A Load Balancing Method Based on Node Features in a Heterogeneous Hadoop Cluster

Pengcheng Yang[1,3], Honghao Gao[1,2], Huahu Xu[1,3(✉)], Minjie Bian[1,3], and Danqi Chu[4]

[1] School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China
huahuxu@163.com
[2] Computing Centre, Shanghai University, Shanghai 200444, China
[3] Shanghai Shang Da Hai Run Information System Co., Ltd., Shanghai 200444, China
[4] Equipment Office, Shanghai University, Shanghai, People's Republic of China

Abstract. In a heterogeneous cluster, how to handle load balancing is an urgent problem. This paper proposes a method of load balancing based on node features. The method first analyses the main indexes that determine node performance. Then, a formula is defined to describe the node performance based on the contributions of those indexes. We combine node performance with node busy status to calculate the relative load value. By analysing the relative load value of each node and the cluster storage utilization rate, the recommended value of the storage utilization rate for each node is calculated. Finally, the balancer threshold is generated dynamically based on the current cluster's disk load. The results of experiments show that the load balancing method proposed in this paper provides a more reasonable equilibrium for heterogeneous clusters, improves efficiency and substantially reduces the execution time.

Keywords: Cloud computing · Hadoop · Heterogeneous cluster
Load balancing · Relative load

1 Introduction

With the rapid development of the Internet and information technologies, increasing amounts of research and business data are produced. The amount of data has reached the PB level [1], and a single machine is unable to handle such large-scale data. This situation has led to the creation of cloud computing. Hadoop is an open source distributed processing system developed by the Apache Foundation [2]. In the field of big data, Hadoop has become an outstanding platform that can handle intensive computing tasks. The platform consists of many nodes, and data is stored in those nodes. After a job is submitted, Hadoop splits the job into several tasks, distributes it to the nodes, and returns the results. In data-intensive supercomputing, the cost of moving data is much higher than the cost of moving computing [3]. Moving the process of calculation to the data nodes saves large amounts of network bandwidth, effectively reduces the number of data transfers, and improves cluster efficiency. However, when new nodes are added

I. Romdhani et al. (Eds.): CollaborateCom 2017, LNICST 252, pp. 344–354, 2018.
https://doi.org/10.1007/978-3-030-00916-8_32

or the number of files in the cluster change dynamically, the amount of data in each node can become unbalanced. When the data load among cluster nodes becomes unbalanced, many problems can arise. For example, the MapReduce program no longer makes good use of local computing, because the probability that the task must be assigned non-locally increases [4]. Thus, the node must copy the data from the other nodes, increasing the network load. The nodes cannot improve network bandwidth utilization and disk utilization. Consequently, the efficiency of the cluster is reduced and the execution time increases. Obviously, ensuring balanced data in the Hadoop cluster is very important. Data load balancing has gradually become an important research area in the field of distributed computing [5].

There are two solutions to this problem: static load balancing algorithms and dynamic load balancing algorithms. A static load balancing algorithm calculates the load of nodes using a pre-designed load balancing algorithm, which then allocates tasks [6]. However, this approach does not consider the node's resource load status. A static load balancing algorithm is simple to implement, but because it neglects the resource load, the final allocation scheme may not meet the requirements for load balancing and can lead to load imbalance. A dynamic load balancing algorithm considers the load information of the node to allocate tasks reasonably. However, because the load information must be calculated in real time, dynamic load balancing increases resource consumption [7]. Therefore, based on an analysis of the shortcomings of the two solutions, a data load balancing method based on node features for heterogeneous clusters is proposed.

The method proposed in this paper combines the static load balancing algorithm with the dynamic load balancing algorithm. First, the node performance is defined to calculate the node's data processing ability. Second, the node load status is evaluated by dynamically obtaining the current number of node connections. The node's relative load value is calculated by combining the node performance with the node load status. Then, node data is allocated considering the relative load value. Nodes with larger relative load values will be allocated more data.

The rest of this paper is organized as follows. Section 2 introduces load balancing and related works. Section 3 describes the construction of the load balancing method based on node features. The algorithm proposed in this paper is simulated and the simulation results are analysed in Sect. 4. Finally, Sect. 5 concludes and presents prospects for future work.

2 Related Works

As a framework for dealing with big data, Hadoop can fully utilize the advantages of large-scale clusters. The most important goals are to improve cluster efficiency and ensure full resource utilization. However, when the load in a cluster is unbalanced, problems will occur, such as low overall throughput of the cluster and resource underutilization. Many scholars have studied cluster load balancing through task scheduling, in which the load statuses and overall performance of clusters are analysed, and then, reasonable decisions are made. Gao et al. [8] conducted an in-depth study for implementing of MapReduce. The deviation of each node's running time in the Reduce

phase is caused by differences in the performances of the nodes in the cluster. A load balancing algorithm based on node performance was proposed to solve this problem. First, the algorithm creates a pre-allocated list, named RPReduceNum, based on the performance of all nodes. Data is allocated to Reduce tasks based on the RPReduceNum list, and high-performance nodes process more tasks in the Reduce phase. Therefore, this approach reduces the overall MapReduce running time and improves efficiency. However, it also causes large network consumption in the cluster. In the Reduce phase, the amount of received data for the Reducer is determined by a partitioning function. The Reducers with high input data become a performance bottleneck, which delays execution completion. To solve this problem, Fan et al. [9] proposed a method for storing the input data for a Reducer by creating a virtual partition. Each output of the Map task is assigned to a different virtual partition based on a hash function. All the virtual partitions are combined into the same number as the Reduce task by the LBVP algorithm, which ensures that the input data for each Reduce task is balanced. However, this approach ignores the problem of node heterogeneity. Zheng et al. [10] proposed a method of adaptive task scheduling method based on node capability. The tasks for nodes are distributed according to each node's history, current load status, performance, task characteristics and failure rate. Each node can adaptively adjust the number of running tasks, which decreases the time to completion for all tasks and improves node load balance. In HDFS, multiple copies of data blocks are placed on different DataNodes. Although this approach improves the fault tolerance of HDFS, the randomness of the default block may cause load imbalances between DataNodes. Lin et al. [11] proposed an improved load balancing algorithm to avoid adjusting data by using a balancer and minimizing block movement as much as possible to reduce the movement costs. This method mainly balances the loads of file blocks after addition or deletion, but it ignores the effects of dynamic addition or deletion in the cluster for load balancing. Wei et al. [12] proposed a new replica placement strategy to uniformly distribute the data, thus meeting the requirements of HDFS replica placement. This method does not need to run the balancer to make adjustments, but it still cannot avoid the problem of data imbalances after new nodes have been added or when node failures in clusters occur. Liu et al. [13] proposed an improved load balancing algorithm based on heterogeneous clusters that analysed four aspects of load imbalance: imbalance in input splits, imbalance in computation, imbalance in partition sizes, and imbalance in heterogeneous hardware. Xie et al. [14] proposed a strategy for storing data proportionally that involved distributing a large number of data sets to multiple nodes based on the computing ability of each node. In addition to the data redistribution algorithm for HDFS, a data reorganization algorithm was also implemented to solve the problem of data deviation caused by dynamic insertions or deletions. This strategy takes the heterogeneity of the nodes into account, but it ignores the impact of the heterogeneity of node storage on data storage.

First, this paper analyses the importance of default load balancing and the limitations exposed under heterogeneous clusters. Next, a method of load balancing for heterogeneous clusters is proposed. The method analyses the features of each node and then calculates a relative load value for each node. According to the relative load value,

the data for clusters can be allocated more reasonably. A large number of conducted experiments show that the method reduces cluster imbalance, improves cluster efficiency and reduces execution time under certain environments.

3 Improved Load Balancing Method

In this paper, we propose a load balancing method based on node features in heterogeneous clusters. The heterogeneity of a heterogeneous cluster is reflected by high performance nodes handling the same work in less time and the different storage allocated by each node to the HDFS. When the data is allocated proportionally according to the performance of each node, due to the heterogeneity of the node's storage, the node's disk may not support the allocated amount of data. The scheme proposed in this paper is based on the performance of each node and on HDFS storage to calculate the theoretical value of data allocation and the threshold used to determine whether a node is at equilibrium status. Finally, the load balancing problem of heterogeneous clusters is processed in a method similar to homogeneous clusters. This approach simplifies the complexity of load balancing for heterogeneous clusters.

Storage of Node $\left(C_{conf}(i)\right)$: Capacity is allocated to the HDFS by nodes and is not limited to the capacity of the node disk. $C_{conf}(i)$ represents the allocated capacity of the i^{th} node.

Used capacity $(C_{used}(i))$: The used capacity of the HDFS by the node's storage. $C_{used}(i)$ represents the capacity used by the i^{th} node.

CPU Performance of a Node $\left(P_{cpu}(i)\right)$: As is known, it is impossible for dual-core performance to reach a full 1 + 1 = 2 efficiency. By referring to the relevant information, in a multi-core CPU, the performance of each core is approximately only 0.8–0.9 times as much as that of a single-core CPU. Therefore, we set $\rho = 0.8$ and $\delta = 0.1$. The CPU Performance of a Node is defined as follows:

$$P_{cpu}(i) = \begin{cases} F_{cpu}(i), N_{core} = 1 \\ \left(\rho + \delta * e^{2-N_{core}(i)}\right) * F_{cpu}(i), N_{core} > 1 \end{cases} \quad (1)$$

where $N_{core}(i)$ represents the number of cores in the CPU, and $F_{cpu}(i)$ represents the CPU frequency of the i^{th} node in GHz.

Memory Performance of Node $(P_{mem}(i))$: We use $P_{mem}(i) = N_{mem}(i) * F_{mem}(i)$ to measure the memory performance of nodes, where $N_{mem}(i)$ represents the memory size of the i^{th} node in MBs. Here, $F_{mem}(i)$ denotes the memory frequency of the i^{th} node in GHz.

Connection Number of Node $(Q(i))$: This value indicates the number of active connections established by a DataNode with the outside world. These connections are generally used to transmit or receive data, send control signals, etc. The larger the

value, the busier the node and the greater its load. The sum of clustered connections is defined as:

$$\mathrm{Q} = \sum_{i=1}^{n} Q(i) \tag{1}$$

Definition 1: Relative load value:

$$L(i) = \alpha \sin^2\left(\frac{P_{cpu}(i)}{\max(P_{cpu})} * \frac{\pi}{2}\right) + \beta \cos^2\left(\frac{P_{mem}(i)}{\max(P_{mem})} * \frac{\pi}{2}\right) + \gamma \cos^2\left(\frac{Q(i)}{\max(Q)} * \frac{\pi}{2}\right) \tag{2}$$

The relative load value describes the node's ability to carry its current load. Here, α, β and γ represent weight parameters of CPU performance, memory performance and number of connections, respectively, and $\alpha + \beta + \gamma = 1$. The upper bound of the relative load value is set to 1 and will be simplified in later calculations. By analysing the contributions of these parameters, we set $\alpha = 0.5$, $\beta = 0.3$, and $\gamma = 0.2$. The sum of the relative load value is defined as follows:

$$L = \sum_{i=1}^{n} L(i) \tag{3}$$

Definition 2: Cluster storage utilization rate:

$$R_{Avg} = \frac{C_{used}}{C_{conf}} * 100\% = \frac{\sum_{i=1}^{n} C_{used}(i)}{\sum_{i=1}^{n} C_{conf}(i)} * 100\% \tag{5}$$

where n is the total number of nodes in the cluster.

Definition 3: Relative load storage capacity:

$$C_{ideal}(i) = \frac{L(i)}{L} * C_{used}(i) \tag{4}$$

Definition 4: Relative load storage utilization rate:

$$R_{ideal}(i) = \frac{C_{ideal}(i)}{C_{conf}(i)} * 100\% \tag{5}$$

According to the ratio of a node's relative load value to its total relative load value, the data can be distributed proportionally. We obtain the storage capacity of nodes based on its relative load, which is relative to its load storage capacity. For one node, the ratio of storage capacity based on its relative load value to the storage capacity assigned to the HDFS is defined as its relative load storage utilization rate.

Definition 5: Maximum load value:

$$M = \left(0.8 + 0.2 * \sin^2\left(\frac{\pi}{2} * R_{Avg}\right)\right) \tag{6}$$

Due to heterogeneity of node disks, the storage capacity of each node will differ. When the node's relative load value is high, the node's storage is fairly limited. At this point, the allocated data size is greater than the available node storage. For this problem, this article defines the maximum load value for the node. The maximum load value is dynamically adjusted according to the cluster storage utilization rate. When the cluster storage utilization rate is larger, the maximum load value of the node will be increased accordingly. The value of M is between 80% and 100%.

We find nodes whose data ratios are greater than the maximum load value and calculate the total overflow capacity. The total overflow capacity of the nodes in the cluster is calculated as follows:

$$C_{supr}(i) = \sum_{i=1}^{n} \left(R_{ideal}(i) - M\right) * C_{conf}(i) \tag{7}$$

where n is the total number of nodes in the cluster, $i = 1, 2, \cdots, n$ and $R_{ideal}(i) > M$. Then, data is allocated to other nodes. This iteration runs until the cluster contains no overloaded nodes—those whose allocated data ratio is greater than the node's maximum load value.

Definition 6: Relative load threshold:

$$T(i) = t * R_{ideal}(i) \tag{8}$$

where the parameter t is a threshold entered by the user. This indicates the maximum value of the deviation between the node storage utilization rate and the cluster storage utilization rate. In the default load balancer, if the deviation is less than the threshold, we assume that the node is balanced. Due to the heterogeneity of node storage, the actual storage expressed by the same threshold will be different between nodes. Therefore, this paper recalculates the threshold of each node by referring to its relative load value.

The procedure of the algorithm is as follows.

By obtaining the hardware parameters of all nodes, the CPU performance and memory performance are calculated according to formulas (1) and (2). Then, combined with the number of current connections, the relative load value and the sum of the relative load values will be calculated according to formula (3). Next, analysing the proportion of the relative load value to the sum, the relative load storage capacity and utilization rates are calculated according to formulas (6) and (7). The maximum load value will be calculated according to formula (8). To ensure that all the relative load storage values are valid values, we compare the relative load storage utilization rate with the maximum load value; the relative load utilization rate is set to the maximum load value when the former is larger than the latter. Next, the overflow capacity is calculated by formula (9). Overflow will then be allocated to other nodes until, eventually no node whose relative load utilization rate is greater than the maximum

load value will exist. After the threshold is entered, the relative load threshold is calculated for each node according to formula (10). All the nodes are divided into four groups as shown in Table 1.

Table 1. Node group

Groups	Conditions
overUtilizedDataNodes	$R(i) > R_{ideal}(i) + T(i)$
aboveAvgUtilizedDataNodes	$R_{ideal}(i) < R(i) < R_{ideal}(i) + T(i)$
belowAvgUtilizedDataNodes	$R_{ideal}(i) - T(i) < R(i) < R_{ideal}(i)$
underUtilizedDataNodes	$R(i) < R_{ideal}(i) - T(i)$

4 Experiments and Analysis

The following describes the simulation experiments conducted for this study. The experimental environment consists of three racks containing nine nodes. There are two nodes in rack 1, four in rack 2, and three in rack 3. The NameNode is located in rack 2, and its number is PC3. This node also acts as a DataNode. All the nodes are configured with the CentOS 7 operating system. The network topology is shown in Fig. 1. In the experiment, the load balancer is run on the NameNode. The hardware parameters of all the nodes are shown in Table 2.

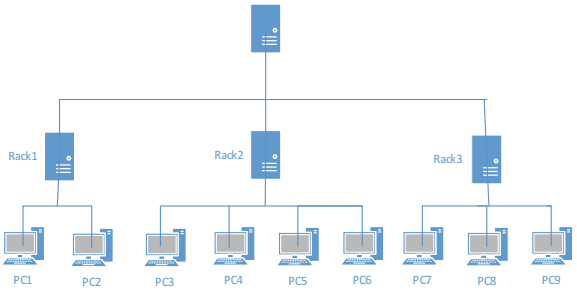

Fig. 1. Network topology

All the nodes are sorted in descending order by relative load value. PC1 is the node with the greatest relative load value and PC9 is the node with the lowest relative load value. The relative load values of all the nodes are shown in Fig. 2.

To compare the default load balancer with the improved load balancer in this heterogeneous cluster, the data block copies parameter is set to 3, and we delete some data to force the cluster to be unbalanced. We execute the instruction start-balancer.sh–threshold 10, which means that we set the threshold to 10% and execute the load balancer. Then, we run the balancer separately and observe the equilibrium effect. Finally, the WordCount program is executed to record the finish time for each balanced scheme. The cluster statuses are shown in Tables 3 and 4 (Fig. 3).

Table 2. Node hardware parameters

Number of node	CPU	Memory	Storage
PC1	3.5 GHz (4 CPUs)	4 GB (1.6 GHz)	80 GB
PC2	3.5 GHz (4 CPUs)	4 GB (1.6 GHz)	60 GB
PC3	3.5 GHz (4 CPUs)	4 GB (1.6 GHz)	40 GB
PC4	3.5 GHz (4 CPUs)	4 GB (1.6 GHz)	40 GB
PC5	2.1 GHz (2 CPUs)	3 GB (1.6 GHz)	70 GB
PC6	2.6 GHz (2 CPUs)	2 GB (1.333 GHz)	60 GB
PC7	2.6 GHz (2 CPUs)	2 GB (1.333 GHz)	30 GB
PC8	2.0 GHz (2 CPUs)	2 GB (1.333 GHz)	40 GB
PC9	2.2 GHz (2 CPUs)	1 GB (1.333 GHz)	50 GB

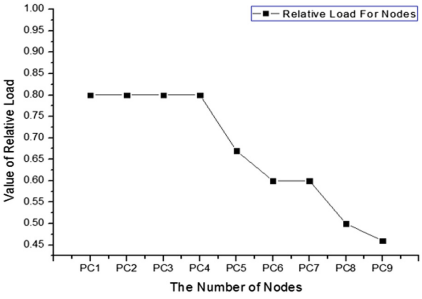

Fig. 2. Relative load value

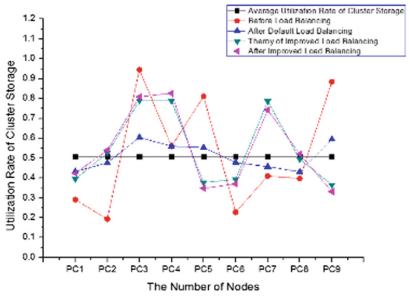

Fig. 3. Cluster storage utilization

Table 3. Cluster initial status

Number of node	Storage (GB)	Used (GB)	Used (%)	Relative load value
PC1	80	23.3	29.13	0.8
PC2	60	11.6	19.33	0.8
PC3	40	37.8	94.50	0.8
PC4	40	22.4	56.00	0.8
PC5	70	56.7	81.00	0.67
PC6	60	13.7	22.83	0.6
PC7	30	12.3	41.00	0.6
PC8	40	15.9	39.75	0.5
PC9	50	44.2	88.40	0.46

In Fig. 4, after running the improved load balancer, the data storage distribution is consistent with the trend in Fig. 2. This result shows that the improved load balancer works as expected.

The improved balancer will adjust the data distribution into an improved equalization status based on the relative load value. To verify that the improved load

Table 4. Cluster load status

Number of node	Default balancer			Improved balancer				
	Threshold T (%)	Balanced (%)	Balanced (GB)	Ideal storage (%)	Ideal storage (GB)	Threshold T (%)	Balanced (%)	Balanced (GB)
PC1	10	43.27	34.62	39.45	31.56	3.95	42.40	33.92
PC2	10	47.68	28.61	52.59	31.56	5.26	53.88	32.32
PC3	10	60.43	24.17	78.89	31.56	7.89	80.83	32.33
PC4	10	56.00	22.4	78.89	31.56	7.89	82.64	33.06
PC5	10	55.32	38.72	37.68	26.38	3.77	34.80	24.36
PC6	10	47.81	28.68	39.34	23.60	3.93	37.02	22.21
PC7	10	45.72	13.72	78.69	23.60	7.87	74.27	22.28
PC8	10	43.12	17.25	49.53	19.81	4.95	52.06	20.82
PC9	10	59.46	29.73	36.54	18.27	3.65	33.18	16.59

balancer maintains a good equalization effect, the WordCount program is executed to compare the finish times. WordCount is the demo program used for MapReduce on the Hadoop official website; it reads a text file and counts the frequency of words in the file. WordCount programs are I/O-intensive jobs and highly sensitive to data distribution. The data is distributed in the cluster before running the program, and the program is run 20 times.

Figure 5 shows the finish times for jobs from different balancers. The abscissa represents the sequence number of jobs, and the ordinate represents the finish time of jobs. The 20 groups of contrasting experiments show that there are still some uncontrollable factors in the experiment, and consequently, some deviation appears. However, the overall results are relatively stable, and the average completion time of the default balancer is longer than that of the improved balancer. These results verify that the improved balancer has a substantial effect on cluster performance.

In a data distribution-balanced cluster, the proposed balancer can effectively reduce the data migration problems of computing resources and data resources in different physical nodes caused by job scheduling. Thus, the network load is alleviated, the number of non-localized tasks decreases, and the execution time is reduced.

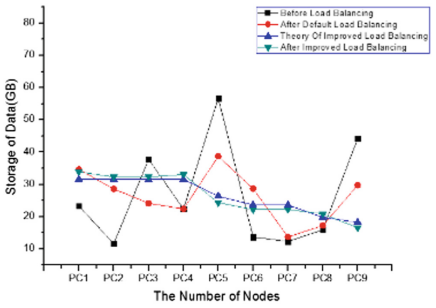

Fig. 4. Load balancer comparing experiment

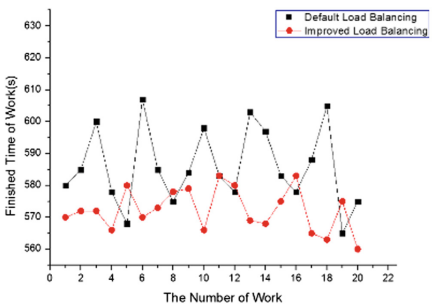

Fig. 5. Jobs running status

5 Conclusion

In this paper, we analysed the working principles of the default load balancer. However, in heterogeneous clusters, based on the performance differences, storage and current loads of each node, a load balancing method based on node features is proposed. This method provides a scheme for calculating a relative load value for each node. Then, the amount of data to be allocated and the appropriate threshold are calculated by considering the relative load value. Next, we move data and ensure that the deviations of the storage utilization rate are not greater than the threshold. Finally, through experimental analysis, the method is shown to cause the cluster to be more balanced, reduce the execution time of jobs, and improve the overall performance of the cluster.

Acknowledgements. This paper is supported by National Natural Science Foundation of China under Grant No. 61502294, Natural Science Foundation of Shanghai under Grant No. 15ZR1415200, CERNET Innovation Project under Grant No. NGII20160210, NGII20160614, NGII20160325, The Special Development Foundation of Key Project of Shanghai Zhangjiang National Innovation Demonstration Zone under Grant No. 201411-ZB-B204-012, and The Development Foundation for Cultural and Creative Industries of Shanghai under Grant No. 201610162.

References

1. Su, F., Peng, Y., Mao, X., et al.: The research of big data architecture on telecom industry. In: International Symposium on Communications and Information Technologies, pp. 280–284 (2016)
2. Apache Hadoop. http://hadoop.apache.org
3. Parsola, J., Gangodkar, D., Mittal, A.: Efficient storage and processing of video data for moving object detection using Hadoop/MapReduce. In: Lobiyal, D.K., Mohapatra, D.P., Nagar, A., Sahoo, M.N. (eds.) Proceedings of the International Conference on Signal, Networks, Computing, and Systems. LNEE, vol. 395, pp. 137–147. Springer, New Delhi (2017). https://doi.org/10.1007/978-81-322-3592-7_14
4. Bezerra, A., Hernandez, P., et al.: Job scheduling for optimizing data locality in Hadoop clusters. In: European MPI Users' Group Meeting, pp. 271–276 (2013)
5. Lin, W.W., Liu, B.: Hadoop data load balancing method based on dynamic bandwidth allocation. Huanan Ligong Daxue Xuebao/J. South China Univ. Technol. **40**(9), 42–47 (2012)
6. Fan, K., Zhang, D., Li, H., et al.: An adaptive feedback load balancing algorithm in HDFS. In: International Conference on Intelligent NETWORKING and Collaborative Systems, pp. 23–29 (2013)
7. Babu, B.G., Shabeera, T.P., Madhu Kumar, S.D.: Dynamic colocation algorithm for Hadoop. In: International Conference on Advances in Computing, Communications and Informatics, pp. 2643–2647 (2014)
8. Gao, Z., Liu, D., Yang, Y., et al.: A load balance algorithm based on nodes performance in Hadoop cluster. In: Network Operations and Management Symposium, pp. 1–4. IEEE (2014)

9. Fan, Y., Wu, W., Cao, H., et al.: LBVP: a load balance algorithm based on Virtual Partition in Hadoop cluster. In: IEEE Asia Pacific Cloud Computing Congress, pp. 37–41. IEEE (2012)
10. Zheng, X., Ming, X., Zhang, D., et al.: An adaptive tasks scheduling method based on the ability of node in Hadoop cluster. J. Comput. Res. Dev. **51**(3), 618–626 (2014)
11. Lin, C.Y., Lin, Y.C.: A load-balancing algorithm for Hadoop distributed file system. In: International Conference on Network-Based Information Systems, pp. 173–179. IEEE (2015)
12. Wei, D., Ibrahim, I., Bassiouni, M.: A new replica placement policy for Hadoop distributed file system. In: International Conference on Big Data Security on Cloud, pp. 262–267. IEEE (2016)
13. Liu, Y., Li, M., Alham, N.K., et al.: Load balancing in MapReduce environments for data intensive applications. In: Eighth International Conference on Fuzzy Systems and Knowledge Discovery, pp. 2675–2678. IEEE (2011)
14. Xie, J., Yin, S., Ruan, X., et al.: Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In: IEEE International Symposium on Parallel and Distributed Processing - Workshop Proceedings, IPDPS 2010, Atlanta, Georgia, USA, 19–23 April 2010, pp. 1–9. DBLP (2010)