isrjournals

International Journal of Advanced Research in Computer Science Engineering and Information Technology

Volume: 6 Issue: 3 Dec,2016,ISSN_NO: 2321-3337

Key-Value Pair Level Incremental Processing in Big Data Environment

Karthikeyan S[1] Dr.Manimegalai R[2], Dr.Hamsapriya T[3]

[1]karthikeyan@psgitech.ac.in, [2]mmegalai@yahoo.com, [3]hamsapriya.t@gmail.com

[1] Assistant Professor, PSG Institute of Technology and Applied Research, Coimbatore.

[2]Professor, Park College of Engineering and Technology, Coimbatore.

[3]Professor, PSG Institute of Technology and Applied Research, Coimbatore.

Abstract— A novel Incremental Processing method is proposed for data analysis in order to keep the mining results up-to-date. To reduce the running time of refreshing the big data results, the proposed workimplements a technique called i[2]Map Reduce which is an extension of Map Reduce that supports Fine Grain Incremental Processing for both One step and Four iterative computation. i[2]MapReduce performs key value pair level incremental processing compared to the state-of- art Incoop, that uses task level re computation. It 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 tale techniques to reduce I/O overhead for accessing preserved fine-grain computation states.

Index Terms—Bigdata,Hadoop,Hdfs,MapReduce,Hive.

INTRODUCTION

Before big data appear, database has become an important processing platform because of the data processing convenience. But when database is faced with non-relational or large-scale data, there is a difficulty dealing with them. Big data not only enhance the related computing services technologies but also change the traditional mode of many industries. Big data is the term for a collection of data sets which are large and complex, it contain structured and unstructured both type of data. Big data is data that exceeds the processing capacity of conventional database systems and it is used when data is too big, moves too fast, or doesn't fit the structures of database architectures. Data mining is a technique for discovering interesting patterns as well as descriptive, understandable models from large scale data. Today huge amount of digital data is being accumulated in many important areas, including e-commerce, social network, finance, health care, education, and environment. It has become increasingly popular to mine such big data in order to gain insights to help business decisions or to provide better personalized, higher quality services. In recent years, a large number of computing frameworks have been developed for big data analysis. Among these frameworks, Map Reduce (with its open-source implementations, such as Hadoop) is the most widely used in production because of its simplicity, generality, and maturity. The main aim of the project to focus on improving map reduce. In recent years big data applications have become ubiquitous. In the big data environment, the traditional data management technology represented by relational database has been unable to effectively manage these data. How to quickly access big data represented by Hadoop has emerged. Many big data applications have the characteristics of increment and iteration. Big data processing tasks always run repeatedly when the data changes very small. The characteristics of data incremental change in big data indicates that it can greatly improve performance by using the incremental mode to process big data.

Big data is constantly evolving. As new data and updates are being collected, the input data of a big data mining algorithm will gradually change, and the computed results will become stale and obsolete over time. In many situations, it is desirable to periodically refresh the mining computation in order to keep the mining results up-to-date. Incremental processing is a promising approach to refreshing mining results. Given the size of the input big data, it is often very expensive to rerun the entire computation from scratch. The core of Hadoop is a distributed file system HDFS and a computing framework MapReduce. Typically, when we process big data in Hadoop platform. data is stored in the HDFS distributed file system and we use MapReduce framework for processing. The existing Hadoop big data management technology don't consider the incremental feature of big data, as a result, it has large processing cost and poor timelines, In order to improve the processing efficiency of incremental big data, distributed file system iHDFS which supports incremental computing, providing basic guarantee for incremental processing of big data This distributed file system iHDFS is an extension to HDFS. The iHDFS reserves compatibility with HDFS and provides the same interfaces and semantics as HDFS. The main idea of iHDFS is to make the input data of MapReduce, namely the data stored in HDFS, divided into chunks based on content. Many Big data application have characteristics of increment and iteration. Big data processing tasks always run repeatedly when the data changes very small. The characteristics of incremental change data improve performance.

isrjournals

International Journal of Advanced Research in Computer Science Engineering and Information Technology

Volume: 6 Issue: 3 Dec,2016,ISSN_NO: 2321-3337

The project has investigate the realization of this principle in the context of the Map Reduce computing framework. A number of previous studies have followed this principle and designed new programming models to support incremental processing. Unfortunately, the new programming models (BigTable observers in Percolator, stateful translate operators in CBP, and timely dataflow paradigm in Naiad) are drastically different from MapReduce, requiring pro grammers to completely re-implement their algorithms.This project implemented i2MapReduce by modifying Hadoop-1.0.3 and evaluate i2MapReduce using a one-step algorithm (A-Priori) and four iterative algorithms (PageRank, SSSP, Kmeans, GIM-V) with diverse computation characteristics. Experimental results on Amazon EC2 show significant performance improvements of i2MapReduce compared to both plain and iterative Map Reduce performing re-computation. For example, for the iterative Page Rank computation with 10 percent data changed, i2 map reduce improves the run time of re-computation on plan map reduce by an eight fold speedup.

RELATED WORK

In Existing various works are corresponds to this area IncMR framework is proposed for incrementally processing new data of a large data set, which takes state as implicit input and combines it with new data. Map tasks are created according to new splits instead of entire splits while reduce tasks fetch their inputs including the state and the intermediate results of new map tasks from designate nodes or local nodes. Data locality is considered as one of the main optimization means for job scheduling. It is implemented based on Hadoop, compatible with the original MapReduce interfaces and transparent to users. Experiments show that non-iterative algorithms running in MapReduce framework can be migrated to IncMR directly to get efficient incremental and continuous processing without any modification. IncMR is competitive and in all studied cases runs faster than that processing the entire data set.In existing work puts focus on the transplantation of parallel algorithms based on MapReduce model and compatibility of non incremental and incremental processing. A parallel programming framework is presented, which aims to be compatible with the original MapReduce APIs so that programmers do not need to rewrite the algorithms and it presents an incremental data processing model which is compatible with the MapReduce model and its runtime. It supports MapReduce-based applications without any modification. One of the existing work presents iMapReduce, a distributed framework that supports iterative processing. iMapReduce allows users to specify the iterative computation with the separated map and reduce functions, and provides the support of automatic iterative processing within a single job. More importantly, iMapReduce significantly improves the performance of iterative implementations by reducing the overhead of creating new MapReduce jobs repeatedly,eliminating the shuffling of static data and allowing asynchronous execution of map tasks. They have implement an iMapReduce prototype based on Apache Hadoop, and show that iMapReduce can achieve up to 5 times speedup over Hadoop for implementing iterative algorithms and they. First, it provides a framework for programmers to explicitly model iterative algorithms. Second, it proposes the concept of persistent tasks to perform the iterative computation to avoid repeatedly creating, destroying, and scheduling tasks. Third, the input data are loaded to the persistent tasks once and do not need to be shuffled between map and reduce. This can significantly reduce the I/O and the network communication overhead and the processing time. Fourth, it facilitates asynchronous execution of tasks within the same iteration, to accelerate the processing speed and they also proposed iMapReduce that supports the implementation of iterative algorithms under a large cluster environment.. The results show a factor of ranging from 1.2 to 5 speedup can be achieved for these iterative algorithms. In addition, the data communication cost can be significantly reduced.

EXISTING SYSTEM

In most existing service reference systems, the data analysis is performed in the task level analysis. This will take the entire data into process and write the output data back in the disk after completing the process. Disadvantages of existing system is no intermediate data is saved in the disk and if the task fails then it needs to start from the beginning again.

PROPOSED WORK

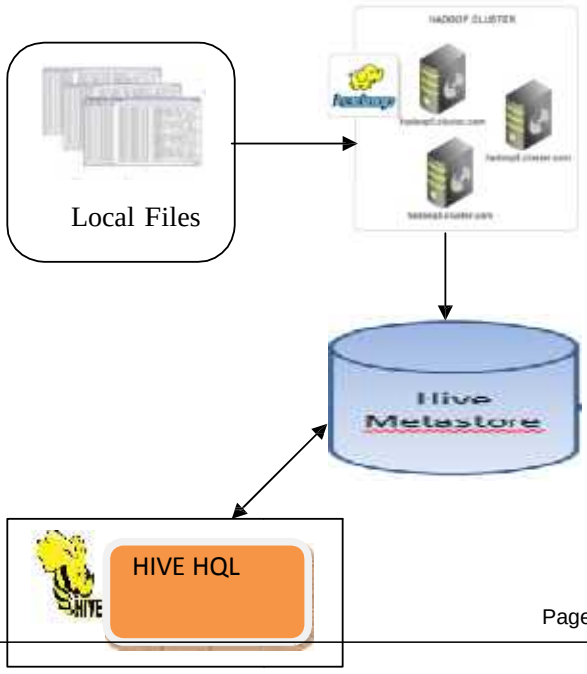
Due to some disadvantages in the current existing system, we propose the new trend of system as below,

- In proposed model it will handle the map reduce in incremental fashion.
- It can store the data intermediately.
- Data are stored in the local disk after each sub task.
- It ensure that the data is not lost if the process fails also.

.

The Intermediate data will be stored in the disk which can be used later to start from those steps if needed. Data is more stable. Accuracy of the task is improved and the data analysis is not only performed in the task level analysis.

SYSTEM ARCHITECTURE



The Design Phase is where many potential solutions are looked upon and the choices are narrowed down to determine the most effective and efficient way to construct the solution. The Design Phase answers the questions about "how" you will build the best solution. Design phase focus on the transformation of all requirements into detailed covering all aspects of the system, Assessment and planning for security risks and Approval to progress to the Development Phase. The requirements identified in the Requirements Analysis Phase are transformed into a System Design Document that accurately describes the design of the system and that can be used as an input to system development in the next phase

MODULES

- Setting up the hadoop Cluster
- Moving of datasets from local to hadoop
- Create the hive Meta Store
- Create Hive DDL for the data in HDFS.
- Hive Hql to perform the MR Analysis.

FINE-GRAIN INCREMENTAL PROCESSING

A vertex in the Map task represents an individual Map function call instance on a pair of (K1; V 1). Each vertex in the Reduce task represents an individual Reduce function call instance on a group of (K2; ({V2}). MRBGraph edges are the fine-grain states M that we would like to preserve for incremental processing. An edge contains three pieces of information: (i) the source Map instance, (ii) the destination Reduce instance (as identified by K2), and (iii) the edge value (i.e., V 2). Map input key K1 may not be unique, i2MapReduce generates a globally unique Map key MK for each Map instance. Therefore, i^2MapReduce will preserve (K2, MK, V 2) for each MRBGraph edge. For incremental processing, we preserve the fine-grain MRBGraph edge states Usually it's consider latter because during incremental processing original intermediate values can be obtained at the Reduce side without any shuffling overhead. The engine transfers the globally unique MK along with (K2; V2)during the shuffle phase. Then it saves the states (K2;MK;V2) in a MRBGraph file at every Reduce task, Delta input. i^2MapReduce expects delta input data that contains the newly inserted, deleted, or modified kv-pairs as the input to incremental processing.Many incremental data acquisition or incremental crawling techniques have been developed to improve data collection performance.

Incremental map computation to obtain the delta MRBGraph. The engine invokes the Map function for every record in the delta input. For an insertion with '+', its intermediate results (K2;MK V2')is represent newly inserted edges in the MRBGraph. For a deletion with '-', its intermediate results indicate that the corresponding edges have been removed from the MRBGraph. The engine replaces the V 20s of the deleted MRBGraph edges with '-'. During the Map-Reduce shuffle phase, the intermediate (K2;MK;V2)is and (K2;MK;)'_'is with the same K2 will be grouped together. The delta MRBGraph will contain only the changes to the MRBGraph and sorted by the K2 order.

Incremental reduce computation. The engine merges the delta MRBGraph and the preserved MRBGraph to obtain the updated MRBGraph using the Query algorithm . For each (K2;MK '-'), the engine deletes the corresponding saved edge state. For each (K2;MK;V2'), the engine first checks duplicates, and inserts the new edge if no duplicate exists, or else updates the old edge if duplicate exists. (Note that (K2, MK) uniquely identifies a MRBGraph edge.) Since an update in the Map input is represented as a deletion and an insertion, any modification to the intermediate edge state (e.g., (2; 0; '*') in the example) consists of a deletion (e.g., (2; 0;'_') followed by an insertion (e.g., (2; 0; 0:6). For each affected K2, the merged list of V 2 will be used as input to invoke the Reduce function to generate the updated final results.

Fine-grain state retrieval and merging.

A MRBGraph file stores fine-grain intermediate states for a Reduce task, a chunk corresponds to the input to a Reduce instance, our design treats chunk as the basic unit, and always reads, writes, and operates on entire chunks. Every record represents a change in the original (last preserved) MRBGraph. There are two kinds of records. An edge insertion record (in green color) contains a valid V 2 value; an edge deletion record (in red color) contains a null value (as marked by '_'). The merging of the delta MRBGraph with the MRBGraph file in the MRBG-Store is essentially a join operation using K2 as the join key. Since the size of the delta MRBGraph is typically much smaller than the MRBGraph file, it is wasteful to read the entire MRBGraph file. Therefore, we construct an index for selective access to the MRBGraph file: Given a K2, the index returns the chunk position in the MRBGraph file. As only point lookup is required, we employ a hash-based implementation for the index. The index is stored in an index file and is preloaded into memory before Reduce computation. We apply the index nested loop join for the merging operation.It has been observe that the MapReduce shuffling phase will sort the intermediate keys. An introduction to a read cache and a dynamic read window technique for further optimization. A sequence of K2s, there are two ways to read the corresponding chunks: (i) performing an individual I/O operation for each chunk; or (ii) performing a large I/O that covers all the required chunks. The former may lead to frequent disk seeks, while the latter may result in reading a lot of useless data.

CONCLUSION

The proposed strategy have described i^2MapReduce, a MapReduce-based framework for incremental big data processing. i^2 MapReduce combines a fine-grain incremental engine, a general-purpose iterative model, and a set of effective techniques for incremental iterative computation. Real-machine experiments show that i^2 MapReduce can significantly reduce the run time for refreshing big data mining results compared to re-computation on both plain and iterative MapReduce. This i^2 will save the time by saving the intermediate data in the disk and therefore when the operation fails the previous process data will be in the disk and later it can be used for processing.

REFERENCES

1. D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications," in Proc. 9th USENIX Conf. Oper.Syst.Des.Implementation, 2010, pp. 115

2. Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: Efficient iterative data processing on large clusters," in Proc. VLDB Endowment, 2010, vol. 3, no. 1–2, pp. 285–296.

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

4. 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 Comput., 2012, pp.pp. 534–541.

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. Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," in Proc. VLDB Endowment, 2012, vol. 5, no. 8, pp. 716–727.

7. T. Jorg, R. Parvizi, H. Yong, and S. Dessloch, "Incremental recomputations in mapreduce," in Proc. 3rd Int. Workshop Cloud Data Manage., 2011, pp. 7–14.

8. J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: A runtime for iterative mapreduce," in Proc. 19th ACM Symp. High Performance Distributed Comput., 2010, pp. 810–818.

9. S. R. Mihaylov, Z. G. Ives, and S. Guha, "Rex: Recursive, deltabased data-centric computation," in Proc. VLDB Endowment, 2012, vol. 5, no. 11, pp. 1280–1291.

10. R. Power and J. Li, "Piccolo: Building fast, distributed programs partitioned tables," in Proc. 9th USENIX Conf. Oper. Syst. Des. Implementation, 2010, pp. 1–14.

11. G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2010, pp. 135–146.

12. S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl, "Spinning fast iterative data flows," in Proc. VLDB Endowment, 2012, vol. 5,no. 11, pp. 1268–1279.

13. S. Brin, and L. Page, "The anatomy of a large-scale hypertextual web search engine," Comput. Netw. ISDN Syst., vol. 30, no. 1–7, pp. 107–117, Apr. 1998.

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

15. Y. Zhang, Q. Gao, L. Gao, and C. Wang "Acceleratelarge-scale iterative computation through asynchronous accumulative updates," in Proc. 3rd Workshop Sci. Cloud Comput. Date, 2012, pp. 13–22.

16. Y. Zhang, Q. Gao, L. Gao, and C. Wang, "Priter: A distributed framework for prioritized iterative computations," in Proc. 2nd ACM Symp. Cloud Comput., 2011, pp. 13:1–13:14.

Volume: 6 Issue: 3 Dec,2016,ISSN_NO: 2321-3337