ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

IJARMATE
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2,Special Issue 6, March 2016

i2MapReduce: Incremental Map Reduce for Mining Evolving Big Data

ANNAL.D, V.S ARCHANA, R.DHASS, R.RAJASHREE

AbstractAs new data and updates are constantly arriving, the results of data mining applications become stale and obsolete overtime. Incremental processing is a promising approach to refreshing mining results. It utilizes previously saved states to avoid the expense of re-computation from scratch. In this paper, we propose i2 MapReduce, a novel incremental processing extension to MapReduce, the most widely used framework for mining big data. Compared with the state-of-the-art work on Incoop, i2 MapReduce (i) performs key-value pair level incremental processing rather than task level re-computation, (ii) supports not only one-step computation but also more sophisticated iterative computation, which is widely used in data mining applications, and (iii) incorporates a set of novel techniques to reduce I/O overhead for accessing preserved fine-grain computation states. We evaluate i2MapReduce using a one-step algorithm and four iterative algorithms with diverse computation characteristics. Experimental results on Amazon EC2 show significant performanceimprovements of i2MapReduce compared to both plain and iterative MapReduce performing re-computation

I.INTRODUCTION

Today amount of digital data is being accumulatedin many important areas, including e-commerce, socialnetwork, finance, health care, education, and environment.It has become increasingly popular to mine such big data inorder to gain insights to help business decisions or to providebetter personalized, higher quality services.Inmany situations, it is desirable to periodically refresh themining computation in order to keep the mining resultsup-to-date. For example, the PageRank algorithmcomputes ranking scores of web pages based on the webgraph structure for supporting web search. However, theweb graph structure is constantly evolving; Web pagesand hyper-links are created, deleted, and updated. As theunderlying web graph evolves, the PageRank rankingresults gradually become stale, potentially lowering thequality of web search.

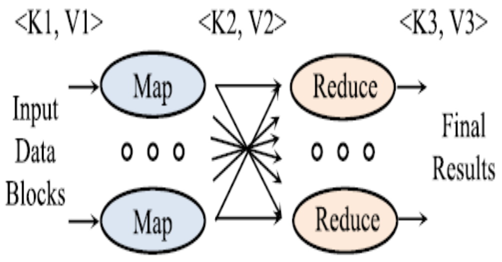

Fig. 1. MapReduce computation.

Incremental processing is a promising approach torefreshing mining results. Given the size of the input bigdata, it is often very expensive to rerun the entire computationfrom scratch. Incremental processing exploits the factthat the input data of two subsequent computations A and Bare similar. Only a very small fraction of the input data haschanged. The idea is to save states in computation A, re-useA's states in computation B, and perform re-computationonly for states that are affected by the changed input data. Inthis paper, we investigate the realization of this principlein the context of the MapReduce computing framework.

II.LITERATURE SURVEY

Big data is constantly evolving. As new data andupdates are being collected, the input data of a big data mining algorithm will gradually change, and the computedresults will become stale and obsolete over time. Inmany situations, it is desirable to periodically refresh themining computation in order to keep the mining resultsup-to-dateMapReduce-based frameworkfor incremental big data processing. MapReduce combinesa fine-grain incremental engine, a general-purposeiterative model, and a set of effective techniques for increment MapReduce reschedules the failed Map/Reducetask in case task failure is detected. However, the interdependencyof prime Reduce tasks and prime Map tasks inMapReduce requires more complicated fault-tolerancesolution. i2MapReduce checkpoints the prime Reduce task'soutput state data and MRBGraph file on HDFS To the bestof our knowledge, the task-level coarse-grain incrementalprocessing system,

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

IJARMATE
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2,Special Issue 6, March 2016

Incoopis not publicly available.Therefore, we cannot compare i2 MapReduce with Incoop.Nevertheless, our statistics show that without carefuldata partition, almost all tasks see changes in the experiments,making task-level incremental processing lesseffective.

Disadvantagesof Existing System

- Task-level incremental processing less effective.
- Plain and iterative MapReduce performing re-computation.
- MapReduce re-computation takes long time.
- Performance is Low inRuntime.

III.PROPOSED SYSTEM

MapReduce, a novel incremental processing extension to MapReduce, themost widely used framework for mining big data. Compared with the state-of-the-art work on Incoop, i2MapReduce performs key-valuepair level incremental processing rather than task level re-computation, supports not only one-step computation but also moresophisticated iterative computation, which is widely used in data mining applications, and incorporates a set of novel techniques toreduce I/O overhead for accessing preserved fine-grain computation states. We evaluate i2MapReduce using a one-step algorithm andfour iterative algorithms with diverse computation characteristics. It show significant performanceimprovements of i2MapReduce compared to both plain and iterative MapReduce performing re-computation.we propose a general-purpose MapReduce model for iterative computation and describe how to efficiently support this computationi2MapReduce must transfer the updated state kv-pairs to their corresponding prime Map task, which caches theirdependent structure kv-pairs in its local file system. Real-machine experiments showthat i2 MapReduce can significantly reduce the run time for refreshing big data mining results compared to re-computationon both plain and iterative MapReduce.

Advantages of Proposed System

It performs key-value pair level incremental processing.It supports one-step computation and moresophisticated iterative computation.Performance is very high in Runtime.

Basic Idea

Consider two MapReduce jobs A and A0 performing the same computation on input data set D and D0, respectively. D0 ¼ D þ DD, where DD consists of the inserted and deleted input hK1; V 1is1. An update can be represented as a deletion followed by an insertion. Our goal is to recompute only the Map and Reduce function call instances that are affected by DD. Incremental computation for Map is straightforward. We simply invoke the Map function for the inserted or deleted hK1; V 1is. Since the other input kv-pairs are not changed,their Map computation would remain the same. We now have computed the delta intermediate values, denoted DM,including inserted and deleted hK2; V2is.To perform incremental Reduce computation, we need to save the fine-grain states of job A, denoted M, which includes hK2; fV 2gis. We will recompute the Reduce function for each K2 in DM. The other K2 in M does not see anychanged intermediate values and therefore would generate the same final result. For a K2 in DM, typically only a subset of the list of V 2 have changed. Here, we retrieve the saved hK2; fV 2gi from M, and apply the inserted and/or deleted values from DM to obtain an updated Reduce input. We then re-compute the Reduce function on this input to generate the changed final results hK3; V 3is.It is easy to see that results generated from this incremental computation are logically the same as the results from completely re-computing A0.

Incremental iterative processing

In this section, we present incremental processing techniquesfor iterative computation. Note that it is not sufficientto simply combine the above solutions for incremental onestep processing and iterative computation. In the following, we discuss three aspects thatwe address in order to achieve an effective design.

Fault-Tolerance

Vanilla MapReduce reschedules the failed Map/Reduce task in case task failure is detected. However, the interdependency of prime Reduce tasks and prime Map tasks in i2MapReduce requires more complicated fault-tolerance solution. i2MapReduce checkpoints the prime Reduce task's output state data and MRBGraph file on HDFS in every iteration.Upon detecting a failure, i2MapReduce recovers by considering task dependencies in three cases. (i) In case a prime Map task fails, the master reschedules the Map task on the worker where its dependent Reduce taskresides. The prime Map task reloads the its structure data and resumes computation from its dependent state data(checkpoint). (ii) In case a prime Reduce task fails, the master reschedules the Reduce task on the worker where its dependent Map task resides. The prime Reduce task reloads its MRBGraph file (checkpoint) and resumes computation by

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

IJARMATE
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2,Special Issue 6, March 2016

re-collecting Map outputs. (iii) In case a worker fails, the master reschedules the interdependent prime Map task and prime Reduce task to a healthy worker together. The prime Map task and Reduce task resume computation based on the checkpointed state data and MRBGraph file as introduced above.

Reducing Change Propagation

In incremental iterative computation, changes in the delta input may propagate to more and more kv-pairs as the computation iterates. For example, in PageRank, a change thataffects a vertex in a web graph propagates to the neighborvertices after an iteration, to the neighbors of the neighborsafter two iterations, to the three-hop neighbors after threeiterations, and so on. Due to this effect, incremental processingmay become less effective after a number of iterations.

To address this problem, i2MapReduce employs a changepropagation control technique, which is similar to the dynamiccomputation in GraphLab [6]. It filters negligible changes of state kvpairs that are below a given threshold. These filteredkv-pairs are supposed to be very close to convergence. Onlythe state values that see changes greater than the thresholdare emitted for next iteration. The changes for a state kv-pairare accumulated. It is possible a filtered kv-pair may later beemitted if its accumulated change is big enough.The observation behind this technique is that iterativecomputation often converges asymmetrically: Many statekv-pairs quickly converge in a few iterations, while theremaining state kv-pairs converge slowly over many iterations.

Mapreduce Background

The Reduce function takes a K2 and a list of fV 2g as input and computes the final output kv-pairs hK3; V 3is.A MapReduce system (e.g., Apache Hadoop) usually reads the input data of the MapReduce computation from and writes the final results to a distributed file system(e.g., HDFS), which divides a file into equal-sized(e.g., 64 MB) blocks and stores the blocks across a cluster of machines. For a MapReduce program, the MapReduce system runs a JobTracker process on a master node to monitor the job progress, and a set of TaskTracker processes on worker nodes to perform the actual Map and Reduce tasks.TheJobTracker starts a Map task per data block, and typically assigns it to the TaskTracker on the machine that holds the corresponding data block in order to minimize communication overhead. Each Map task calls the Map function for every input hK1; V 1i, and stores the intermediate kv-pairs hK2; V 2is on local disks. Intermediate results are shuffled to Reduce tasks according to a partition function (e.g., a hash function) on K2. After a Reduce task obtains and merges intermediate results from all Map Tasks, it invokes the Reduce function oneach hK2; fV 2gi to generate the final output kv-pairs hK3; V 3is.For a MapReduce program, the MapReducesystem runs a JobTracker process on a master node tomonitor the job progress, and a set of TaskTracker processeson worker nodes to perform the actual Map andReduce tasks.TheJobTracker starts a Map task per data block, and
typically assigns it to the TaskTracker on the
machinethat holds the corresponding data block in
order to minimizecommunication overhead.

MRBG-Store

The MRBG-Store supports the preservation and retrieval offine-grain MRBGraph states for incremental processing. Wesee two main requirements on the MRBG-Store. First, theMRBG-Store must incrementally store the evolvingMRBGraph. Consider a sequence of jobs that incrementallyrefresh the results of a big data mining algorithm. As inputdata evolves, the intermediate states in the MRBGraphwill also evolve. It would be wasteful to store the entireMRBGraph of each subsequent job. Instead, we would like to obtain and store only the updated part of the MRBGraph.Second, the MRGB-Store must support efficient retrieval of preserved states of given Reduceinstances. For incremental Reduce computation, i2MapReduce re-computes the Reduce instance associated with each changed MRBGraph edge, as described in Section 3.3. For a changed edge, it queries the MRGB-Store to retrieve the preserved states ofthe in-edges of the associated K2, and merge the preserved states with the newly computed edge changes.

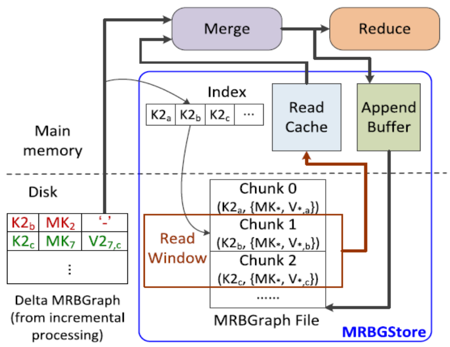

Fig. 4. Structure of MRBG-store.

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com

IJARMATE
Your uh-MATE Research Paper !!!

International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE) Vol. 2,Special Issue 6, March 2016

Extending MRBG-Store for Multiple Iterations

As described previously in Section 3.4, MRBG-Store appends newly computed chunks to the end of the MRBGraph file
and updates the chunk index to reflect the new positions.Obsolete chunks are removed offline when the worker machine is idle. In an incremental iterative job, every iteration
will generate newly computed chunks,which are sorted due to the MapReduce shuffling phase. Consequently, the MRBGraph file will consist of multiple batches of sorted
chunks, corresponding to a series of iterations. If a chunk exists in multiple batches, a retrieval request returns the latest
version of the chunk (as pointed to by the chunk index).

Optimization for Special Accumulator Reduce

This property allows us to process the two data set D and DD separately and then to simply combine the results by the '_' operation to obtain the full result. We call this kind of Reduce function accumulator Reduce. For this special case, it is not necessary to preserve the MRBGraph.Then it simply invokes the accumulator Reduce to accumulate changes to the result kv-pairs.ManyMapReduce algorithms employ accumulator Reduce. A well-known example is WordCount. The Reduce function of WordCount computes the count of word appearances using an integer sum operation, which satisfies the above property. Other common operations that directly satisfy the distributive property include maximum and minimum. Moreover, some operations can be easily modified to satisfy the requirement of accumulator Reduce.

For example, average is computed as dividing sum by count. While it is not possible to combine two averages into a single average, we can modify the implementation to allow/produce a partial sum and a partial count in the function input and the output. Then the implementation can accumulatepartial sums and partial counts in order to compute the average of the full data set.

General-Purpose Iterative MapReduce Model

In general, the improvements focus on two aspects: Reducing job startup costs. In vanilla MapReduce, every algorithm iteration runs one or several MapReduce jobs. Note that Hadoop may take over 20 seconds to start a job with 10–100 tasks. If the computation of each iteration is relatively simple, job startup costs may consist of an overly large fraction of the run time. The solution is to modify MapReduce to reuse the same jobs across iterations, and killthem only when the computation completes.Caching structure data. Structure data is immutableduring computation. It is also much larger than state data in many applications (e.g., PageRank, Kmeans,and GIM-V). Therefore, it is wasteful to transfer structure data over and over again in every iteration.An optimization is to cache structure data in local file systems to avoid the cost of network communication and reading from HDFS.

IV.CONCLUSION

Wehave described i2MapReduce, a MapReduce-based framework for incremental big data processing. i2 MapReduce combinesa fine-grain incremental engine, a general-purposeiterative model, and a set of effective techniques for incrementaliterative computation.Real-machine experiments showthat i2 MapReduce can significantly reduce the run time forrefreshing big data mining results compared to re-computation on both plain and iterative MapReduce.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processingon large clusters," in Proc. 6th Conf. Symp. Opear. Syst. Des.Implementation, 2004, p. 10.
[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley,M. J. Franklin, S.Shenker, and I. Stoica, "Resilient distributeddatasets: A fault-tolerant abstraction for, in-memory cluster computing," in Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation,2012, p. 2.
[3] R. Power and J. Li, "Piccolo: Building fast, distributed programswith partitioned tables," in Proc. 9th USENIX Conf. Oper. Syst. Des.Implementation, 2010, pp. 1–14.
[4] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn,N. Leiser, and G. Czajkowski, "Pregel: A system for large-scalegraph processing," in Proc. ACM SIGMOD Int. Conf. Manage. Data,2010, pp. 135–146.