Resource-based Data Placement Strategy for Hadoop Distributed File System

Nang Kham Soe, Tin Tin Yee, Ei Chaw Htoon
University of Information Technology, Yangon, Myanmar
nangkhamsoe@uit.edu.mm, tintinyee@uit.edu.mm, htoon.eichaw@gmail.com

Abstract

Big-Data is a term for data sets that are so large or complex that traditional data processing tools are inadequate to process or manage them. 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 default Hadoop data placement strategy works well in homogeneous cluster. But it performs poorly in heterogeneous clusters because of the heterogeneity (in terms of processing, memory, throughput, I/O, etc.) of the nodes capabilities. It may cause load imbalance and reduce Hadoop performance. Therefore, Hadoop Distributed File System (HDFS) has to rely on load balancing utility to balance data distribution. The utility consumes the cost of extra system resources and running time. 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 algorithm based on storage utilization and computing capacity of each data node in heterogeneous Hadoop Cluster is proposed. The proposed policy can balance the workload as well as reduce overhead of data transmission between different computing nodes.*

Keywords- HDFS, Data Placement Policy, Load Balancing.

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. It has two kinds of nodes, where a namenode serves as the master node and datanodes as slave nodes.

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 utilization and computing capacity to distribute data as even as possible with keeping same rules of existing HDFS RPP. As a result, there is no need to run balancer tool and reduce 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 [8] 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-awareness. 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 presents the proposed policy in detail. Expected results are presented in Section 5, and we conclude in Section 6.

3. Related Works

Some data placement policies have been proposed in Hadoop framework for load balancing problem in recent years. In [3, 6], a new replica placement policy is proposed for HDFS, 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 [9]. 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. The researchers in [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. Reference [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 user submits a job to Hadoop for some required process, it needs to specify the location of input as well as output files in HDFS. As the client write file, this file is split into data blocks. To handle this process, a 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. Then the list of datanodes forms a pipeline. The client then writes the data block onto the first datanode in the pipeline and forwards it to the second datanode. Similarly, after the block is stored in the second datanode, forwards it to the third datanode. The data placement process is complete after all replicas of the blocks have been written as depicted in Figure 1.

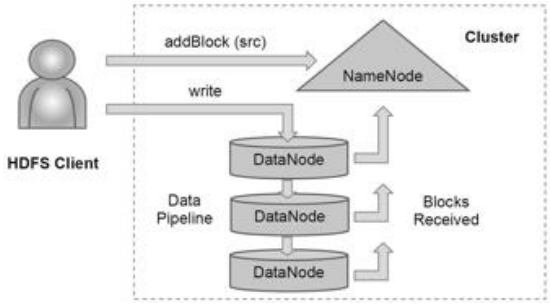

Figure 1. Data Placement in HDFS

4.1. Proposed Replica/Data Placement Policy

In a heterogeneous cluster, the storage and the computing capacity for each 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 everytime NameNode receives heartbeat message from DataNodes. The proposed system uses the number of finished tasks of each data node for a given time interval as computing capacity.

-To calculate storage utilization of the k^{th} DataNode SD(k), uses formula (1)

$$SD(k) = \frac{SU(k)}{ST(k)} \% \quad (1)$$

(SU(k) denotes used space of the k^{th} DataNode and ST(k) denotes total storage capacity of the k^{th} DataNode).

-To calculate average storage utilization of the j^{th} rack SR(j), uses formula (2)

$$SR(j) = \frac{\sum_{k=0}^{DN-1} SD(k)}{DN} \quad (2)$$

(DN denotes the number of available data nodes of the j^{th} rack).

-To get the least storage utilization of the rack sui_Rack, uses formula (3)

Min (SR(j), SR(j+1), ….., SR(r-1)) (3)

(r denotes the number of racks in the cluster).

Table 1. Notations used in Proposed Data Placement Policy

Notation	Description
RF	Replication factor of the file. (maximum replication factor is 3 in this system.)
TargetNode	A array holds the target data nodes.
LocalNode	The DataNode where the client initiates a write
highestComputingNode	The DataNode has the maximum number of finished task for a given time interval in the rack.
nodeFlag	It is to be used to know the node is whether in local or remote rack.
nodeStatus Table	The table stores storage utilization, computing capacity of each node in the cluster
reqBlocks	The number of blocks for requested file to be distributed.
sui_Nodes	An array holds the datanodes which storage utilization are less than SR(j)
sui_Rack	the least storage utilization of the rack in the cluster

Algorithm 1: Making nodeStatus Table

Step 1: **For** each node **do**

Step 1.1: Calculate storage utilization SD(k) using formula (1).

Step 1.2: Get the computing capacity of each node by retrieving the number of finished tasks at a given time interval.

Step 1.3: Fill storage utilization and computing capacity in the nodeStatus Table.

End for.

Algorithm 2: Proposed Data Placement Policy

Input: RF, reqBlocks

Output: targetNode[reqBlocks][RF]

Step 1: nodeFlag=0

Step 2: **For** each block **do**

Step 2.1: **If** targetNode.size==0 **then**

Step 2.1.1: Choose localNode.

Step 2.1.2: targetNode.add(localNode).

End If

Step 2.2: **while** targetNode.size<RF

Step 2.2.1: **If** nodeFlag==1 **then**

Find SR(j) for the remote rack using formula (2).

//to calculate SR(j), get storage utilization of each datanode from nodeStatus Table.

Step 2.2.1.1: **For** each DataNode **do**

If SD(k) < SR(j) **then**

sui_Nodes[] = DN(k)

End If

End For

Step 2.2.1.2: Choose the highestComputingNode from sui_Nodes[].

//get computing capacity of each datanode from nodeStatus Table.

End If

Else

Step 2.2.2: Find SR(j) for the local rack using formula (2).

Step 2.2.2.1: *//to calculate SR(j), get storage utilization of each datanode from nodeStatus Table*.

For each DataNode **do**

If SD(k) < SR(j) **then**

sui_Nodes[] = DN(k)

End If

End For

Step 2.2.2.2: Choose the highestComputingNode from sui_Nodes[].

//get workload of each datanode from nodeStatus Table.

If nodeFlag==0 **then** Set nodeFlag to 1.

End If

Step 2.2.2.3: **End If**

targetNode.add(highestComputingNode).

End while

End For

Step 2.2.2.4:

Step 2.2.3:

Step 3: return targetNode.

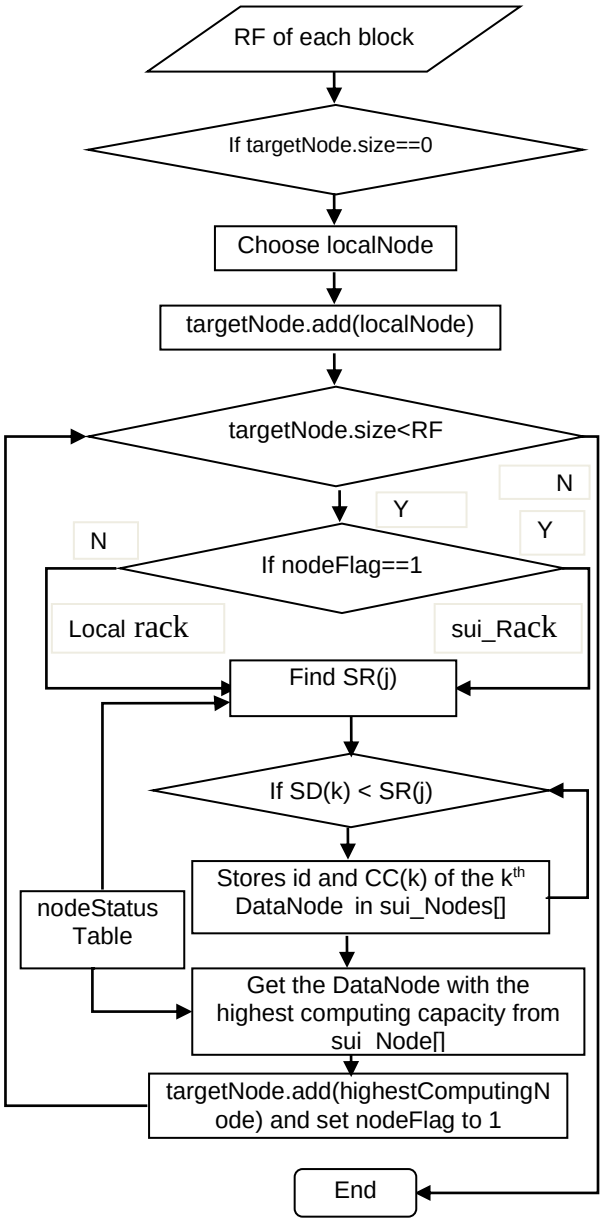

Figure 2. Flow Chat of Proposed Data Placement Policy

6. 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 computing capacity of each node is proposed. By considering the storage utilization of each data node for data placement, it can reduce load unbalance problem. Moreover, it can reduce the overhead of data transmission from the slow node to the fast node during execution time by placing data based on computing capacity. As a result, the proposed paper can improve the performance of Hadoop.

7. References

[1] Avishan Sharafi, Ali 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]Ibrahim Adel Ibrahim*, Wei 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, Sai Qian Zhang∗, Alberto Leon−Garcia∗, Raouf Boutaba ,” Aurora: Adaptive Block Replication in DistributedFile Systems”, IEEE 35th International Conference on Distributed Computing Systems, 2015.

[6] Wei Dai *, Ibrahim Ibrahim*, Mostafa 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] https://hadoop.apache.org/docs/r2.8.0/hadoop-projectdist/hadoop-hdfs/HdfsDesign.html#Data_Replication

[9] Hadoop Data Balancer Administrator Guide, (01-20 2014),https://issues.apache.org/jira/secure/attachment/12369201/BalancerAdminGuide.pdf