Performance-Aware Data Placement Policy for Hadoop Distributed File System

Nang Kham Soe
University of Information Technology
Yangon, Myanmar
nangkhamsoe@uit.edu.mm

Tin Tin Yee
University of Information Technology, Yangon, Myanmar
tintinyee@uit.edu.mm

Ei Chaw Htoon
Computer University KyaingTone, Myanmar
htoon.eichaw@gmail.com

Abstract

Apache Hadoop is an open-source software framework for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. The Hadoop Distributed File System (HDFS) is the underlying file system of a Hadoop cluster. The default HDFS data placement strategy works well in homogeneous cluster. But it performs poorly in heterogeneous clusters because of the heterogeneity of the nodes capabilities. It may cause overload in some computing nodes and reduce Hadoop performance. Therefore, Hadoop Distributed File System (HDFS) has to rely on load balancing utility to balance data distribution. As a result, data can be placed evenly across the Hadoop cluster. But it may cause the overhead of transferring unprocessed data from slow nodes to fast nodes because each node has different computing capacity in heterogeneous Hadoop cluster. In order to solve these problems, a data/replica placement policy based on storage utilization and computing capacity of each data node in heterogeneous Hadoop Cluster is proposed. The proposed policy tends to reduce the overload of some computing nodes as well as reduce overhead of data transmission between different computing nodes.

1. Introduction

Hadoop is one of the platforms for big data. Hadoop includes two parts, namely, the MapReduce and Hadoop Distributed File System (HDFS). MapReduce [4] is a software framework to process large data. It includes two phases: Map phase and Reduce phase. Data is split into small parts so that many map tasks can process them simultaneously. The results of map tasks are shuffled and merged by reduce tasks. HDFS is a storage part of Hadoop. HDFS [10] has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

In HDFS, each data file is stored as a sequence of blocks, the blocks of a file are replicated for reading performance and fault tolerance. HDFS uses a uniform triplication policy (i.e. three replicas for each data block) to improve data locality and ensure data availability and fault tolerance in the event of hardware failure [3]. The placement policy of replicas is critical to HDFS performance and reliability. For the common case, the triplication policy in HDFS works well in term of high reliability and high performance. The HDFS Replica Placement Policy (RPP) is a rack-aware policy. The drawback of the policy is that it cannot evenly distribute replicas to cluster nodes. As a result, such data placement policy can noticeably reduce heterogeneous environment performance and may cause increasingly the overhead of transferring unprocessed data from slow nodes to fast nodes.

Data placement problem needs to be considered to obtain an optimal placement solution that balances workload in the cluster. Although HDFS provides a balancing utility to address the issue of unbalanced HDFS cluster, which does not consider on computing capacity of each node. This paper presents data placement policy based on storage and computing capacity to distribute data as even as possible with keeping same rules of existing HDFS RPP. As a result, the proposed policy can reduce overload among the computing nodes and overhead of data transmission between different computing nodes. The proposed policy assigns the data blocks to cluster nodes in accordance with their storage and computing capacity.

2. Background

HDFS [9] is designed to reliably store very large files across machines in a large cluster. It has two kinds of nodes, where a namenode serves as the master node and datanodes as slave nodes. The namenode[5] maintains the metadata of the file

system, which stores the directory structure, file descriptions and a block map which identifies the location of each block replica in the cluster. Each datanode is responsible for storing the actual data blocks on each machine, and handling incoming read and write requests. Each datanode also periodically sends a heartbeat message to the namenode to report machine and block status. The namenode receives such messages because it is the sole decision maker of all replicas in the system. Each application creates a HDFS client to access the file system.

HDFS stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance. The default HDFS replica placement policy is rack-aware. The purpose of this replica placement policy is to improve data reliability, availability, and network bandwidth utilization. For the common case, the replication factor is three by default, the data placement policy is to put one replica on one node in the local rack, another on a different node in the local rack, and the last on a different node in a different rack. Such default data placement strategy assumes that the computing capacity and storage capacity of each node in the cluster is the same [1]. In a heterogeneous environment, the difference in nodes computing capacity may cause load imbalance and overhead of data transmission. The reason is that different computing capacities between nodes cause different task execution time, so the faster nodes finish processing local data blocks earlier than slower nodes do. At this point, the master assigns non-performed tasks to the idle faster nodes, but these nodes do not own the data needed for processing. The required data should be transferred from slow nodes to idle faster nodes through the network. Because of waiting for the data transmission time, the task execution time increases. It causes the entire job execution time to become extended. Moving large number of data affects Hadoop performance.

The rest of the paper is organized as follows. Section 3 describes related works. Section 4 introduces the HDFS RPP and section 5 presents the proposed policy in detail. Performance Experiments are presented in Section 6. Finally, we conclude in Section 7.

3. Related Works

Some data placement policies have been proposed in Hadoop framework for load balancing problem and improve Hadoop Performance in recent years. I. A. Ibrahim and Wei Dai proposed a new replica placement policy for HDFS in [3, 6], which addresses the load balancing issue by evenly distributing replicas to cluster nodes. The policies mainly focus on storage utilization of each node in homogeneous cluster environments where all cluster nodes have the same computing capabilities. Hadoop provides a mechanism to rebalance data manually [8]. Rebalancing is necessary in a Hadoop cluster, as it avoids under-utilization or over-utilization of data nodes. Data rebalancing is done by a rebalancing server. The drawback of this method is that it has to be invoked manually whenever a new data node is added to the Hadoop cluster or when the cluster is imbalanced. C.-W. Lee et al [2] proposed a data placement algorithm to resolve the unbalanced node workload problem. The algorithm is based on different computing capacities of nodes to allocate data blocks, thereby improving data locality and reducing the additional overhead to enhance Hadoop performance. The computing capacity of each node is based on the average execution time of a single task in that node. Y. Fan [7] proposed a method that first computes the nodes computing capability based on the log information about the history tasks. Then data is divided into different sized blocks according to the nodes computing capacity. Further the dynamic data migration policy aims at the transfer of data from slow DataNode to headmost DataNode during execution time. The computing capability is defined as the time cost which is spent to execute a unit size of data.

4. Data/Replica Placement in HDFS

When a client writes the data file into HDFS, this file is split into data blocks. The HDFS client first asks the namenode to update the metadata. The namenode responds with a write permission indicating the datanode on which the blocks of the file should be written. Number of datanodes depends on the replication factor of that file. By default replication factor is three in HDFS, therefore location of three datanodes are returned by namenode. Namenode chooses location of datanodes with default data/replica placement policy. The policy puts the first replica on data node – either local node or on a random node depending upon the HDFS client running in the cluster. For the second replica, it puts on a different node in the rack where the first replica is placed. For the third replica, it puts on the node in different rack. The data placement process is complete after all replicas of the blocks have been written as depicted in Figure 1.

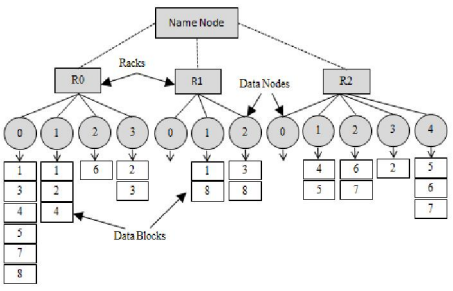

Figure 1 . Default replica placement policy

5. Proposed Data/Replica Placement Policy

In a heterogeneous hadoop cluster, the storage and the computing capacity of each data node are not the same. Therefore, the proposed data placement policy first calculates storage utilization, computing capacity of each node in the cluster and stores in nodeStatus table. When the Hadoop startup, the nodeStatus table is created in NameNode using algorithm 1 and updated it every time slot. The proposed system uses the number of finished tasks of each data node as computing capacity.

As presented in algorithm 2, when data are written into the HDFS, the proposed system splits the data to the data blocks according to user requested size and replicate the data blocks. Then, the proposed policy calculates average storage utilization of each rack and chooses the two racks that have more free space in the cluster. After rack selection, the proposed policy chooses the data nodes which free space are greater or equal to the average storage utilization of the rack. After node selection, the policy distributes two third of data in the freest rack and a third of data in the second freest rack. The newly written data will be allocated to each node in accordance with the computing capacity that record in nodeStatus table. Figure 2 shows the flow chart of proposed data placement policy.

Table 1. Notations used in proposed data placement policy

Notation	Description
D_S (x)	Storage utilization of x^{th} data node.
FreeSpace(x)	The size of free disk space on x^{th} data node.
TotalSpace(x)	The size of disk space on x^{th} data node.
D_C (x)	Computing capacity of x^{th} data node.
finishTask(t)	The number of finished map tasks at a specific timeslot.
DN	Number of data node in each rack.
$AVG_S(r)$	Average storage utilization of r^{th} rack of the cluster where r={0,1,2,…m-1}, where m is the number of racks in the cluster.
targetNode	An array holds the selected rack, the selected data nodes on which the blocks of the file should be written and the number of data blocks that can be placed.
numOfBlock	Number of data blocks need to be distributed in each rack.
availBlock	Number of data blocks can be distributed to each selected nodes with their computing capacity.

Algorithm 1. Making nodeStatus Table

1. **For** each node **do**
2. Calculate the available storage space of the node

$$D_S(x) = \left\lceil \frac{\text{FreeSpace(x)}}{\text{TotalSpace(x)}} \right\rceil \%.$$

3. Calculate node's computing capacity

$$D_C(x) = \text{finishTask(t)}$$

4. Fill nodeID, rackID, storage utilization and computing capacity in the nodeStatus Table.
5. **End For**

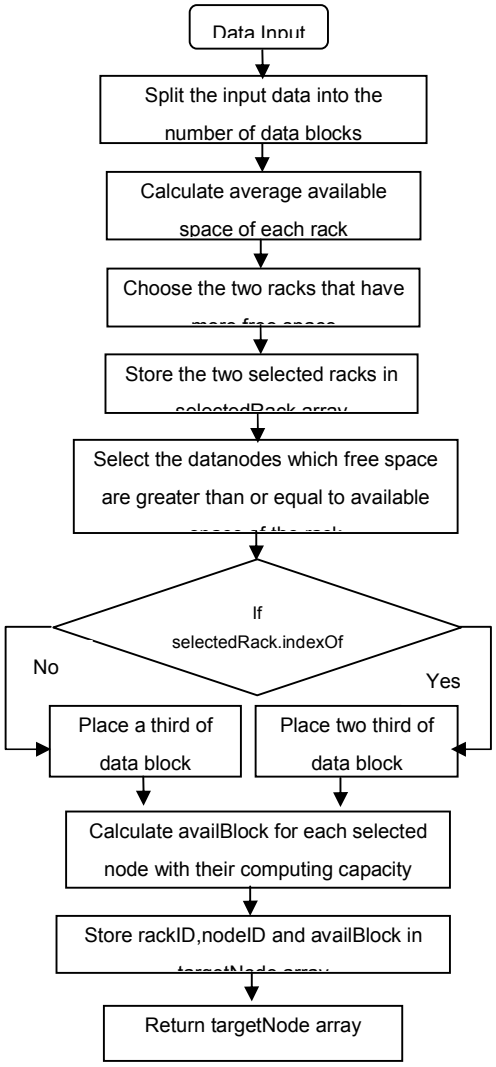

Figure 2. Flow chart of proposed data placement policy

The proposed system uses Tasks API to get the number of finished map tasks based on timeslot duration. Then the required values are extracted such as id, type and state of each task object in tasks array.

To implement the system, a new method will be created in BlockPlacementPolicyDefault.java file. This file is in org.apache.hadoop.hdfs.server. namenode package. The method is responsible for retrieving three nodeID(the first two nodeID with the same rackID and one nodeID with the different rackID) from the targetNode Array to place block replicas.

Algorithm 2: Proposed Data Placement Policy

Input: Data, nodeStatus Table

Output: targetNode Array

1. When a data is written into HDFS
2. DataSize←obtain from data information
3. BlockSize←set by user
4. TotalBlockNumber ← $\left\lceil \frac{DataSize}{BlockSize} \right\rceil$
5. **For** each rack **do**
6. Calculate $AVG_S(r) = \sum_{x=0}^{DN-1} D_S(x) / DN$
7. **End for.**
8. Get the two racks that are more free space in the cluster and store selectedRack array.
9. **For** each rack **in** selectedRack **do**
10. Select the datanodes which free space are exceed average free storage capacity of the rack.
11. Store it in selectedNodes array.
12. **For** each node **in** selectedNodes **do**
13. **If** selectedRack.indexOf(rack)==0 **then**
14. *numOfBlock*←*2/3* * TotalBlockNumber
15. **Else**
16. *numOfBlock*←*1/3* * TotalBlockNumber
17. **End If.**
18. availBlock← $\frac{Dc(x)}{\sum_{x=0}^{selectedNode-1} Dc(x)}$ * *numOfBlock*
19. Store rackID, nodeID and availBlock in targetNode array.
20. **End for.**
23. **End for.**
24: **return** targetNode

6. Performance Experiments

To test the distribution of the replicas, theoretical simulation is shown in Figure 3, 4 and Table 2, 3 and 4. Assume free space and the number of finished map tasks of each data node as shown in Table 2. To get computing capacity, the proposed system evaluates the number of finished tasks of each data node by Tasks API based on a specific time slot. Average processing time for a single map tasks may be between 5sec to 7sec depend on the nature of computing nodes. Therefore, the probability of the number of finished map tasks are calculated for the specific timeslots(10sec, 30sec and 50sec) as depicted in table 2.

Table 2. Probability of the number of finished map tasks are calculated for the specific timeslots

Timeslot 1 (10sec)	Timeslot 2 (30sec)	Timeslot 3 (50sec)
2	6	10
2	5	8
1	4	7
2	7	12
2	5	8
1	2	3
1	4	7
4	11	18
1	4	7
2	8	13
3	9	15
2	5	8

According to the result of table 2, a specific timeslot (10sec) is not suitable for the simulation because the probabilities of the number of finished map tasks are not significantly different. And a specific timeslot (50sec) is also not suitable because the nodeStatus table can be out of date for long time. Therefore, assume the time slot duration is 30 seconds for this simulation to get the number of finished map tasks. In the time slot, the number of successful finished map tasks is counted and stored for individual nodes in nodeStatus table according to algorithm 1 as shown in table 3.

Table 3. Sample nodeStatus table

rack ID	node ID	Available space (%) (D_S (x))	Computing Capacity D_C (x)
R0	0	50	6
	1	44	5
	2	30	4
	3	65	7
R1	0	34	5
	1	20	2
	2	29	4
R2	0	66	11
	1	68	4
	2	74	8
	3	55	9
	4	47	5

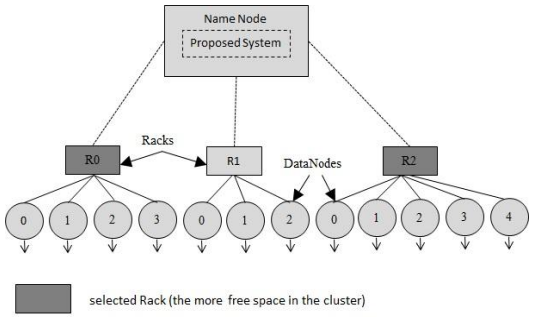

Figure 3. Choosing the two racks based on available storage space

As shown in figure 3, when data are written into the HDFS, the proposed policy calculates average free space of each rack. The two racks (R0 and R2) are chosen based on storage utilization. In each selected rack, chooses the data nodes ((node 0 and 3) in R0 and (node 0, 1, 2) in R2) because which free space are greater than average free space of these racks as shown in figure 4.

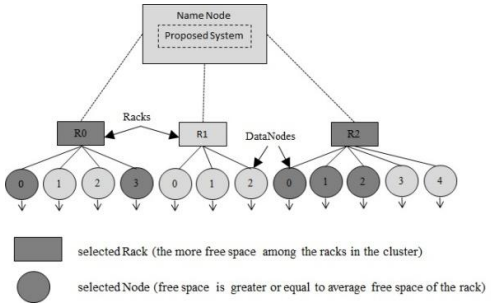

Figure 4. Choosing the data nodes.

If there are data to be written into the HDFS, assume that these data can be partitioned into 32 data blocks and is replicated these blocks with 3 times. Therefore, 96 data blocks need to be distributed in the selected racks. According to the proposed data placement policy, 64 data blocks are placed in the freest rack (R2) and 32 blocks in the second freest rack (R0). Finally, the policy calculates the number of data blocks can be placed on each node according to its computing capacity as shown in Table 4. Therefore, in the rack R2, node 0 is assigned [64 $*\frac{11}{11+4+8} = 31$] data blocks, node 1 is assigned [64 $*\frac{4}{11+4+8} = 11$] data blocks and node 2 is assigned [64 $*\frac{8}{11+4+8} = 22$] data blocks. In the rack R0, node 0 is assigned [32 $*\frac{6}{6+7} = 15$] data blocks and node 3 is assigned [32 $*\frac{7}{6+7} = 17$] data blocks. By using the values of targetNode array, the first and second replicas of each data block of the input file will be placed on two different nodes in the first selected rack and the third replica will be placed on a node in the different rack.

According to our simulations, the proposed policy will reduce overload among the computing nodes because the policy takes into account data nodes' available storage space. Moreover, it will reduce data transmission overhead in task execution time because the number of data blocks on each node is proportional to its computing power.

Table 4. Output of proposed system (targetNode array)

Index	rackID	nodeID	availBlock
0	R2	0	31
1		1	11
2		2	22
3	R0	0	15
4		3	17

In the future, many experiments have to be done in order to get the efficiency of proposed data placement policy based on reasonable time slots.

7. Conclusion

The default data placement strategy assumes that the computing capacity and storage capacity of each node in the cluster is the same. The HDFS Replica Placement Policy cannot evenly distribute replicas to cluster nodes. As a result, such data placement policy can noticeably reduce heterogeneous environment performance and may cause increasingly the overhead of transferring unprocessed data from slow nodes to fast nodes. Therefore, a data placement strategy based on storage utilization and computing capacity of each node is proposed. By considering the storage utilization of each data node for data placement, it will reduce overload problem among the data nodes. Moreover, it will be reduced the overhead of data transmission from the slow node to the fast node during execution time by selecting data nodes based on computing capacity. As a result, the proposed paper will improve the performance of Hadoop. The proposed policy will be implemented and tested in simulated hadoop environment as the future work.

References

[1] A. Sharafi, A. Rezaee, "Adaptive Dynamic Data Placement Algorithm for Hadoop Heterogenous Environment", JACET Journal of Advance in Computer Engineering and Technology,2(4) 2016.

[2] C.-W. Lee et al., A Dynamic Data Placement Strategy for Hadoop in Heterogroneous Environments, Big Data Research(2014), http://dx.doi.org/10.1016/ j.bdr.2014.07.002

[3] I. A. Ibrahim*, W. Dai *, Mostafa Bassiouni, "Intelligent Data Placement Mechanism for Replicas Distribution in Cloud Storage

Systems", IEEE International Conference on Smart Cloud, 2016.
[4] J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", in Proceedings of Sixth Symposium on Operating System Design and Implementation (OSDI'04), San Francisco, CA, December 2004.
[5] Q. Zhang, S. Q. Zhang∗, A. Leon−Garcia∗, R. Boutaba ," Aurora: Adaptive Block Replication in DistributedFile Systems", IEEE 35th International Conference on Distributed Computing Systems, 2015.
[6] W. Dai *, I. Ibrahim*, M. Bassiouni, "A New Replica Placement Policy for Hadoop Distributed File System, IEEE 2nd International Conference on Big Data Security on Cloud, IEEE International Conference on High Performance and Smart Computing, 2016 IEEE International Conference on Intelligent Data and Security
[7] Y. Fan, W. Wu, H. Cao, H. Zhu, X. Zhao, W. Wei, "A heterogeneity-aware data distribution and rebalance method in Hadoop cluster", 2012 Seventh ChinaGrid Annual Conference.
[8] Hadoop Data Balancer Administrator Guide, (01-202014), https://issues.apache.org/jira/secure/attachment/12369201/BalancerAdminGuide.pdf
[9] https://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html #Data_Replication
[10] https://hadoop.apache.org/docs/r1.2.1/hdfs_design .html