http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (23): 7260-7264, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.7260.7264

A Novel Improved Data Load Balancing Algorithm for Hadoop

[1,2]Kun Liu, [1]Gaochao Xu, [2]Tingmei Wang and [2]Jun'e Yuan
[1]College of Computer Science and Technology, Jilin University, Jilin, Changchun, 130012, China
[2]College of Applied Science and Technology, Beijing Union University, Beijing, 102200, China

Abstract: This study is designed to solve the problem of Hadoop data load balancing algorithm. A big difference arises in response time for similar files as it balances the data according to the space usage of each node and doesn't handle the factors like processing power, bandwidth and files' access frequency. This study proposes a novel load balancing model based on the factors of files' size, files' concurrent access time, files' access frequency, nodes' processing power, bandwidth and nodes' available storage space. Experimental results reveal that the new model can not only guarantee the storage space load balancing, but also maintain similar response time for similar files.

Key words: Hadoop, load balancing, cloud computing, cloud storage

INTRODUCTION

As an application program for large data, Hadoop has had many relatively successful applications such as Yahoo and Facebook (Lin, 2012). Hadoop (White, 2009), originating from open source network search engine Apache Nutch, is a widely used file search data developed by Doug Cutting-pache Lucene founder (Chen and Zheng, 2009). It is a distributed systematic framework, under which users are able to develop distributed programs without understanding distributed basic details. It is famous for MapReduce and its distributed file system HDFS; meanwhile, there are many subprojects providing supplementary service (Dean and Ghemawat, 2004).

Due to the random storage of blocks in Hadoop's HDFS and with the adding and deleting of cluster nodes, the cluster is prone to load imbalance (Borthakur, 2008). A program called Balancer exists in Hadoop (Liu and Dong, 2012). Manually calling this program can process load balancing for the whole cluster. Balancer works like, adding up the average storage space utilization rate of Datanodes on all racks; based on the average utilization rate dividing Datanodes into four categories and forming four chain lists which are aboveAvgUtilizedDatanodes, over Utilized Datanodes, below Avg Utilized Datanodes and under Utilized Datanodes, whereby data balancing on the Datanodes take place (Caibinbupt, 2009). Balancing happens within the rack firstly and then spreads between racks.

After a Hadoop cluster has run for some time, the nodes' dynamic adding and deleting can cause data load imbalance. The newly-joining nodes need load balance procession (Lin and Liu, 2012). Good data load balancing strategies are capable of effectively avoiding bottlenecks, such as network load distribution imbalance, data flow congestion, long response time, etc. and raising implementation efficiency. Balancer, a data load balance program provided by HDFS, can balance storage load on each node (Liu *et al*., 2012). However, Hadoop's load balancing algorithm just considers storage space, according to the utilization rate of which carries out load balancing.

IMPROVED HADOOP DATA LOAD BALANCING MODEL

Problem description: In cloud data load balancing, Hadoop only considers the factor of storage space. However, many other factors need considering. For example, when storage utilization rates are equal on Machine A and B, if files on Machine A are accessed frequently and visited by many users at the same time, thus, with the same utilization rate it is obvious that the load on Machine A is heavier than that on Machine B. Take another example. For Machine A and B, if the bandwidth of the net where A stays is broader than that of B, it is clear that A's throughput is larger. Even if the space utilization rates are the same, with better capability A can provide more services. Therefore, the following

Corresponding Author: Gaochao Xu, College of Computer Science and Technology, Jilin University, Jilin, Changchun, 130012, China

model is proposed as a proper data load balancing one in cloud date storage, comprehensively considering all the previous factors.

Analysis of influencing factors : File access (read) times (at): The more one file has been accessed, the more probably it will be accessed again, consequently the heavier the load it occupies is:

- Concurrent file access (read) times (ct): the more one file has been accessed, the heavier the load it occupies is
- File's idle time (i): if one file has not been visited for a long time, its load will gradually decrease
- File size (l): the bigger the size, the heavier the load
- Duration for each access (di): the longer the duration, the heavier the load
- Bandwidth (W): while serving the same amount of data, the broader the bandwidth is, the lighter the load will be
- Current available storage space in servers (L): while serving the same amount of data, the larger the space is, the lighter the load will be

CPU load capacity (C) and memory (M) of servers: while serving the same amount of data, the better the server is, the lighter the load will be.

Related definitions

Definition 1: Load of file i in the server j is accessed for the tth time:

$$e_j(f_i, t) = e_j(fi, t-1) * (1-0^{l}) + l_i * (d_{i1} + d_{i2} + ... + d_{im})/(i*d) \quad (1)$$

- o is a set value
- r is the time difference between this load adjusting and last load adjusting
- j is the number of the server, i is the number of the file and t is the time of access
- e_j is the load of the No. j server
- f_i is the No. j file
- $e_j(f_i, 0) = 0$, meaning when the file is not accessed, the load is 0
- What this formula describes is the load when the file is accessed for the t^{th} time. Only when the file is accessed, can this load be changed
- $e_j(f_i, t-1)*(1-0^{r})$ indicates that if the file has not been accessed, its load will gradually decrease, not fixing on one value, until the value approaches 0. The load decrease is related to the time when it has not been visited. The longer it has not been visited, the larger the load decrease is:

$$l_i * (d_{i1}+d_{i2}+....+d_{im})/(1*d)$$

Illustrates that each visit will increase the file's load. The load increase has to do with file size, concurrent access times and duration for each access. l_i is the size of file i; m is concurrent access times; $d_1, d_2, \ldots\ldots, d_m$ are each concurrent access duration of m times; l' is the average of all file sizes; d' is the average of time all files accessed. Take the ratio of the file size, access time and average access time as the load increase.

The file load will be adjusted according to formula-each time it is accessed.

Definition 2: Load of long-time files which are not accessed for a long time at the time of T:

$$e_j(f_i, t) = e_j(f_i, T) = e_j(f_i, T-\Delta t)*(1-0^{\Delta t}) \quad (2)$$

- t is the time difference. Other parameters refer to Eq.1
- Every other □t automatically adjust each file's load, to decrease the load of files not accessed for a long time. The adjusted file load is taken as the load of time t (the last time)
- This formula only applies to adjusting files not being visited during □t

Definition 3: Total load of server j:

$$E_j = \sum_{i=1}^{a} e_j(f_i, t) \quad (3)$$

Hereinto, n is the number of files in server j.

Definition 4: Relative load of server j:

$$P_j = E_j * (K_w * W^* / W_j + K_1 * L^* / L_j) + K_G * C^* / C_j + K_m * M^* / M_j) \quad (4)$$

- W' is the server's average bandwidth, L' is the available average storage space, C' is CPU's average processing capacity and M' is the average memory
- C_j and M_j are respectively server j's CPU processing capacity and memory, expressed with measure value in this Eq. They are ranked according to CPU and memory's capacity. C' and M' are determined by C_j and M_j, while C_j and M_j are marked by the administrator on the basis of equipment condition
- L_j is the server's available storage space and W_j is sever j's bandwidth
- Equation 4 discloses that the higher the CPU's capacity, the bigger the memory, the broader the bandwidth and the larger the available storage space is, the lighter the relative load will be

- K_w, K_l, K_c and K_m, respectively refer to the coefficient of four determiners, namely the bandwidth, storage space, CPU capacity and memory. $K_w+K_l+K_c+K_m = 1$. The mainly considered factor changes by just altering the relative coefficient, e.g., if the bandwidth is the only factor considered in measuring server's relative load, set $K_w = 1$, $K_l = K_c = K_m = 0$; if the equipment capacity is mainly taken into consideration, set $K_w = 0$, $K_l = K_c = 0.5$, $K_m = 0$

IMPROVED ALGORITHM

The principle of the improved algorithm is constructing two queues as Q and S. The queue Q contains the servers whose load over average load and the queue S contains the servers whose load below average load. We move files from Q to S according a certain regulation. The detailed algorithm is shown in Fig. 1.

EXPERIMENTS

The testing environment consists of three racks as illustrated in Fig. 2, including Rack A, Rack B and Rack C. There is one computer in Rack A, namely A1, three computers in Rack B, namely B1, B2 and B3, while two computers in Rack C, namely C1 and C2. C1 , B1 and A1 are configured as follows: 1.3G CPU, 2G memory and Ubuntu 10.04 OS. C2 and B2 are configured as follows: 2.0 G CPU, 4 G memory and Ubuntu 10.04 OS. B3 is configured as follows: 3.2G CPU, 4 G memory and Ubuntu 10.04 OS. All these computers have a 8G hard disk. Rack A and C have a bandwidth of 10M/S, while Rack B 100M/S. Based on the configurations of each node, their CPU processing capacities are ranked as 3, 3, 4, 5, 3, 4 and memory are ranked as 3, 3, 4, 4, 3, 4. In the definition 4, the values of K_w, K_l, K_c, K_m are 0.4,0,0.3,0.3.

In experiments, A_1 is used as client for storing data, which contains 100 files of 1 M, 100 files of 2 M, 100 files of 3 M, 100 files of 5 M, 100 files of 10 M, 10 files of 15 M, 10 files of 20 M, 10 files of 30 M, 10 files of 40 M, 10 files of 50 M, 4 files of 200 M, 2 files of 500 M and 2 files of 1G. The initial space usage ratio is shown in Table 1. The first replications of all files are stored on A_1, the other two of all respectively on B_1, B_2, B_3, C_1 and C_2. Now the load on each datanode is obviously imbalanced. After storage finished, respectively visit 5 files of all sizes in C_1. Response time situations are revealed by the before-balancing line chart from Fig. 3 to 6. The abscissa refers to the files visited, which are numbered as follows: F1M1 to F1M5 for files of 1M, F2M1 to F2M5 for files of 2M, F3M1 to F3M5 for files of 3M and the like. The ordinate refers to

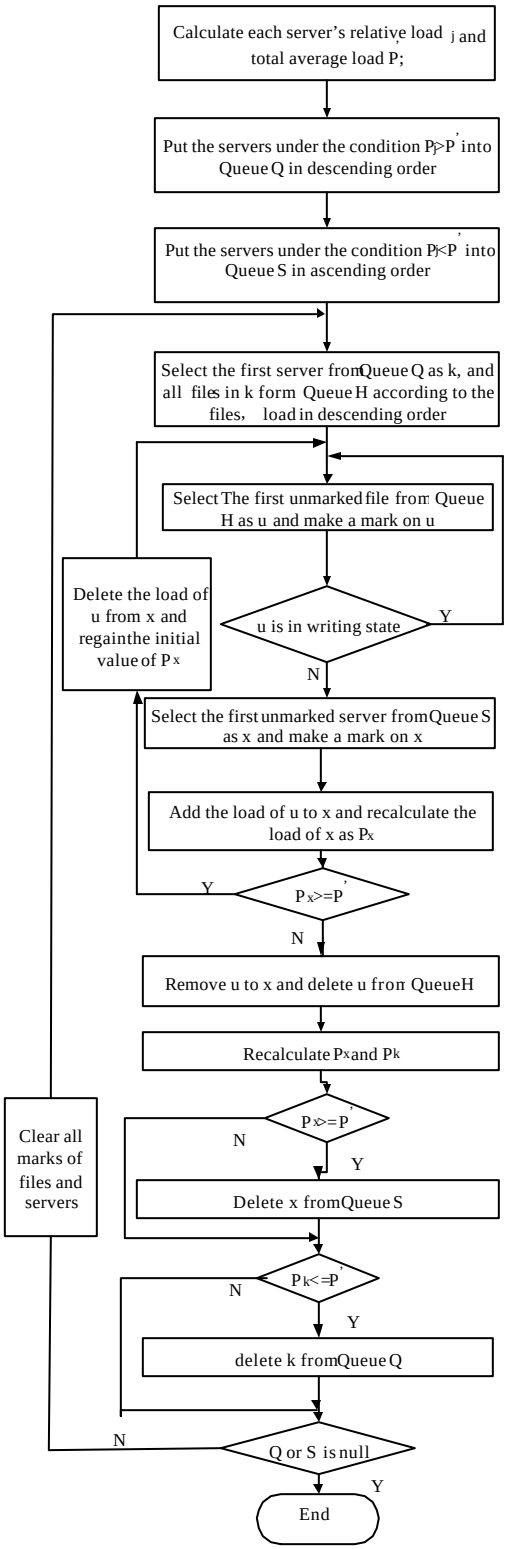

Fig. 1: Flow chart

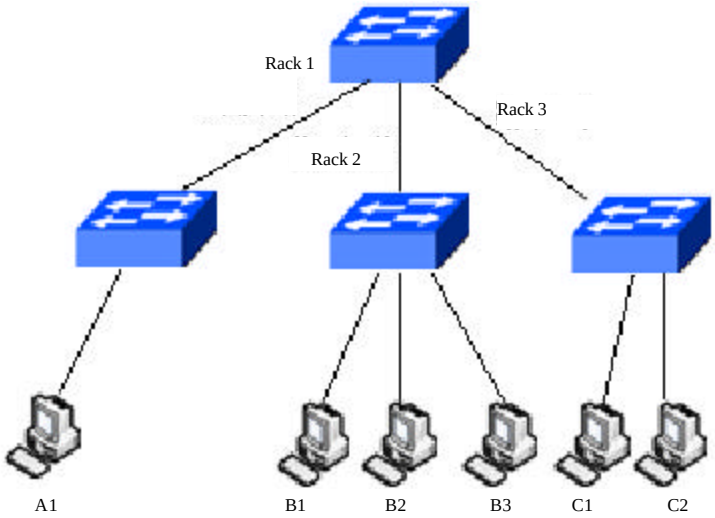

Fig. 2: Experiment topological graph

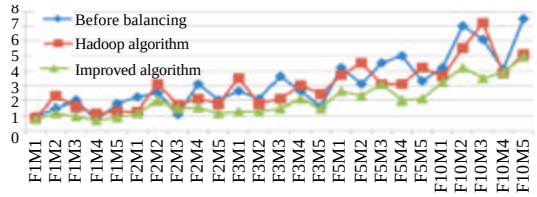

Fig. 3: Response time when accessing files of 1, 2, 3, 5 and 10 m

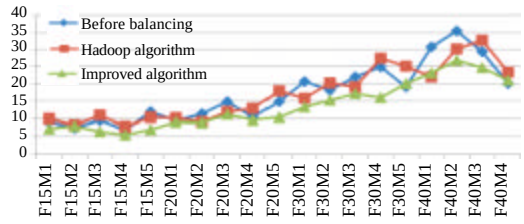

Fig. 4: Response time when accessing files of 15, 20, 30 and 40 m

the response time, whose unit is second. The figures demonstrate that before balancing, the response time varies from very short to very long. Though this has to do with file sizes, small files' response time is not necessarily shorter than that of big files, sometimes even longer, e.g., F1M3's response time longer than that of F2M1, F5M4' response time longer than that of F10M1 and so on. The reason for such a situation is that when visiting these files from datanode C_1, since they are stored on different nodes, response time has to be influenced by such factors as bandwidth, datanode' processing capacity, etc.

Table 1: Initial space usage ratio

		Before hadoop algorithm	
Data node	Hard disk (GB)	Utilized (GB)	Utilization ratio (%)
A1	8.0	7.48	93.50
B1	8.0	4.06	50.75
B2	8.0	4.08	51.00
B3	8.0	4.02	50.25
C1	8.0	1.26	15.75
C2	8.0	1.24	15.50

Table 2: Space usage after hadoop algorithm

		Before hadoop algorithm	
Data node	Hard disk (GB)	Utilized (GB)	Utilization ratio (%)
A1	8.0	4.12	51.50
B1	8.0	4.12	51.50
B2	8.0	4.10	51.25
B3	8.0	4.12	51.50
C1	8.0	2.84	35.50
C2	8.0	2.84	35.50

Employ Hadoop's load balancing algorithm to balance these datanodes. Space usage ratios are shown in Table 2 after load balancing finished, respectively 51.5, 51.5, 51.25, 51.5, 35.5 and 35.5%. Judging from space usage ratio these datanodes definitely have been balanced. When accessing 5 files of all sizes from C1, the response time can be seen in Fig. 3 to 6. We can get the conclusion is that the response time remains basically the same as that before balancing, that is, small files' response time is longer than that of big files in most cases.

Adopt the load balancing algorithm in this study to balance these datanodes. Space usage ratios are shown in Table 3 after load balancing finished, respectively 36.25, 42.5, 55, 59.75, 33.75 and 49.5%. Judging from space usage

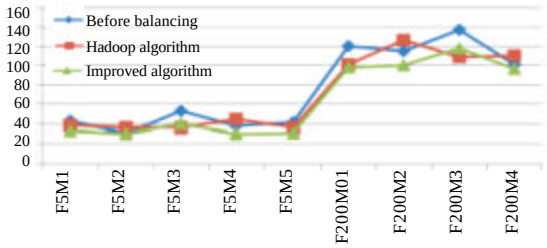

Fig. 5: Response time when accessing files of 50 and 200 m

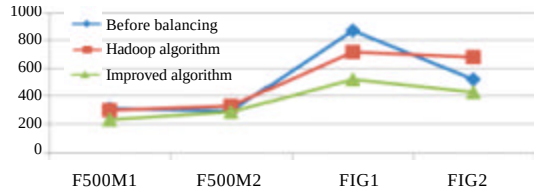

Fig. 6: Response time when accessing files of 500 and 1 g

Table 3: Space usage after this study's algorithm

		This study's algorithm	
Data node	Hard disk (GB)	Utilized (GB)	Utilization ratio (%)
A1	8.0	2.90	36.25
B1	8.0	3.40	42.50
B2	8.0	4.40	55.00
B3	8.0	4.78	59.75
C1	8.0	2.70	33.75
C2	8.0	3.96	49.50

ratio these datanodes definitely have basically balanced loads. When accessing 5 files of all sizes from C1, the response time is reflected by the line chart of after the load balancing algorithm in this study from Fig. 3 to 6. It can be seen that the response time is roughly in accordance with the size of files, that is, small files' response time is short while big files' response time is a little longer.

Comparing the experiment results of two algorithms, we can conclude that this study's algorithm succeeds in guaranteeing that small files' response time is short, which can save small file visitors' waiting time; from the angle of space utilization ratio, although Hadoop algorithm makes the load on each datanode more balanced, yet this study's algorithm can also realize the balance of storage space load.

CONCLUSION

This study analyzed Hadoop load balancing algorithm and proposed an improved model focusing on Hadoop algorithm's limitation that it only handles storage space load balancing. The new model in this study can not only manage storage space load balancing, but also incorporate factors like file size, file visit frequency, datanode's CPU processing capacity, datanode's memoery, bandwidth, etc., which enables the datanodes with better capacity to burden more load, consequently ensuring the consistency of each user's response time. Experiments proved the effectiveness of the algorithm in this study.

ACKNOWLEDGMENTS

The study is subsidized by Beijing City Board of Education Science and Technology Project (SQKM201211417008) and Funding Project of Competence Development Program for Beijing VET Teachers.

REREFENCES

Borthakur, D., 2007. The hadoop distributed file system: Architecture and design. https://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf

Caibinbupt., 2009. Hadoop source code analyses. http://caibinbupt.javaeye.com/blog/318949

Chen, K. and W.M. Zheng, 2009. Cloud computing: System instances and current research. J. Software, 20: 1337-1348.

Dean, J. and S. Ghemawat, 2004. MapReduce:Simplied data processing on large clusters. Proceedings of the 6th Symposium on Operating System Design and Implementation, (OSDI'04), ACM Press, New York, pp:137-150.

Lin, W.W. and B. Liu, 2012. Hadoop data load balancing algorithm based on dynamic broad band distribution. South China Univ. Technol. J. Nat. Sci. Edn., 40: 42-47.

Lin, W.W., 2012. An improved hadoop data placement strategy. South China Univ. Technol. J. Nat. Sci. Edn., 40: 152-157.

Liu, K. and L.J. Dong, 2012. Research on cloud data storage technology and its architecture implementation. Proceedings of the International Workshop on Information and Electronics Engineering, March 10-11, 2012, Harbin, pp:133-137.

Liu, K., L. Xiao and H.Y. Zhao, 2012. Research and improvement of hadoop's cloud load balancing algorithm. J. Microelectron. Comput., 29: 18-22.

White,T., 2009. Hadoop:The Definitive Guide. O'Reilly Media, United States of America, pages: 528.