Improved Input Data Splitting in MapReduce

Reema Rhine
PG student, Department of Information Technology
Rajagiri School of Engineering and Technology,
Kochi, India

Nikhila T. Bhuvan
Asst. Professor, Department of Information Technology Rajagiri School of Engineering and Technology, Kochi, India

ABSTRACT
The performance of MapReduce greatly depends on its data splitting process which happens before the map phase. This is usually done using naive methods which are not at all optimal. In this paper, an Improved Input Splitting technology based on locality is explained which aims at addressing the input data splitting problems which affects the job performance seriously. Improved Input Splitting clusters data blocks from a same node into the same single partition, so that it is processed by one map task. This method avoids the time for slot reallocation and multiple tasks initializing. Experiment results demonstrated that this can improve the MapReduce processing performance largely than the traditional Hadoop implementation.

General Terms
Big data, Apache Hadoop.

Keywords
HDFS, Improved input splitting, MapReduce

1. INTRODUCTION
Millions of bytes of data are generated daily, according to IBM. These data are from social networks, Internet searches, e-commerce transactions etc. This collection of data forms a large amount of big data that is growing on a daily basis. It is required to analyze this large amount of data and extract the useful information from it for future use in a short time. Big data is a popular term used to describe the exponential growth and availability of data, both structured and unstructured. More accurate analysis may lead to much better decision making. This can mean greater operational efficiencies, cost reductions and reduced risk. The three Vs of big data are: Volume, Velocity and Variety. Two additional parameters to be considered for big data are variability and veracity.Apache Hadoop[9] is an open source software library. It allows the distributed processing of large data sets across clusters of computers using simple programming models. It has a variety of options ranging from single computer to thousands of computers, each of which offering local computation and storage.

Apache Hadoop includes Hadoop core, Hadoop Distributed File System, and Hadoop MapReduce. Requirements submitted by user to Hadoop engine will take input data from HDFS. Data is spread across a number of DataNodes. There is one NameNode or JobTracker which is responsible for assigning the work among DataNodes and producing the result and responding back to user. Architecture of Apache Hadoop[14] is very robust and fault-tolerant. JobTracker is continuously tracing the status of DataNode and if the DataNode remains silent for more than predefined time, task of that DataNode is given to another DataNode.

According to the MapReduce[2] scheduling mechanism, data splitting controls the data processing parallelism. So it has a critical influence on the job completion time and cluster utilization. Here, an Improved Input Splitting mechanism based on locality[1] is introduced to cluster blocks located at the same machine to a split and schedule a map task onto the node to process it. This avoids the wastage of time for slot reallocation and task initialization. The replicas[4] are chosen based on the peer cooperation and resource utilization .This is done by recording and monitoring the nodes and network utilization. In this paper, the section 2 describes the details of HDFS, MapReduce and input splitting. Section 3 deals with the Hadoop data splitting method and Section 4 deals with the Improved Input Splitting method. Section 5 evaluates both the methods.

2. BACKGROUND
HDFS and MapReduce[13] are the two main components of Apache Hadoop. HDFS handles the storage aspects while the MapReduce handles the programming aspects of Hadoop. Thus, the storage system is not physically separate from a processing system.

2.1 Hadoop Distributed File System
HDFS has a master/slave architecture. An HDFS cluster has a single NameNode and a number of Datanodes. The NameNode acts as the master server that manages the file system namespace. It also regulates the access to files by clients. The DataNodes, usually one per node in the cluster, is responsible for managing the storage attached to the nodes that they run on. A file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes operations like opening files, closing files, and renaming files. The DataNodes are responsible for handling read and write requests from the clients of the file system. The DataNodes also perform block creation, deletion, and replication according to the instruction from the NameNode. HDFS is built using the Java. The NameNode and the DataNode software can be run on any machine that supports Java. Usage of Java language enables HDFS to be deployed on a wide range of machines. A usual deployment has a dedicated machine that runs the NameNode. Each of the other machines in the cluster runs one instance of the DataNode. The NameNode is the repository for all HDFS metadata.

2.2 MapReduce
A MapReduce[7] program consist of the Map function which performs mapping that includes filtering and sorting and also the Reduce function which performs the reduction operation that give the final output. There is also a shuffle operation that occurs between the map and reduce phase which improves the overall performance of the system.

MapReduce gives the user an opportunity to operate on every record in the data set individually, during the map phase. This phase is used to filter out unwanted fields or transform fields. Certain types of joins and grouping can also be done in the map. It is not necessary that there is an output record for every input record. Maps can choose to remove records or explode one record into multiple records. During the shuffle phase, MapReduce partitions data among the various reducers. The input to the reduce phase is each key from the shuffle along with all of the records associated with that particular key. Because all records with the same key are now collected together, it is possible to perform joins and aggregation operations. The MapReduce user explicitly controls parallelism in the reduce phase. MapReduce jobs that do not require a reduce phase can set the reduce count to zero. These are called as map-only jobs.

2.3 Input Splitting

MapReduce is the framework for running jobs in Hadoop. It provides a simple and powerful mechanism for parallelizing data processing. The JobTracker is the central coordinator of jobs in MapReduce. It controls the jobs those are running, the resources they are using, etc. Each node in the cluster has a TaskTracker that is responsible for running the map or reduce tasks assigned to it by the JobTrackerInput split[10] is the part of the input that is processed by a single map task. It can also be referred to as the data to be processed by an individual mapper. Each split is further divided into records, and the map task processes each record, which is a key value pair. Split is actually the number of rows and record is that number. The length of the InputSplit[11] is measured in bytes. Every InputSplit has a storage location. The storage locations are used by the MapReduce system to place map tasks as close to split's data as possible. Usually the tasks are processed in the order of the size of the splits, the largest one get processed first. This is done in order to minimize the job runtime. InputSplit doesn't contain input data but only a reference to the input data. One need not use InputSplits directly, InputFormat will do that job. An InputFormat is a class selects the files or other objects that should be used for input. It then defines the InputSplits that break a file into tasks. It also provides a factory for RecordReader objects that read the file. The client calculates the splits for the job by calling getSplits function. These splits are the send to the JobTracker, which then uses their storage locations to schedule map tasks that will process them on the TaskTrackers. On a TaskTracker, the map task passes the split to the createRecordReader method on InputFormat to obtain a RecordReader for that split. Map task uses RecordReader to generate record key-value pairs, which it passes to the map function.The minimum split size is usually 1 byte, although some formats allow a lower bound on the split size. We can impose a minimum split size. By setting the split size to a value larger than the block size, they force splits to be larger than a block. But this is not a good method while using HDFS, as it will increase the number of blocks that are not local to a map task. The maximum split size is by default the maximum value that can be represented by a Java long type. It has an effect only when it is less than the block size, which will force splits to be smaller than a block.

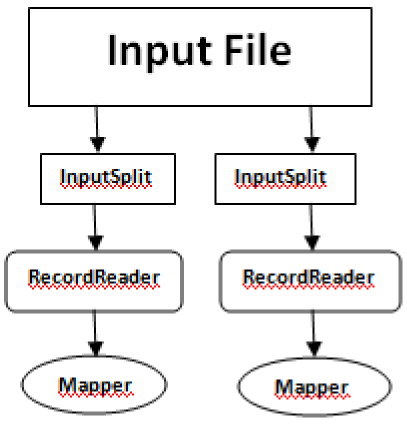


Fig 1:Input Splitting

3. HADOOP DATA SPLITTING

Hadoop consist of two main components: a cluster file system HDFS and an open source implementation of MapReduce. HDFS stores the data while MapReduce runs programs in parallel on top of it. HDFS is designed maintain load balance by storing data uniformly across nodes. However, in practice, its balance is not well for each file distribution. Partition of these blocks are in-practical in some way by hash-based partitioning.

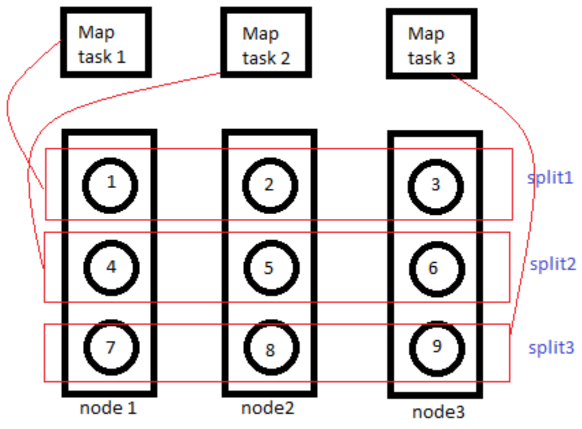


Fig 2: Hash based partition will split blocks cross-node into a split causing network transferring.

Another drawback of hash-based splitting is that it may split block across nodes into the same partition. This will result in non-local map tasks reading data across nodes from network. This consumes a lot of network resources. Also the job completion time is also high.One job’s improper partitioning may result multiple jobs to interfere with each other for sequenced job. MapReduce jobs have a sequence of dependent phases like map/combine/reduce, and each phase compose of several tasks running in parallel.MapReduce client sets the input file and split configuration. This is then submitted to the MapReduce master(JobTracker). When a MapReduce worker(TaskTracker) have a idle slot, it reports to the JobTracker. JobTracker receive the request and create a task which will be placed to the TaskTracker. One map task for each block will cause too many tasks wasting time for slot reallocation and task initialization.

4. IMPROVED INPUT SPLITTING

Improved Input Splitting clusters data blocks co-located in the same node, to address the problems of original splitting method of Hadoop implementation and improve the MapReduce performance. Input data splitting is processed before the Map phase. Here it is extended to include the Improved Input Splitting mechanism based on locality[5]. Improved Input Splitting mechanism consults to the resource masters for slots utilization details. It also asks the DFS master for the location of the replicas of data blocks and then clusters the suitable blocks into the respective partitions.

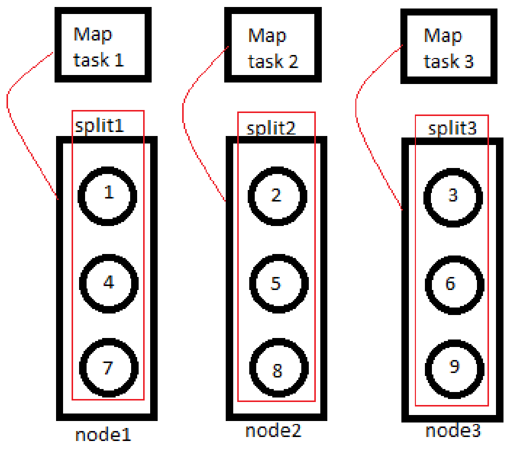
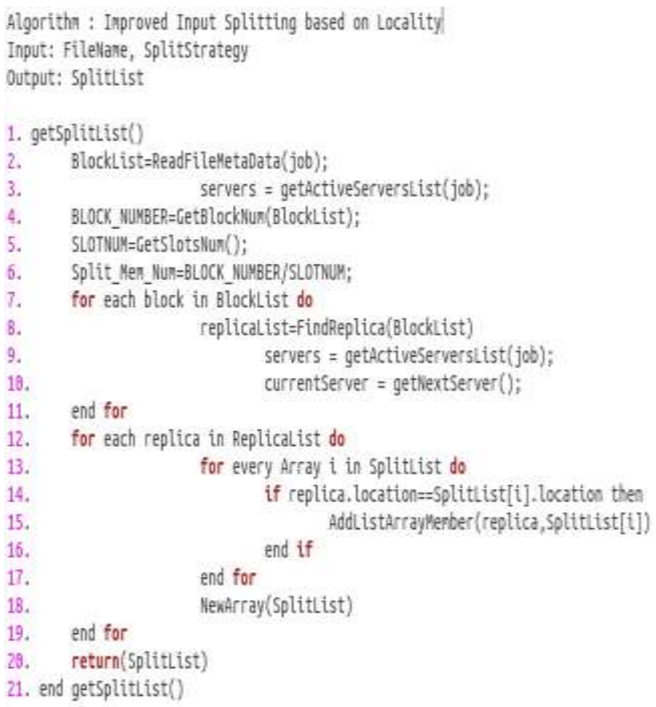


Fig 3: Improved Input Splitting Strategy co-locates the same node into one partition

Here data is split into multiple nodes. Also there are multiple replicas available for each split. To improve the overall performance, the appropriate replica is to be chosen based on the locality factor.The DataNodes send heart-beat messages to the NameNode to keep in touch. Through these messages it is possible to understand their status. The JobTracker is consulted for the corresponding nodes utilization to decide which node replica we choose. So the data for the map task is chosen based on the location of the data replicas. If a particular replica is chosen, then all the data of that node is given to the local map task. This avoids non local reading of data by the map task. This avoids network problems and also improves the overall processing time.SplitList is an array list. Here each array contain the blocks of the same partition, and SplitMemNum is the arrays in this list. In step 8 we choose the proper replica for every block. Then we cluster the replicas into partitions based on its locality. We do this by checking the location of a replica. If it exist in the SplitList, it is added to the location partition, if not then insert a new list into SplitList of this new location. HDFS master is consulted for file mate-data information including file blocks, numbers of block replica and the replicas' location etc. The MapReduce master is monitored for slot utilization.



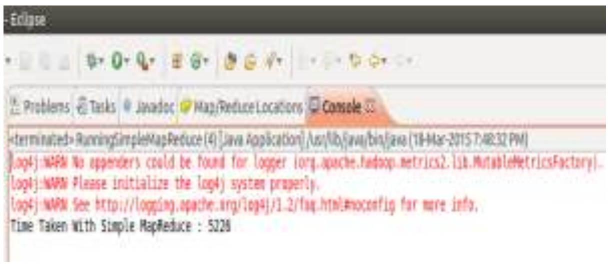
```
Algorithm : Improved Input Splitting based on Locality
Input: FileName, SplitStrategy
Output: SplitList

1. getSplitList()
2.      BlockList=ReadFileMetaData(job);
3.                     servers = getActiveServersList(job);
4.      BLOCK_NUMBER=GetBlockNum(BlockList);
5.      SLOTNUM=GetSlotsNum();
6.      Split_Mem_Num=BLOCK_NUMBER/SLOTNUM;
7.      for each block in BlockList do
8.                     replicaList=FindReplica(BlockList)
9.                               servers = getActiveServersList(job);
10.                              currentServer = getNextServer();
11.     end for
12.     for each replica in ReplicaList do
13.                    for every Array i in SplitList do
14.                              if replica.location==SplitList[i].location then
15.                                        AddListArrayMember(replica,SplitList[i])
16.                              end if
17.                    end for
18.                    NewArray(SplitList)
19.     end for
20.     return(SplitList)
21. end getSplitList()
```

Alg 1: Improved Input Splitting based on Locality

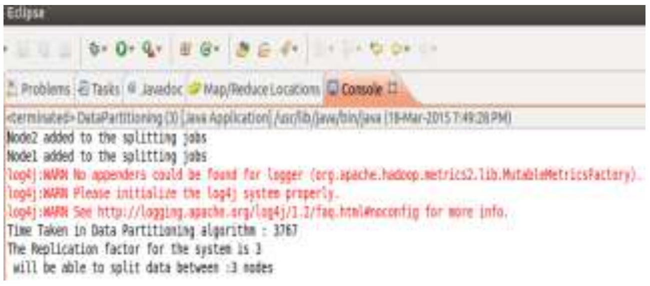
5. EVALUATION

WordCount program is used to compare the MapReduce performance with and without the Improved Input Splitting method. The execution of the WordCount program using the inbuilt class is shown in figure 4. The total time taken for the execution is 5226ms.Figure 5 shows the result of the execution of the WordCount program using the extended class which implements the Improved Input Splitting. The time taken for the execution of the program in this case is 3767ms.Figure 6 is the comparison of the two methods, using the built in class and using the Improved Input Splitting method. It is clearly visible that the Improved Input Splitting method performs better than the default method.



```
<terminated> RunningSimpleMapReduce (4) [Java Application] /usr/lib/java/bin/java (18-Mar-2015 7:48:32 PM)
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Time Taken With Simple MapReduce : 5226
```

Figure 4: Execution of WordCount Program using inbuilt class



```
<terminated> DataPartitioning (3) [Java Application] /usr/lib/java/bin/java (18-Mar-2015 7:49:28 PM)
Node2 added to the splitting jobs
Node1 added to the splitting jobs
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Time Taken in Data Partitioning algorithm : 3767
The Replication factor for the system is 3
 will be able to split data between :3 nodes
```

Figure 5: Execution of WordCount Program using Improved Input Splitting mechanism

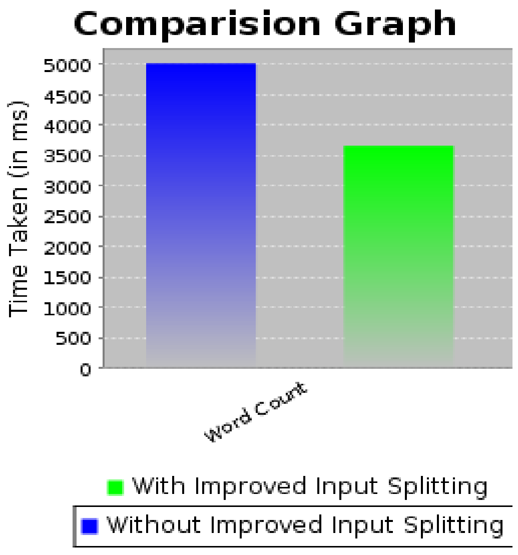


Figure 6: Comparison of the two methods

6. CONCLUSION

The performance of the MapReduce greatly depends on the input data splitting that happens before the map phase. The number of input splits decides the number of map tasks. Usually, the input data splitting is done using hash based methods. But these methods may result in non local reading of data by the map task and consumption of large network resources. The new Improved Input Splitting method splits the input data in such a way that the replicas are chosen based on their locality. This results in assigning data to local map tasks. This greatly improves the performance of MapReduce. The performance was tested using the WordCount program. It is seen that the Improved Input Splitting method performs better than the already existing method.

7. REFERENCES

[1] J. Tan, S. Meng, X. Meng, et al., ”Improving ReduceTask data locality for sequential MapReduce jobs,” in INFOCOM, 2013 Proceedings IEEE, 2013, pp. 1627-1635

[2] R. Vernica, A. Balmin, K. S. Beyer, et al., ”Adaptive MapReduce using situation-aware mappers,” in Proceedings of the 15th International Conference on Extending Database Technology, 2012, pp. 420-431.

[3] A. Rasmussen, M. Conley, G. Porter, et al., ”Themis: an I/O-efficient MapReduce,” in Proceedings of the Third ACM Symposium on Cloud Computing, 2012, p. 13.

[4] S. Ibrahim, H. Jin, L. Lu, et al., ”Maestro: Replica-aware map scheduling for mapreduce,” in Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, 2012, pp. 435-442.

[5] M. Hammoud and M. F. Sakr, ”Locality-aware reduce task scheduling for mapreduce,” in Cloud Computing Technology and Science (Cloud- Com), 2011 IEEE Third International Conference on, 2011, pp. 570- 576.

[6] T. Condie, N. Conway, P. Alvaro, et al., ”Online aggregation and continuous query support in mapreduce,” in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, 2010, pp. 1115-1118.

[7] J. Dean and S. Ghemawat, ”MapReduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51,pp.107-113,2008.

[8] H.-c. Yang, A. Dasdan, R.-L. Hsiao, et al., ”Map-reduce-merge:simplified relational data processing on large clusters,” in Proceedings of the 2007 ACM SIGMOD international conference on Management of data, 2007, pp. 1029-1040.

[9] Hadoop is released as source code tarballs with corresponding binary tarballs for convenience http://hadoop.apache.org/

[10] https://www.mapr.com/blog/understanding-mapreduce-input-split-sizes-and-mapr-fs-chunk-sizes#.VQcuCfmUegI

[11] http://dailyhadoopsoup.blogspot.in/2014/02/mapreduce-inputs-and-splitting.html

[12] The paperwork for opening a business or getting unemployment http://www.openstack.org/

[13] http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/hdfs-and-mapreduce.html

[14] http://www.revelytix.com/?q=content/hadoop-overview

[15] MarkLogic Connector for Hadoop Developer’s Guidehttp://docs.marklogic.com/hadoop:get-splits

[16] http://grepcode.com/file/repository.cloudera.com/content/repositories/releases/com.cloudera.hadoop/hadoop-core/0.20.2737/org/apache/hadoop/mapreduce/lib/input/FileInputFormat.java

[17] Chunguang Wang; Qingbo Wu; Yusong Tan; Wenzhu Wang; Quanyuan Wu, "Locality Based Data Partitioning in MapReduce," Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on , vol., no., pp.1310,1317, 3-5 Dec. 2013