Scientific Journal Impact Factor (SJIF) : 1.711
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745 ISSN (PRINT) : 2393 - 8161

Survey on Incremental MapReduce for Data Mining

Trupti M. Shinde[1], Prof.S.V.Chobe[2]

[1]*Research Scholar, Computer Engineering Dept., Dr. D. Y. Patil Institute of Engineering &Technology,*
[2]*Associate Professor, Computer Engineering Dept., Dr .D. Y. Patil Institute of Engineering & Technology,*

Abstract — As new data and updates are continuously arriving; the results of data mining applications turn out to be stale and obsolete over time. Incremental processing is a talented move towards to refreshing that mining results, it utilizes previously saved states to avoid the expense of re-computation from scratch. Incremental MapReduce, a work of fiction incremental processing extension to MapReduce, the mostly used structure for mining data. Compared with the state-of-the-art work on Incoop, Incremental MapReduce performs key-value pair level incremental processing somewhat than task level re-computation, supports not only one-step computation but also more sophisticated iterative computation, which is widely used in data mining applications, and incorporates a set of techniques to reduce I/O overhead for accessing preserved fine-grain computation states. Incremental MapReduce using a one-step algorithm and four iterative algorithms with diverse computation description works efficiently than existing Incoop.

Keywords—Data Mining, MapReduce, Incoop, IncMR, Incremental MapReduce

I. INTRODUCTION

The vast amount of digital data is being collected in many important areas, like electronic commerce, Social media, finance, health care, education, and environment. Now it is necessity of mining such a data to get meaningful insights into the business needs which will assist to take business decisions and also affords to an individual advanced quality services. Computing frameworks are available which are developed for big data analysis MapReduce is one of them (with its open-source implementations, such as Hadoop), and is the most generally used in production. The Incremental MapReduce enhancing the existing MapReduce with added features. Big data is constantly changing, when new data and updates are being collected, the input data of a big data mining algorithm will gradually change, and the computed results will be outdated as time passes. In some cicumstances, it is very important to refresh those mining results periodically, so that the mining results become up-to-date and in time.

In [1]. Y.Zhang,Et.al proposed Incremental MapReduce which supports incremental processing which is a talented move towards refreshing the mining results. For any given set of input data, rerunning the entire computation from the starting incurs high cost. Incremental processing gives incremental computation approach for refreshing those mining results with saving the state of existing data to perform mining only on that much data which are affected by the change. It provides incremental processing for both one step and iterative computation.

II. RELATED WORK

In [2]. J. Dean and S. Ghemawat discussed about MapReduce framework as following.

2.1 MapReduce –

This Programming model consists two functions Map & Reduce. Map, written by the user, it takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups

together all intermediate values associated with the same intermediate key and passes them to the Reduce function, and the Reduce function, also written by the user, accepts an intermediate key and a set of values for that key, it merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. MapReduce programs written in this functional format are automatically parallelized which are executed on large set of machines as shown in fig.1.

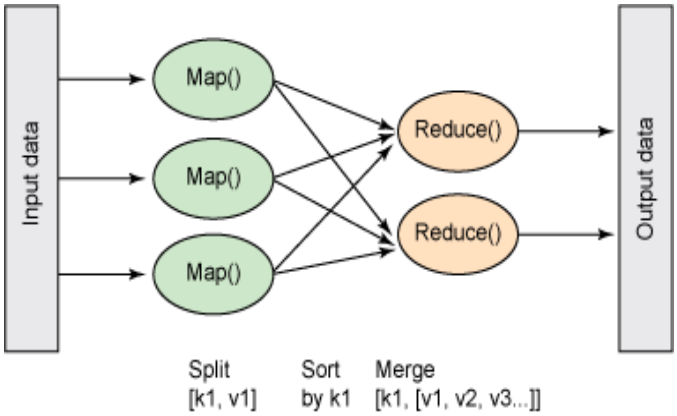

Fig.1 MapReduce basic operation

The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This enable programmers without any experience with parallel and distributed systems to utilize the resources of a large distributed system in simple way, but still there are some limitations while MapReduce taking into consideration, although same operation in each iteration of MapReduce is performed, it will take new job for every iteration when it will start implementation, and initialization and loading of new job data is required in MapReduce,which incurs unnecessary I/O overhead and after finishing all the reduce tasks of previous iteration only, the map task will start in next iteration but, still there is need of starting the map task, whenever input data is available. MapReduce lacks built-in support for the iterative process.

In [3]. P. Bhatotia, Et.al discussed about Incoop framework in following manner.

2.2 Incoop –

This is the framework of MapReduce which is an extension to hadoop for incremental processing. Incoop identifies changes in the inputs and allows the automatic update in the outputs by using an efficient, detailed result re-use mechanism. To achieve efficiency with transparency, incremental computation of the reduce tasks is more efficient. Incoop computations enables automatic respond to modification in input data with reuse of previous computation and incremental updation in the output with respective to change in input. Incoop includes content-based chunking to the file system which identifies incremental changes in the input file and partition the data so that it allows re-use, In the reduce phase, a new scheduler that takes the location of previously computed results into account, which delivers incrementally in computation for bulk data processing as shown in fig.2. It's design is based on the self-adjusting computation. This means keeping track of the dependency exist between input and output of various parts of single computation and recomputation on only those parts which are affected by changes.

.

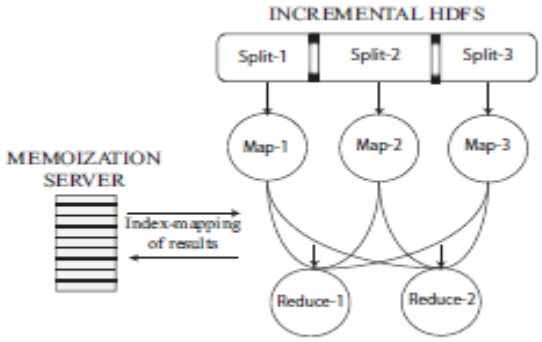

Fig.2 Basic design of Incoop

Incoop, allows existing MapReduce programs incremental processing but with efficient modifications in input and with reuse of intermediate results from previous computations.

In[4]. M. Zaharia, Et.al proposed iMapReduce framework which is as below.

2.3 iMapReduce –

iMapReduce enables users to state the iterative operations with map and reduce functions, at the same time supporting the automatic iteration regardless of user participation. iMapReduce extensively improves the performance of iterative algorithms by -
I . Avoiding the creation of new job for each iteration.
II. Enables beginning of new map task regardless of previous reduce tasks completion.
All map and reduce task in iMapReduce is persistent. The persistence of map or reduce task enables the iterative process through keeping alive during the whole iterative process. It also differs static and state data, static data remains same while state data changes in each iteration. Unlike MapReduce, iMapReduce supports passing of the state data from reduce task to the map task for triggering between state data and static data and starts the next iteration as shown in fig.3.

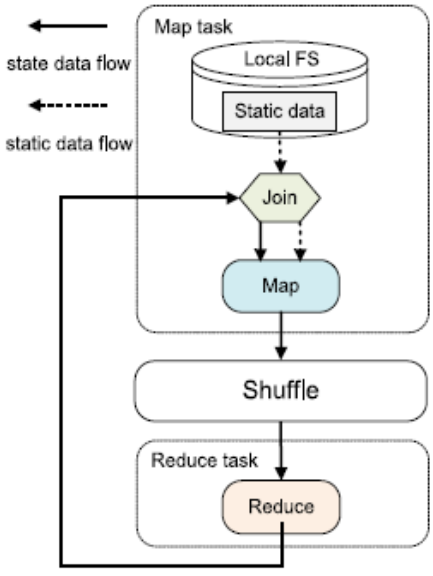

Fig.3 Dataflow of the state data and the static data in an iMapReduce worker

iMapReduce, schedules the execution of Map tasks asynchronously, as each map task needs only the state data from its corresponding reduce task, a map task can start its execution as soon as its state data arrives, without waiting for other map tasks. As well as it gives better load balancing, as the master gets notified for completion by the reduce tasks subsequent to each iteration. iMapReduce also provides fault tolerance if failure occurs, iMapReduce returns to the last iteration with the state data, instead of begins the iterative process from scratch.

In[5]. Y. Zhang, Et.al put their great insights onto IncMR framework which is as following.

2.4 IncMR-

MapReduce programming model generally used for large scale and distributed computing, but are unable to processing small incremental data. IncMR framework enables incremental process of new small data within large data set, it will take state as an implicit input and join it with new data, according to this new splits within large data set Map tasks are created, at the same time reduce task obtain their inputs which having the state and intermediate results of new Map task. IncMR is a enhanced model for large-scale incremental data processing , that uses the original MapReduce model. Without any modification in existing Map-Reduce model IncMR delivers incremental processing. It supports incremental data processing, intermediate state preservation and incremental map and reduce functions.

In this model the Map tasks are same as MapReduce model map tasks, and reduce tasks executions are same as MapReduce Model's map tasks and reduce task. But there is a change in copy phase of reduce task, due to the inclusion of current map output with state in data resource. It allows MapReduce-based applications without any modification. It enables incremental processing with task-level re-computation, but in this framework users themselves have to manipulate the states.

In [6]. C. Yan, Et.al put their great thoughts on Spark System.

2.5 Spark System (Resilient Distributed Datasets (RDDs)-

A distributed memory abstraction which allows programmers to process in-memory computations on large clusters with fault-tolerant capacity. RDDs are forced by two types of applications which present computing framework can't handle in an efficient manner-
- Iterative algorithms and Interactive data.
-Data mining tools.
Above mentioned applications, can improve their performance with an order of magnitude with keeping that data in memory, they also achieve fault tolerance efficiently as they are allowing use of shared memory in limited form. It mainly used to share data in cluster application. RDDs are parallel data structure which allows users data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that enables users with in memory persistence of intermediate results, as well as hold control on results partitioning for optimal placement of data and manipulate them with rich set of operators. RDDs offer an API which provides coarse grained transformations which enables them in recovery of data efficiently using lineage this means RDD gives such an interface which is based on coarse gained transformations which allows to do same operation for variety of data items. RDDs provide an interface which enables them for providing fault tolerance mechanism with logs of transformations which are used to build lineage or data set instead of actual data.

RDD's are performed in Spark to achieve all included features. Spark system respresents RDDs by a language-integrated Application programming interface. In this API every dataset denoted as object and transformations are called with the use of methods. Programmers in this API begins with defining one or more than one RDDs by transformation apply on data in Steady storage after this

these RDDs use in action, those are operations which returns value to the respective application or send data to steady storage, and some of the actions are count, which gives the no. of elements in dataset, collect will return the element itself, and save action outputs dataset to storage system.

Programmers start by defining one or more RDDs through transformations on data in stable storage (e.g., map and filter). They can then use these RDDs in actions, which are operations that return a value to the application or export data to a storage system. Examples of actions include count (which returns the number of elements in the dataset), collect (which returns the elements themselves), and save (which outputs the dataset to a storage system). The programmers can call a persist method to indicate which RDDs they want to reuse in future operations. Spark keeps persistent RDDs in memory by default, but it can spill them to disk if there is not enough RAM. Users can also request other persistence strategies, such as storing the RDD only on disk or replicating it across machines, through flags to persist.

Finally, users can set a persistence priority on each RDD to specify which in-memory data should spill to disk first. Spark gives better performance than Hadoop due to it's in-memory processing nature, but it's performance suffers if input and intermediate data are not able to fit into memory.

III. CONCLUSION

As described above, the existing frameworks which are available to mine data iteratively, amongst them some are providing one step incremental computation and some others are providing iterative incremental processing , but Incremental MapReduce provides combined one step as well as iterative approach for incremental processing which incorporates small as well as big data set to refresh mining results and saves computation time, and provides high efficiency in computation of mining results.

REFERENCES

[1] Y.Zhang, S.Chen, Q.Wang, G.Yu, "I2MapReduce:Incremental MapReduce for mining evolving Big data," Corr,Vol.ABS/1501.04854, 2015.

[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Proc. 6th Conf. Symp. Opear. Syst. Des.Implementation, p. 10, 2004.

[3] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin, "Incoop: Mapreduce for incremental computations," Proc. 2nd ACM Symp. Cloud Computing., pp. 7:1–7:14,2011.

[4] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for, in-memory cluster computing," in Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation, p. 2,2012,.

[5] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "imapreduce: A distributed computing framework for iterative computation," J. Grid Comput., vol. 10, no. 1, pp. 47–68, 2012.

[6] C. Yan, X. Yang, Z. Yu, M. Li, and X. Li, "IncMR: Incremental data processing based on mapreduce," in Proc. IEEE 5th Int. Conf. Cloud Computing., pp 534–541, 2012.

