International Journal of Engineering Trends and Technology (IJETT) – Volume 12 Number 2 - Jun 2014

Speed-up Extension to Hadoop System

Sayali Ashok Shivarkar,

Computer Network, Sinhgad College of Enginnering
Pune, India

Abstract— For storage and analysis of online or streaming data which is too big in size most organization are moving toward using Apaches Hadoop- HDFS. Applications like log processors, search engines etc. using Hadoop Map Reduce for computing and HDFS for storage. Hadoop is most popular for analysis, storage and processing very large data but there need to be lots of changes in hadoop system. Here problem of data storage and data processing try to solve which helps hadoop system to improve processing speed and reduce time to execute the task. Hadoop application requires streaming access to data files. During placement of data files default placement of Hadoop does not consider any data characteristics. If the related set of files is stored in the same set of nodes, the efficiency and access latency can be increased. Hadoop uses Map Reduce framework for implementing large-scale distributed computing on unpredicted data sets. There are potential duplicate computations being performed in this process. No mechanism is to identify such duplicate computations which increase processing time.

Solution for above problem is to co-locate related files by considering content and using locality sensitive hashing algorithm which is a clustering based algorithm will try to co - locate related file streams to the same set of nodes without affecting the default scalability and fault tolerance properties of Hadoop and for avoiding duplicate computation processing mechanism is developed which store executed task with result and before execution of any task stored executed tasks are compared if task find then direct result will be provided . By storing related files in same cluster which improve data locality mechanism and avoiding repeated execution of task improves processing time, both helps to speed up execution of Hadoop.

Key term — Hadoop, Hdfs, MapReduce, Hashing Algorithm.

I. INTRODUCTION

Apaches Hadoop is open source implementation of Google Map/Reduce framework, it enables data intensive, distributed and parallel applications by diving massive job into smaller tasks and massive data sets into smaller partition such that each task processes a different partition in parallel. Map tasks that process the partitioned data set using key/value pairs and generate some intermediate result. Reduce tasks merged all intermediate values associated with keys. Hadoop uses Hadoop Distributed File System (HDFS) which is distributed file system, used for storing large data files. Each file is divided into numbers of blocks and replicated for fault tolerance. HDFS cluster is based on master/slave architecture. Name Node work as master which manages and store the file system namespace and provide access to the client. The slaves are number of Data Nodes. HDFS provides a file system name space and allows user data to be stored in files. File is divided into number of block; size of block is normally 64MB which is too large.

The default placement of Hadoop does not consider any data characteristics during placement. If related files are kept in same set of data nodes, the access latency and efficiency will be increased. The file similarity will be calculated by comparing content of it and to reduce comparison, a Locality Sensitive Hashing will be used. Hash function hash the points using different hash function in such way that probability of collision will be higher for similar points. Client is controlling overall process and providing sub-clusterid where file will be placed otherwise default placement strategy is used. Data aware cache is introduced for avoiding execution of repeated task, which requires each data object indexed by its content and also implement cache request and reply protocol. The rest of this paper is organized as follows: Section II gives overview of related work done before; Section III describes programmer's design which include mathematical model; Section IV discuses result and discussion and Section V conclude with conclusion.

II. RELATED WORK

Performance of Hadoop system will be improved if related files are placed in similar set of nodes. Considering past work some techniques are used which provides some degree of collocation but needs lots of changes in framework. Co-locating related file in HDFS, Co-Hadoop[1],provides solution, which helps the application to control data placement at file system level. For achieving this, one new file level property is added to HDFS called Locator, which gives information where file will be place. File locator table is added to Name Node to store locator information. If files are having same locator value then all files are placed on same set of node but if locator information is not provided then default placement strategies of Hadoop is used. Co-Hadoop is best for applications which continuously consume data and it improves performance without effecting Hadoop framework. But Co- Hadoop requires detailed knowledge of input to place file with correct locator value.

RCFile[2], is data placement structure introduced for MapReduce based data warehouse like HDFS. It combines advantage of row and column structure. RCFile is based on principal if data is in one row then that will be placed on same set of nodes.

Performance of data intensive application is enhanced if data is placed near to the computation node this mechanism is used in DARE[3]. High degree of data locality is achieved in DRAE using data replication mechanism, in which each node executes algorithm for creating replicas of heavily accessed files in short interval of time. Usually when map task requires data on remote node, that data will be fetched and used without storage but in DARE that remote data will be stored locally by increasing number of replication factor by one. DARE improves data locality without network overhead but requirement of storage is very high. Considering data in HDFS can be classified into three data access patterns:

1. Hot data, data having concurrent access
2. Cold data, data having rarely access or unpopular data and
3. Normal data, rest of data other than hot and cold.

Data in HDFS is replicated three times by default, replication of hot data only on three machines is not adequate at same time replicating cold data on three machines introduces storage wastage. Considering all this ERMS[4], uses active/standby model where some node working as active node where hot data is replicated on more and more machine and at same time cold data are removed. In non-dedicated distributed computing environment host could be interrupted randomly and potentially leaves system. When host leaves system, task running on host will fail to execute. An interrupted task can be re-executed on different host or on same host if it returns after interruption.

Considering this ADAPT[5], works to achieve high reliability without need of additional data replication facility . Based on availability of host ADAPT distribute data blocks dynamically which improves data locality and reduces network traffic. ADAPT guarantees that all host finishes processing of their allocated job at same time which improve execution time. When there are large numbers of small files, each having less than size of block size of HDFS then that file size become block size. Hence the corresponding metadata stored at Name Node is considerably high and accessing these files lead to network overhead. To solve this problem of small files, solution is developed called Extended Hadoop (EHadoop) [6], in which set of related files combined into one large file to reduce a file count and job of finding related files is done by client. Indexing mechanism is used to access single file from set of related and combined file which improves I/O performance and minimizes load on Name Node.

Another way to find data relation is to analyze log file which provides information of commonly access files, this technique is used in DRAW[7]. DRAW dynamically scrutinizes data access from system log files. It extracts optimal data grouping and re organizes data to achieve maximum parallelism per group. DRAW work as follow: first, data access history graph is generated to exploit system log files to learn data grouping information. In second step weight of file is calculated and generates the optimized file grouping and finally it uses optimal data placement algorithm to form optimal data placement. With the development in social network the amount of image and video upload on internet site is increasing day by day. HDFS is not designed for storing such small mass files so Hadoop Image Processing Interface(HIPI)[8], is developed, which provides interface for computer vision with MapReduce technology. Hadoop Multimedia Processing Interface based on HIPI, aiming to solve problems of storage of small multimedia files effectively and provides interface for user. Here HMPT finds small image and video files automatically and combined into one large file. Clustering is the classification of object like data, articles, and documents etc. into different groups in such way that object in the group have some same characteristics. Conventional clustering can be divided into hierarchical and partitional clustering. Hierarchical clustering finds new cluster using previously created cluster and partitional clustering finds the clusters all at once.

III. PROGRAMMER'S DESIGN

Data placement and data processing modified in such way that it does not affect default characteristics of hadoop and also improves data placement and processing speed of hadoop. For placing related files clustering algorithm is used. Conventional clustering methods are inadequate because these methods may requires multiple iteration and whenever an new object arrive it need to be compare with all existing clusters to find similar one but it requires much delay because cluster is very large.

Instead of using above method for data placement, we propose a method for clustering related files incrementally in Hadoop-HDFS using locality sensitive hashing. The idea behind locality sensitive hashing is that similar files will have similar signature which helps to easily find similar files. If similar files are stored in same nodes then data locality mechanism of hadoop system is improved.

Avoiding execution of duplicate task which waste time but also processing time will be increased. To solve this problem data aware cache aims at extending MapReduce framework by implementing cache description scheme and cache request and reply protocol. Cache description scheme need to identify source input from which a cache item and the operation is obtained.

Cache refers to the intermediate result that is produced by worker node during the execution of a MapReduce phase. A piece of cached data is stored in a distributed file system. The content of cache items describe by two tuple: origin and operation where origin is name of file and operation is operation performed on file and also result is stored. When map task is executed name of file and operation and result are stored in cache. After some time if same task need to be executed then it scan the cache first if same file name and same operation is found then direct result will be provided. After comparing file name and operation if file is not found in cache then the normal map execution take place. Here not only file name and operation are compared but also splitting of file is also need to compare for correct result.

By modifying two modules of hadoop which is most important that helps to improves speed up of hadoop.

A. Design of System

Data is everywhere now. The amount of information available now is very huge for analysis, storage and process such huge information Apaches Hadoop tool is much popular. Hadoop uses HDFS for storage and MapReduce for analysis. If default placement of Hadoop is considered then it places file anywhere in cluster. Hadoop is uses principal of data locality means tasks are executed where data are placed but in practices this will not be true for all data files. If needed files are placed on different nodes then that files need to be copied to worker node for task execution. But when placing files data characteristics are considered then related files are stored in same node which improve data locality and reduce network traffic.

MapReduce does not have any mechanism to find whether task is executed before so that result can be re used but result of executed task is not stored so if task is executed again then there is no mechanism is available to find task and reuse the result.

The purpose of the experiment is to extend Hadoop system by improving the data placement and execution policies of Map/Reduce. The file similarity will be calculated based on its content and similar file will be placed in same data node or nearby data nodes (sub- cluster). The result of Map/Reduce will be stored in cache as framework to access them again if same task is executed on same data repeatedly.

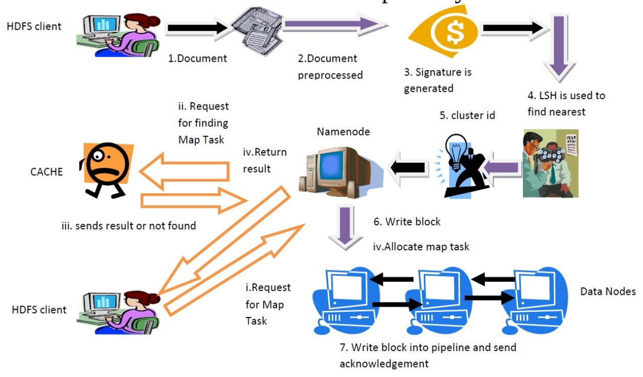
HDFS client
1.Document
2.Document preprocessed
3. Signature is generated
4. LSH is used to find nearest
5. cluster id
Namenode
ii. Request for finding Map Task
iv.Return result
CACHE
iii. sends result or not found
6. Write block
iv.Allocate map task
HDFS client
i.Request for Map Task
Data Nodes
7. Write block into pipeline and send acknowledgement

Fig 2. System Architecture

Overall system architecture is shown fig 2. Suppose user wants to upload a file then client finds a cluster which contain similar file. To find cluster client executes following modules:

1. Preprocessing File: File contain collection of words, file is pre-process means words like stop words are removed, stop word are word like ‘a ’, ‘of ’, ‘the’ etc. and also stemming (historical is replace with history) and many techniques are used to pre-process a file. After preprocessing file will contain collection of word which related to particular file and which can be use to represent that file.

2. File Vector: after preprocessing file which contains collection of words from that words which are presenting that file need to find, this is done using TFIDF technique. TF-IDF(Term Frequencies-Inverse Document Frequencies) technique finds words in file that come many times compare to all remaining files, which indicate that word is representing a file and it is important word in file. If word is representing that file then that word can be use to find similar files.

3. Create Signature - To find similar file it should be compared with content of each and every files available but there are millions of files which makes process time consuming. So to make process faster compact bit representation of each file vector is created, Signature. To create Signature f bit vector is used and this vector initialized to zero first then it hashed with file vector and comparing value is 0 or 1 weight of word will be incremented or decremented. Advantage of Signature is that similar file will have same Signature which makes process faster.

4. Use Locality Sensitive Hashing to find nearest neighbor- In large clustering environment to compare file Signature to each and every cluster is time consuming to avoid comparing each and every cluster locality sensitive hashing technique is used which ensures that only nearest neighbour need to be checked to place file. For this hashing function is used which query file Signature to find nearest neighbour and m number of neighbour is returned to client.

5. Store file with related files- If m neighbour is return to client then only that m neighbour will be compared and after finding cluster where file will be placed, this subclusterid will be given to Name Node. Name Node maintains subclustertable which store subclusterid and file placed on that cluster. If Name Node finds entry then that file will be placed on subclusterid but if subclusterid is not found then new subcluster will be created, file will be stored on newly created cluster and file and cluster Signature will be calculated and this information will be updated to subclustertable.

Now suppose client want execute map task and system should not execute repeated map task for this, cache will be implemented. Cache table will be created which stores file name, operation perform on that file and result file name.

1. Map task execution: when client wants to execute any map task first then it request cache manager to find file name and operation. If file name and operation performed on that file is same then result file name will be given to directly to reduce phase which completely save execution time of task.

2. Lifetime of cache item fixed size cache will be used and if cache is full then older entry will be deleted.

The project requires various data structure to perform various modules in proposed system which is listed below: Data structure for locality sensitive hashing function, Data structure for SubCT, Data structure for storing mapping information, Data Structure for CacheTable, Data Structure for storing intermediate result. All above structure will be either array of structure or linked list or object of classes.

The internal data structures will be used to store result obtained by map task it need to store locally because it improves data locality. The data structure to create mapping information at client will contain 1) clusterid, 2) Signature of cluster ids 3) cluster centroids 4) file name 5) Signature of file. The data structure require to store hash table at client side which store cluster Signature as key and cluster information as value.

The global data structures use for maintaining Subclusterid table for indexing of Sub-cluster id and file which is having same subclusterid. etc. The structure of Sub-cluster is as follow:

TABLE I SUB-CLUSTER TABLE

Sub-clusterid	Files
1	A,B
2	C
N	O

The data structures will be structure used to data structure to create CacheTable will contain 1) name of file, 2) type of operation performed on file 3) result file name.

TABLE II CACHE TABLE

Sr. No	. Origin	Operation	resultfilename
1	ab.txt	Sort	Db.txt
2	cd.pdf	Count	Jk.txt
3	Hg.html	Sort	Io.txt

All the above mentioned data structures will be used only for a particular module, hence they need not be declared globally. The storage requirement is less.

IV. RESULTS

The proposed system "Speed up Extension to Hadoop system" is currently in its development phase. We intend to record the test performance of the system based on following parameter:

1. File Placement: file should be placed in correct and related cluster so that all related file can be access easily which improves data locality.
2. Comparison to cluster should be less: given m cluster is correct so only m cluster need to compare.
3. Access time should be less: Related files are placed in same cluster so access time should be less.
4. Performance of MapReduce should be increase: Because of data locality is improved and avoiding execution of duplication of MapReduce task increases performance of MapReduce.

V. CONCLUSION

"Speedup extension to Hadoop system" is the modification in the input format and task management of the map reduce framework. The applications, using this modified Hadoop need not to change at all. The proposed system shows that it can eliminate all the duplicate tasks and new approach for incremental file clustering is proposed for HDFS which will cluster similar files in the same set of data nodes with minimal changes to the existing framework.

For faster clustering operations bit wise representation of the feature vectors called Signature are used. To reduce the number of cluster centroid comparisons, only the nearest neighbours are considered using the technique of Locality Sensitive Hashing.

In this experiment two modules are combined first is data placement modified with clustering and second is map/reduce tasks execution time is reduce and processing is done faster. So this experiment speeds up hadoop system by changing data placement and task execution.

REFERENCES

[1] Eltabakh, "CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop, " proceedings of the vldb endowment, june 2011, 4 (9), pp. 575-585.
[2] Yongqiang He; Rubao Lee; Yin Huai; Zheng Shao; Jain, N.; Xiaodong Zhang; Zhiwei Xu, "RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems," Data Engineering (ICDE), 2011 IEEE 27th International Conference on , vol., no., pp.1199,1208, 11-16 April 2011.
[3] Abad, C.L.; Yi Lu; Campbell, R.H., "DARE: Adaptive Data Replication for Efficient Cluster Scheduling," Cluster Computing (CLUSTER), 2011 IEEE International Conference on , vol., no., pp.159,168, 26-30 Sept. 2011.
[4] Zhendong Cheng; Zhongzhi Luan; You Meng; Yijing Xu; Depei Qian; Roy, A.; Ning Zhang; Gang Guan, "ERMS: An Elastic Replication Management System for HDFS," Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on , vol., no., pp.32,40, 24-28 Sept. 2012.
[5] Hui Jin; Xi Yang; Xian-He Sun; Raicu, I., "ADAPT: Availability-Aware MapReduce Data Placement for Non-dedicated Distributed Computing," Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on , vol., no., pp.516,525, 18-21 June 2012.
[6] Bo Dong; Jie Qiu; Qinghua Zheng; Xiao Zhong; Jingwei Li; Ying Li, "A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files," Services Computing (SCC), 2010 IEEE International Conference on , vol., no., pp.65,72, 5-10 July 2010.
[7] Pengju Shang; Qiangju Xiao; Jun Wang, "DRAW: A new DatagRouping-AWare data placement scheme for data intensive applications with interest locality," APMRC, 2012 Digest , vol., no., pp.1,8, Oct. 31 2012-Nov. 2 2012
[8] Jia Li; Kunhui Lin; Jingjin Wang, "Design of the mass multimedia files storage architecture based on Hadoop," Computer Science and Education (ICCSE), 2013 8th International Conference on , vol., no., pp.801,804, 26-28 April 2013
[9] Shvachko, K.; Hairong Kuang; Radia, S.; Chansler, R., "The Hadoop Distributed File System," Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on , vol., no., pp.1,10, 3-7 May 2010
[10] Kala Karun, A.; Chitharanjan, K., "A review on hadoop HDFS infrastructure extensions," Information and Communication Technologies (ICT), 2013 IEEE Conference on , vol., no., pp.132,137, 11-12 April 2013
[11] Kala, K.A.; Chitharanjan, K., "Locality Sensitive Hashing based incremental clustering for creating affinity groups in Hadoop HDFS - An infrastructure extension," Circuits, Power and Computing Technologies (ICCPCT), 2013 International Conference on , vol., no., pp.1243,1249, 20-21 March 2013
[12] Yaxiong Zhao; Jie Wu, "Dache: A data aware caching for big-data applications using the MapReduce framework," INFOCOM, 2013 Proceedings IEEE , vol., no., pp.35,39, 14-19 April 2013
[13] Juan Ramos, Using TF-IDF to Determine Word Relevance in Document Queries, Department of Computer Science, Rutgers University, 23515 BPO Way, Piscataway, NJ, 08855
[14] Gurmeet Singh Manku, Arvind Jain, Anish Das Sarma, "Detecting NearDuplicates for Web Crawling"by google
[15] Tom white,"Hadoop definitive guide" o'Reilly ,yahoo,2010
[16] http://hadoop.apache.org/core