energy-efficient algorithm is proposed in this paper. The algorithm dynamically adjusts data replica placement according to the access heat and workload of DataNode. Meanwhile, to ensure data's availability and improve system's performance, the DataNodes actively apply for sleeping, and NameNode makes sure whether a DataNode can sleep. Experimental results show that the proposed algorithm can effectively reduce the energy consumption while guaranteeing the data availability. And the algorithm is more efficient when the system's workload and access heat are low.*

Keywords: Cloud storage, Energy-efficient, Data replica

1. Introduction.

Cloud storage system is served as the core underlying infrastructure for cloud computing. However, with the expansion of its scale and the energy consumption factors being ignored by its designers, the problem of high energy consumption and low efficiency is exposed. According to [1], it is predicted that the average IT energy consumption in 2025 will be 5 times as much as that of 2006, while that energy consumption of networking equipment will be up to 13 times. Meanwhile, Barroso and his teammate have done an investigation on more than 5000 Google servers for half a year [2]. The result shows that most of time the utilization of the server is between 10% and 50%. Even if the workload is lower than 10%, the energy consumption is 50% as much as the peak energy consumption. Obviously, the servers of Google have not been used efficiently. The main reason causing the low energy utilization in Google's servers is that the load balanced algorithm of GFS (Google File System) [3] averages its distribution of user's requests to all servers and uses the replica mechanism to ensure data availability, which improves the availability of the system but does not take the relationship between resource utilization and energy efficiency into account.

Hence, how to reduce the energy consumption but not to affect data availability has been an important element for building cloud storage system. In [4,5], it made some research on the access rules of data block in the HDFS (Hadoop Distributed File System) cluster of Yahoo. In order to save energy, it divided the storage area into Hot-zone and Cold-zone, and turned the nodes of the Cold-zone into sleeping or off mode. In [6], a cluster reconfiguration algorithm was designed for HDFS. To reduce energy consumption, Nitesh et al. dynamically reconfigured the cluster based on the current workload, and turned nodes on or off when the average cluster utilization rose above or fell below a specified threshold. In [7,8], an energy-efficient algorithm was proposed based on block storage structure reconfiguration within rack. The algorithm divided the rack into two storage areas, Active-Zone and Sleeping-Zone, reconfigured the data storage structure by calculating each data file's active factor, and turned the DataNode in Sleeping-Zone to sleep status in order to achieve energy savings. The data blocks are put into different zones and turn the nodes into sleeping or off mode. During the process, it is necessary to do a large number of data transferring operations among nodes. For a large-scale system, it will increase the cost to transfer the data across network and to ensure the data availability.

In this paper, we address an energy-efficient data replica dynamic placement strategy for cloud storage system. Although we focus on HDFS cloud storage system, our approach can also be applied to other similar systems. The remainder of the paper is organized as follows. We provide data models for energy-efficient algorithms in Section 2. Next, an energy-efficient data replica dynamic placement strategy is proposed in Section 3. The evaluation and simulation results of the proposed algorithms are given in Section 4. Finally, in Section 5 some conclusions are presented.

2. **Data Modeling.**

We model the data availability and energy-efficient requirements, and also propose the definition of energy-efficient problems under HDFS. HDFS has a master-slave architecture in which the master is called NameNode and the slaves are called DataNode. The NameNode is responsible for storing the HDFS namespace and it records changes to the file system metadata. HDFS is composed of multiple racks. The DataNodes are spread across multiple racks and store the data in their local file system. HDFS implements a rack-aware data block replica policy for the data. By default, the number of replicas of a data block is 3. The HDFS's replica placement policy is to put one replica of the block on one DataNode in the local rack, another on a different DataNode in some other rack, and the third on a DataNode in the same rack [9].

Definition 2.1. *Cluster DataNode Matrix. Let the HDFS cluster consist of c racks, and each rack with several DataNodes. The number of DataNodes may be different. Let r denote the maximal number of DataNode in rack, and the Cluster DataNode Matrix is denoted by $C_{r\times c}$.*

$$C_{r\times c} = \begin{bmatrix} dn_{11} & dn_{12} & \cdots & dn_{1c} \\ dn_{21} & dn_{22} & \cdots & dn_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ dn_{r1} & dn_{r2} & \cdots & dn_{rc} \end{bmatrix} \tag{1}$$

where, dn_{ij} $(1 \leqslant i \leqslant r,\ 1 \leqslant j \leqslant c)$ represents the ith DataNode in the jth rack in cluster. Let k denote the number of the DataNode in the jth rack. If $k < r$, $d_{mj} = 0$ $(k + 1 \leqslant m \leqslant r)$.

Definition 2.2. *DataNode State Matrix. DataNode may have a variety of states in the cluster. If the sets of the state are denoted as state $= \{0, 1, 2, 3\}$, 0 represents that there is no DataNode in the rack, while 1, 2, 3 represent DataNode in active state, sleeping state and failure respectively. The same point of sleeping and failure state is that DataNode cannot be used at that time. On the basis of Matrix $C_{r\times c}$, and the recent state of each DataNode, the DataNode State Matrix for building can be denoted by $DS_{r\times c}$.*

$$DS_{r\times c} = \begin{bmatrix} ds_{11} & ds_{12} & \cdots & ds_{1c} \\ ds_{21} & ds_{22} & \cdots & ds_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ ds_{r1} & ds_{r2} & \cdots & ds_{rc} \end{bmatrix} \tag{2}$$

Definition 2.3. *File's Block Storage Matrix. The sets of data files in system storage are marked as files* $= \{F_1, F_2, \cdots, F_w\}$*. According to rack-aware data block replica policy in HDFS, if the file* F_i $(1 \leqslant i \leqslant w)$ *divided into* n *data blocks, each of which has* m *replicas, different data blocks should be stored in different DataNodes. The position where* $n \times m$ *data blocks of file* F_i *are stored in DataNode can be denoted by Matrix* $S_{n\times m}$.

$$S_{n\times m} = \begin{bmatrix} bdn_{11} & bdn_{12} & \cdots & bdn_{1m} \\ bdn_{21} & bdn_{22} & \cdots & bdn_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ bdn_{n1} & bdn_{n2} & \cdots & bdn_{nm} \end{bmatrix} \tag{3}$$

where, $bdn_{ij} \in C_{r\times c}$ $(1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m)$*, the position of any data block in* F_i *can be found out quickly through File's Block Storage Matrix* $S_{n\times m}$.

Definition 2.4. *File's Block State Matrix. According to Definition 2.3 and Definition 2.2, all data block storage state of file* F_i *can be denoted by Matrix* $BS_{n\times m}$*. Here,* $bs_{ij} \in \{1, 2, 3\}$ $(1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m)$*, State 1 represents the data block stored in the active DataNode, which means this data block can be used. State 2 represents the data block stored in the sleeping DataNode. State 3 represents the data block stored in failed DataNode. Both state 2 and state 3 represent that the data blocks cannot be used. The availability of the file* F_i *can be judged by Matrix* $BS_{n\times m}$*. From the definition, the file can be used which has state 1 in each line of the Matrix* $BS_{n\times m}$ *at least.*

$$BS_{n\times m} = \begin{bmatrix} bs_{11} & bs_{12} & \cdots & bs_{1m} \\ bs_{21} & bs_{22} & \cdots & bs_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ bs_{n1} & bs_{n2} & \cdots & bs_{nm} \end{bmatrix} \tag{4}$$

Supposing k data blocks are stored in node dn_{ij}, the blockList can be used to denote the sets of all the data blocks. That is $blockList = \{block_1, block_2, \cdots, block_k\}$. Within the set time period T, the access heat of block and node can be defined.

Definition 2.5. *Block Access Heat. If the times of access is* num_i *within the set time period* T*, the access frequency is* $VF_i = num_i/T$*,* $block_i$ *and its access heat can be denoted as follows.*

$$BH\_block_i = \begin{cases} 1, & VF_i \leq h_1 \\ 2, & h_1 < VF_i \leq h_2 \\ 3, & h_2 < VF_i \leq h_3 \\ 4, & VF_i > h_3 \end{cases} \tag{5}$$

where, $1 \leqslant i \leqslant k$*,* h_1*,* h_2*,* h_3 *are three thresholds of access heat, which can be set up.*

Definition 2.6. *DataNode Access Heat. Supposing the whole access heat of DataNode is DH, it can be concluded according to Definition 2.5.*

$$DH = \left(\sum_{i=1}^{k} BH\_block_i \right) / k \tag{6}$$

where k *is the number of data blocks in DataNode.*

3. Proposed Energy-Efficient Algorithms.

In this section, we explain the energy-efficient algorithms for cloud storage system based on dynamic management of data replica. By recording the access information of each block in the DataNode, calculate the access heat of each DataNode within set time period T. When the access heat satisfies the sleeping conditions, DataNode will initiatively apply to NameNode for sleeping. Whether NameNode will agree on the sleeping or not depends on the recent workload of DataNode. When DataNode satisfies the sleeping conditions, it will keep itself in sleeping mode to save energy. To ensure the data availability, NameNode will manage the data replica dynamically. The algorithm process is completed as the following 4 main steps:

(1) Create fundamental data. To create fundamental data is the most basic step of energy-efficient algorithms. Fundamental data is stored in FSnamesystem of NameNode. The fundamental data includes the following.

Check the cluster DataNode status information, and create the Cluster DataNode Matrix $C_{r\times c}$ and the DataNode State Matrix $DS_{r\times c}$.

Check the cluster data file information, and create data file set $\mathit{files} = \{F_1, F_2, \ldots, F_w\}$.

Look for BlocksMap in FSnamesystem, and create the Block Storage Matrix $S_{n\times m}$ for each of file. Let set $sysS = \{S_1, S_2, \ldots, S_w\}$ denote each Block Storage Matrix of w data files, respectively.

By the Block Storage Matrix $S_{n\times m}$ and the DataNode State Matrix $DS_{r\times c}$, get the Block State Matrix $BS_{n\times m}$ of each file. Let set $sysBS = \{BS_1, BS_2, \ldots, BS_w\}$ denote each Block State Matrix of w data files, respectively.

(2) DataNode applies for sleeping algorithm. When the users of the cloud storage system inquire some information about accessing file, NameNode will look for the BlockMaps from FSnamesystem, and return the block identifier which corresponds to the file, and return the DataNode which corresponds to the block. According to the obtained information, the users can send an access request to the DataNode. After receiving the access request, DataNode will record the identifier of accessing block and the access time. And then it will return the data of the request file. Hence, DataNodes will record how many times each block has been accessed in the DataNode. If the access heat of the DataNode is lower than the specified threshold after a time period T, DataNode will apply to NameNode for sleeping.

Algorithm 1: Apply for sleeping algorithm

Input: The access counts set of k block in DataNode $vNum = \{num_1, num_2, \cdots, num_k\}$, the time period T, the threshold of access heat h_1, h_2, h_3, the threshold of sleep sh.

```
Step 1: for each $num_i$ in set $vNum$ do
Step 2:     calculate $VF_i$ by $VF_i = num_i/T$
Step 3:     calculate $BH\_block_i$ by Formula (5)
Step 4: end for
Step 5: calculate $DH$ by Formula (6)
Step 6: if $DH < sh$ then
Step 7:     SleepSender ($nodeId, blockList, T$)   //sent sleeping application
Step 8: endif
```

where, T is the time period of implementing the algorithm (one day, one week, etc.). Set up different values of T according to the characteristics of different clusters. The best way is to implement the algorithm when clusters are free. When reaching the sleeping time period, the DataNode kept in sleeping state will actively restore its running state, not being waken up by NameNode.

(3) Verify data availability algorithm. After receiving the sleeping request from DataNode, NameNode will check each block in DataNode. Firstly, get file from each block i; secondly, find the File's Block State Matrix, and pick up all replicas' storage state corresponding to block i. If there is only one replica in the active state, this block identifier will be put in the set of activeSoleList and return.

Algorithm 2: Verify data availability algorithm

Input: The DataNode that applies for sleeping *nodeId*, DataNode's block set *blockList*.

Output: The block set of only one replica in active state *activeSoleList*

```
Step 1:     for each $block_i$ in set $blockList$ do
Step 2:         $file \leftarrow$ getINodeFile($block_i$)
Step 3:         $BS \leftarrow$ get the Block State Matrix of $file$
Step 4:         $bsRow \leftarrow$ get a row in $BS$, which is the block state of all replicas correspond-
```

```
              ing to $block_i$
Step 5:       $active \leftarrow 0$
Step 6:       for each $bs_{ij}$ in $bsRow$ do
Step 7:         if $bs_{ij} == 1$ then $active$++;
Step 8:       end for
Step 9:       if $active == 1$ $activeSoleList \leftarrow block_i$
Step 10:    end for
Step 11:    return $activeSoleList$
```

(4) Judge and deal with sleeping algorithm. NameNode will judge whether DataNode meets the sleeping condition. If the length of *activeSoleList* is 0, it means more than one block replicas in the active state in this DataNode. Thus, NameNode agrees to sleep and modify DataNode State Matrix. If not, judge the workload of DataNode. If the workload of DataNode is less than the specified threshold, it will select the data block of the destination DataNode and exchange it. If the workload of the destination DataNode is more than the specified threshold and the selected data for exchanging not in the *activeSoleList*, which means more than one replica of exchanging block in active state, it will optimally select the DataNode from the same rack, and then agree to sleep and modify DataNode State Matrix.

Algorithm 3: Judge and deal with sleeping algorithm

```
Input: The DataNode set of applying for sleeping $Dn\_apply = \{dn_1, dn_2, \cdots, dn_t\}$
Output: The DataNode State Matrix $DS_{r\times c}$
Step 1:    for each $dn_i$ in set $Dn\_apply$ do
Step 2:      if $activeSoleList.length == 0$ then
Step 3:        Update $dn_i$'s state with 2 in DataNode State Matrix $DS_{r\times c}$
Step 4:    else
Step 5:      $Ud_i \leftarrow$ calculate current utilization of $dn_i$
Step 6:      if $Ud_i < WT$ then
Step 7:        $dn\_dest \leftarrow$ Select destination DataNode
Step 8:        for each $block_i$ in $activeSoleList$ of $dn_i$ do
Step 9:          $b_i \leftarrow$ Select block not in $activeSoleList$ of $dn\_dest$
Step 10:         $IntraRackTransfer(block_i, dn_i, b_i, dn\_dest)$
Step 11:       end for
Step 12:       Update $dn_i$'s state with 2 in DataNode State Matrix $DS_{r\times c}$
Step 13:     end if
Step 14:   end if
Step 15:   end for
Step 16:   return $DS_{r\times c}$
```

4. **Evaluation and Simulation Results.** In order to evaluate the energy-efficient algorithms proposed in this paper, we use CloudSim toolkit to simulate HDFS architecture and perform our experiment. Through the experiment, the active sleeping energy-efficient algorithm proposed in this paper, is analyzed and compared with the algorithm in [6] and [7]. To evaluate the performance of the algorithm and ensure the availability of the experimental data, the experimental cluster is composed by 10 racks with 10 DataNodes in each rack. Hence, there are totally 100 DataNodes in the cloud storage system. If the time period T is one day for implementing the algorithm, and the access heat threshold of the DataNode sh is 2, the threshold of DataNode workload rate WT is 50%. The number of files is varied from 200 for low workloads to 400 for high workloads and the file size is varied from 3000 MB for low workloads to 15000 MB for high workloads. And the frequency of accessing file is varied from 512 times in the previous five time periods to 5120 times in the back five time periods. The experimental results are shown in Figure 1 and Figure 2.

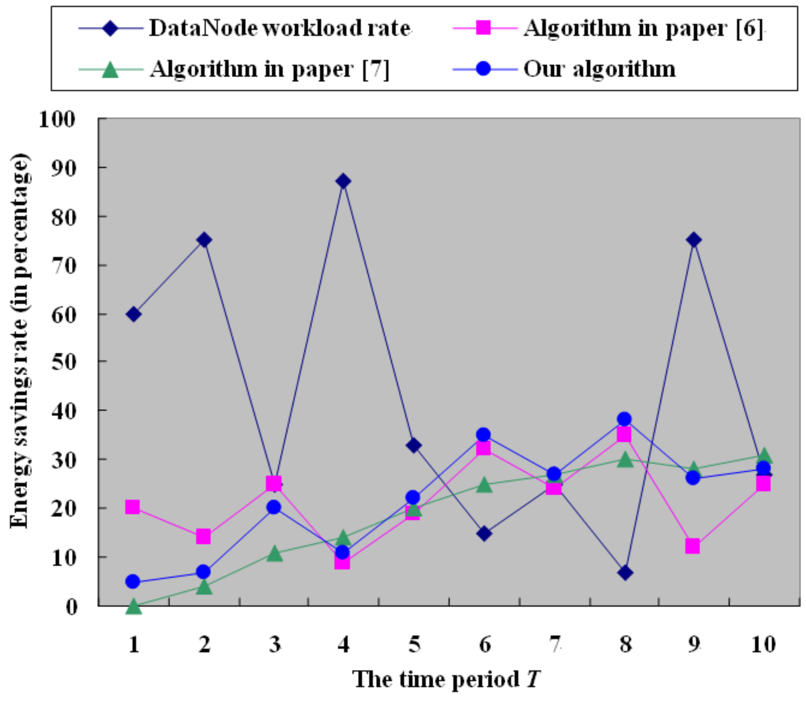

FIGURE 1. The energy saving rates under various workloads

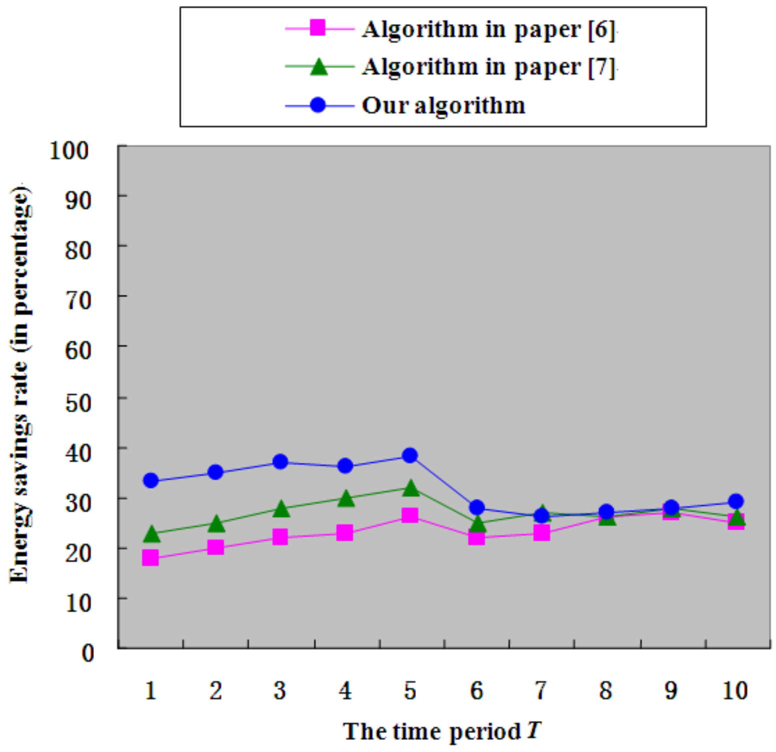

FIGURE 2. The energy saving rates under different accessing frequency

Figure 1 shows the different energy-saving rates between our algorithm and the algorithms of [6] and [7] under various workload rates. In [6], the algorithm changes with the change of DataNode workload. When the workload rate is higher, the rate of energy saving is lower; when the workload rate is lower, the rate of energy saving is higher. In [7], the algorithm is less influenced by the workload rate. After several running cycles, the data influence factors are calculated for the whole system, and the division of the area of rack tends to be stable, which makes the rate of energy saving steadier. While our algorithm changes with the changes of the DataNode workload. When the workload is lower, the rate of energy saving is higher. From Figure 2, we can know when the average accessing frequency is lower, the rate of energy saving is higher in our algorithm than the other two algorithms.

Moreover, we can know that the algorithm in [6] is more suitable for the workload fluctuation system, while the algorithm in [7] is more suitable for system that the data is accessed with regularity and the workload is stable. Our algorithm takes the access heat and the workload rate into account. Hence, it can be suitable for system that the data is accessed with low frequency and the workload is fluctuation. The energy-saving efficiency is more flexible and effective.

5. Conclusions and Future Work.

In this paper, an energy-efficient data replica dynamic placement strategy for cloud storage is proposed. We get the access heat of DataNode by calculating block's access rate. DataNode actively applies for sleeping based on access heat of DataNode. NameNode decides the sleeping mode on or off depending on the recent workload of DataNode and the data available state. Hence, it can dynamically manage the running state of the DataNode. Experiments show that it can increase server utilization and save energy effectively.

Acknowledgment. This work is partially supported by the Guangdong Science and Technology Planning Project of China (No. 2012B010100036), the Guangdong Science and Technology Planning Project of 2015 Public Research and Capacity Building.

REFERENCES

[1] D. Yun and J. Lee, Research in green network for future Internet, *Journal of Korean Institute of Information Scientists and Engineers*, vol.28, no.1, pp.41-51, 2010.

[2] L. A. Barroso and U. Hlzle, The datacenter as a computer: An introduction to the design of warehouse-scale machines, *Synthesis Lectures on Computer Architecture*, Morgan & Claypool Publishers, 2009.

[3] S. Ghemawat, H. Gobioff and S. T. Leung, The Google File System, *Proc. of the 19th ACM Symposium on Operating System Principles*, pp.29-43, 2003.

[4] R. T. Kaushik and M. Bhandarkar, Green HDFS: Towards an energy-conserving, storage-efficient, hybrid Hadoop compute cluster, *Proc. of the 2010 International Conference on Power Aware Computing and Systems*, pp.1-9, 2010.

[5] R. T. Kaushik, M. Bhandarkar and K. Nahrstedt, Evaluation and analysis of green HDFS: A self-adaptive, energy conserving variant of the Hadoop distributed file system, *Proc. of the 2nd IEEE International Conference on Cloud Computing Technology and Science*, pp.274-287, 2010.

[6] M. Nitesh, R. Nanduri and V. Varma, Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework, *Future Generation Computer Systems*, vol.28, no.1, pp.119-127, 2011.

[7] B. Liao, J. Yu, T. Zhang et al., Energy-efficient algorithms for distributed file system HDFS, *Chinese Journal of Computers*, vol.36, no.5, pp.1047-1064, 2013.

[8] B. Liao, J. Yu, T. Zhang et al., Energy-efficient algorithms for distributed storage system based on block storage structure reconfiguration, *Journal of Network and Computer Applications*, vol.48, pp.71-86, 2015.

[9] K. Shvachko, H. R. Kuang, S. Radia et al., The Hadoop distributed file system, *Proc. of the 26th Symposium on Mass Storage Systems and Technologies*, pp.1-10, 2010.