CACHE MECHANISM TO AVOID DULPICATION OF SAME THING IN HADOOP SYSTEM TO SPEED UP THE EXTENSION

Ritu A.Mundada[1], Aakash.A.Waghmare[2]

[1]*M. E Student, Department of Computer Science, Ssbt College, Jalgaon, India*
[2]*Associate Professor, Department of Computer Science, Ssbt College, Jalgaon, India*

Abstract

Cloud computing provides a proper platform for hosting large-scale data-intensive applications. MapReduce is a programming model as well as a framework that supports the model. The main idea of the MapReduce model is to hide details of parallel execution and allow users to focus only on data processing strategies. Hadoop is an open-source implementation for MapReduce. For storage and analysis of online or streaming data which is big in size. Most organization are moving toward Apaches Hadoop HDFS. Applications like log processors, search engines etc. ueses hadoop Map reduce for computing and HDFS for storage. Hadoop is popular for analysis, storage and processing of very large data but require to make changes in hadoop system. There is no mechanism to identify duplicate computations which increase processing time and unnecessary data transmission .To co-locate related files by considering content and using locality sensitive hashing algorithm. By storing related files in same cluster using cache mechanism which improve data locality mechanism and avoids repeated execution of task, both helps to speed up execution of hadoop.

Keywords-*Distributed file system, Datanode, Locality Sensitive Hashing*

----------***----------

1. INTRODUCTION

In present scenario with the internet of things a lot of data is generated and is analyzed mainly for business intelligence. There are various sources of Big Data like social networking sites, sensors, transactional data from enterprise applications/databases, mobile devices, machine generated data, huge amount of data generated from high definition videos and many more sources. Some of the sources of this data have vital value that is helpful for businesses to develop. Hadoop is a distributed computing platform written in Java which incorporates features similar to those of the Google File System and MapReduce programming paradigm. 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.

In section II we discuss in more about some basic types of schedulers used in Hadoop and scheduler improvements. Further section III talks about related work. section IV actual mechanisms of cache. Finally section V concludes the paper after which references follow.

2. SCHEDULING IN HADOOP

The default Scheduling algorithm is based on FIFO where jobs were executed in the order of their submission. Later on the ability to set the priority of a Job was added. Facebook and Yahoo contributed significant work in developing schedulers i.e. Fair Scheduler and Capacity Scheduler respectively which subsequently released to Hadoop Community.

2.1 Default FIFO Scheduler

The default Hadoop scheduler operates using a FIFO queue. After a job is partitioned into individual tasks, they are loaded into the queue and assigned to free slots as they become available on TaskTracker nodes. Although there is

support for assignment of priorities to jobs, this is not turned on by default. Typically each job would use the whole cluster, so jobs had to wait for their turn. Even though a shared cluster offers great potential for offering large resources to many users, the problem of sharing resources fairly between users requires a better scheduler. Production jobs need to complete in a timely manner, while allowing users who are making smaller ad hoc queries to get results back in a reasonable time.

2.2 Fair Scheduler

The Fair Scheduler was developed at Facebook to manage access to their Hadoop cluster and subsequently released to the Hadoop community. The Fair Scheduler aims to give every user a fair share of the cluster capacity over time. Users may assign jobs

to pools, with each pool allocated a guaranteed minimum number of Map and Reduce slots. Free slots in idle pools may be allocated to other pools, while excess capacity within a pool is shared among jobs. The Fair Scheduler supports preemption, so if a pool has not received its fair share for a certain period of time, then the scheduler will kill tasks in pools running over capacity in order to give the slots to the pool running under capacity. In addition, administrators may enforce priority settings on certain pools. Tasks are therefore scheduled in an interleaved manner, based on their priority within their pool, and the cluster capacity and usage of their pool. As jobs have their tasks allocated to Task Tracker slots for computation, the scheduler tracks the deficit between the amount of time actually used and the ideal fair allocation for that job.

2.3 Capacity Scheduler

Capacity Scheduler originally developed at Yahoo addresses a usage scenario where the number of users is large, and there is a need to ensure a fair allocation of computation resources amongst users. The Capacity Scheduler allocates jobs based on the submitting user to queues with configurable numbers of Map and Reduce slots. Queues that contain jobs are given their configured capacity, while free capacity in a queue is shared among other queues. Within a queue, scheduling operates on a modified priority queue basis with specific user limits, with priorities adjusted based on the time a job was submitted, and the priority setting allocated to that user and class of job. When a Task Tracker slot becomes free, the queue with the lowest load is chosen, from which the oldest remaining job is chosen. A task is then scheduled from that job. Overall, this has the effect of enforcing cluster capacity sharing among users, rather than among jobs, as was the case in the Fair Scheduler.

2.4 Delay Scheduling

Fair scheduler is developed to allocate fair share of capacity to all the users. Two locality problems identified when fair sharing is followed are – head-of-line scheduling and sticky slots. The first locality problem occurs in small jobs (jobs that have small input files and hence have a small number of data blocks to read). The problem is that whenever a job reaches the head of the sorted list for scheduling, one of its tasks is launched on the next slot that becomes free irrespective of which node this slot is on. If the head-of-line job is small, it is unlikely to have data locally on the node that is given to it. Head-of-line scheduling problem was observed at Facebook in a version of HFS without delay scheduling. The other locality problem, sticky slots, is that there is a tendency for a job to be assigned the same slot repeatedly. The problems aroused because following a strict queuing order forces a job with no local data to be scheduled.

2.5 Dynamic Priority Scheduling

Thomas Sandholm *et al* .proposed Dynamic Priority Scheduler that supports capacity distribution dynamically among concurrent users based on priorities of the users. Automated capacity allocation and redistribution is supported in a regulated task slot resource market. This approach allows users to get Map or Reduce slot on a proportional share basis per time unit. These time slots can be configured and called as allocation interval. It is typically set to somewhere between 10 seconds and 1 minute. For example a max capacity of 15 Map slots gets allocated proportionally to three users. The central scheduler contains a Dynamic Priority Allocator and a Priority Enforcer component responsible for accounting and schedule enforcement respectively. This model appears to favor users with small jobs than users with bigger jobs. However Hadoop MapReduce supports scaling down of big jobs to small jobs to make sure that fewer concurrent tasks runs by consuming the same amount of resources.

3. 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 colocation but needs lots of changes in framework. Colocating related file in HDFS, Co-Hadoop[1],provides solution, which helps the application to control data placement at file system level.

Considering this ADAPT[3], 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.

DARE[2]Adaptive DataReplication for Efficient Cluster Scheduling .DARE improves data locality without network overhead but requirementof storage is very high.

Down analyzes the performance of widely used hadoop schedulers including FIFO and Fair sharing and compares them with the COSHH (Classi_cation and Optimization

based Scheduler for Heterogeneous Hadoop) scheduler. The scheduler is a combination of the three analyzed algorithms. The selector chooses an appropriate scheduler as the number of jobs and resources scale up or down. However, the overall solution is to use the COSHH algorithm when the system is overloaded (e.g., during peak hours), the FIFO algorithm for under loaded systems (e.g., after hours), and the Fair Sharing algorithm when the system load is balanced. A combination of the FIFO, Fair Sharing, and COSHH schedulers is e_ective, where the selection is based on the load on the system and available system resources. A job scheduler is an essential component of every hadoop system.

4. PROPOSED SOLUTION

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.

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. TFIDF(Term Frequencies-Inverse Document Frequencies) technique finds words in file that come many times compare to all remaining files, which indicate that word is representinga 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 subcluster table.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 1 Sub-Cluster Table

Sub Clusterid	Files
1	A,B
2	C
N	0

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.

5. 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] Yongqiang He, Rubao Lee, Yin Huai, Zheng Shao, "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.

[2] Abad, C.L., Yi Lu, Campbell, R.H., "DARE:Adaptive Data Replication for Efficient ClusterScheduling," Cluster Computing (CLUSTER), 2011IEEE International Conference on , vol., no., pp.159,168, 26-30 Sept. 2011.

[3] Hui Jin; Xi Yang; Xian-He Sun; Raicu, I., "ADAPT: Availability-Aware MapReduce Data Placement for Non-dedicated DistributedComputing," Distributed Computing Systems(ICDCS), 2012 IEEE 32nd International Conference on , vol., no., pp.516,525, 18-21 June 2012.

[4] 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.

[5] Kala Karun,''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.

[6] Kala, K.A., Chitharanjan,''Locality Sensitive Hashing based incremental clustering for creating affinity groups in Hadoop HDFS – An infrastructure extension,"Circuits, Power and ComputingTechnologies (ICCPCT), 2013 International Conference on , vol., no., pp.1243,1249, 20-21 ,March 2013.

[7] Yaxiong Zhao, Jie Wu, ''Dache: A data aware caching for big-data applications using the Map Reduce framework," INFOCOM, 2013 Proceedings IEEE , vol., no., pp.35,39, 14-19 April 2013.

[8] Hui Jin; Xi Yang; Xian-He Sun;,''ADAPT: Availability-Aware MapReduce Data Placement for Non-dedicated Distributed Computing," DistributedComputing Systems (ICDCS), 2012 IEEE 32nd International Conference on , vol., no., pp.516,525, 18-21 June 2012