INCREMENTAL MAPREDUCE FOR MINING EVOLVING BIG DATA

P.ARUMUGAM*1 P.DHINESH KUMAR*2 and M.PRIYA*3

*1 M.Phil Scholar, Sri Vasavi College, Erode, Tamilnadu, India.
*2 Assistant Professor (BCA), Dr.RANM Arts & Science College, Erode, Tamilnadu, India.
*3 Assistant Professor (CS), Sri Vasavi College (SFW), Erode, Tamilnadu, India

Abstract—As new data and updates are constantly arriving, the results of data mining applications become stale and obsolete over time. 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 i2MapReduce, 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 Incoo, i2MapReduce (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 three iterative algorithms with diverse computation characteristics. Experimental results on Amazon EC2 show significant performance improvements of i2MapReduce compared to both plain and iterative MapReduce performing re-computation.

Index Terms—MapReduce, hadoop, pair level, incremental processing

I. INTRODUCTION

In typical data mining systems, the mining procedures require computational intensive computing units for data analysis and comparisons. A computing platform is, therefore, needed to have efficient access to, at least, two types of resources, data and computing processors. A Big Data processing framework in research challenges form a three tier structure and center around the “Big Data mining platform” which focuses on low-level data accessing and computing. Challenges on information sharing and privacy and Big Data application domains and knowledge form Tier II, which concentrates on high-level Semantics, application domain knowledge and user privacy issues.

The outmost circle shows Tier III challenges on actual mining algorithms to fulfil the data mining goals. Indeed, many data mining algorithm are designed for this type of problem settings. For medium scale data mining tasks, data are typically large and cannot be fit into the main memory. Common solutions are to rely on parallel computing or collective mining to sample and aggregate data from different sources and then use parallel computing programming to carry out the mining process.

In Big Data mining, data scale is far beyond the capacity that a single personal computer can handle, a typical Big Data processing framework will rely on cluster computers with a high-performance computing platform, with a data mining task being deployed by running some parallel programming tools, such as Map Reduce or Enterprise Control Language, on a large number of computing nodes. The role of the software component is to make sure that a single data mining task, such as finding the best match of a query from a database with billions of records, is split into many small tasks each of which is running on one or multiple computing nodes.

Such a Big Data system, which blends both hardware and software components, is hardly available without key industrial stockholder’s support. In fact, for decades, companies have been making business decisions based on transactional data stored in relational databases. Big Data mining offers opportunities to go beyond traditional relational databases to rely on less structured data weblogs, social media, e-mail, sensors, and photographs that can be mined for useful information. Major business intelligence companies, such IBM, Oracle, Teradata, and so on, have all featured their own products to help customers acquire and organize these diverse data sources and coordinate with customer’s existing data to find new insights and capitalize on hidden relationships.

II. RELATED WORK

A. Fast Algorithms for Mining Association Rules

User considers the problem of discovering association rules between items in a large database of sales transactions. User present two new algorithms for solving this problem that is fundamentally different from the known algorithms. Empirical evaluation shows that these algorithms outperform the known algorithms by factors ranging from three for small problems to more than an order of magnitude for large problems. User also show how the best features of the two proposed algorithms can be combined into a hybrid algorithm, called AprioriHybrid. Scale-up experiments show that AprioriHybrid scales linearly with the number of transactions. AprioriHybrid also has excellent scale-up properties with respect to the transaction size and the number of items in the database.
B. The Anatomy Of A Large-Scale Social Search Engine

User presents Aardvark, a social search engine. With Aardvark, users ask a question, either by instant message, email, web input, text message, or voice. Aardvark then routes the question to the person in the user’s extended social network most likely to be able to answer that question. As compared to a traditional web search engine, where the challenge lies in finding the right document to satisfy a user’s information need, the challenge in a social search engine like Aardvark lies in finding the right person to satisfy a user’s information need. Further, while trust in a traditional search engine is based on authority, in a social search engine like Aardvark, trust is based on intimacy. We describe how these considerations inform the architecture, algorithms, and user interface of Aardvark, and how they are reflected in the behaviour of Aardvark users.

III. PROPOSED SCHEME OF WORK

A number of studies have followed the principle and designed new programming models to support incremental processing. Among the new frameworks, MapReduce is the most widely used in production because of its simplicity, generality, and maturity. User focus on improving MapReduce in this journal. User propose to reuse the converged state from the previous computation and employ a change propagation control mechanism.

User proposes i2MapReduce, an extension to MapReduce that supports fine-grain incremental processing for both one-step and iterative computation. Compared to previous solutions, modification is modest compared to the effort to re-implement algorithms on a completely different programming paradigm.

- User propose to reuse the converged state from the previous computation and employ a change propagation control mechanism.
- i2MapReduce is the first MapReduce-based solution that efficiently supports incremental iterative computation.

B. MRBGraph ABSTRACTION

User use a MRBGraph abstraction to model the data flow in MapReduce. Each vertex in the Map task represents an individual Map function call instance on a pair of hK1, V 1i. Each vertex in the Reduce task represents an individual Reduce function call instance on a group of hK2, {V 2}i. An edge from a Map instance to a Reduce instance means that the Map instance generates a hK2, V 2i that is shuffled to become part of the input to the Reduce instance. The input of Reduce instance a comes from Map instance 0, 2, and 4. MRBGraph edges are the fine-grain states M that user would like to preserve for incremental processing. An edge contains three pieces of information: (i) the source Map in-21. Incremental data acquisition can significantly save the resources for data collection; it does not re-capture the whole data set but only capture the revisions since the last time that data was captured. stance, (ii) the destination Reduce instance (as identified by K2), and (iii) the edge value (i.e. V 2). Since Map input key K1 may not be unique, i2MapReduce generates a globally unique Map key MK for each Map instance. Therefore, i2MapReduce will preserve (K2, MK, V 2) for each MRBGraph edge.

C. MRBG-Store

The MRBG-Store supports the preservation and retrieval of fine-grain MRBGraph states for incremental processing. User sees two main requirements on the MRBG-Store. First, the MRBG-Store must incrementally store the evolving MRBGraph. Consider a sequence of jobs that incrementally refresh the results of a big data mining algorithm. As input data evolves, the intermediate states in the MRBGraph will also evolve. It would be wasteful to store the entire MRBGraph of each subsequent job. Instead, user would like to obtain and store only the updated part of the MRBGraph. Second, the MRBG-Store must support efficient retrieval of preserved states of given Reduce instances. For incremental Reduce computation, i2MapReduce re-computes the Reduce instance associated with each changed MRBGraph edge. For a changed edge, it queries the MRBG-Store to retrieve the preserved states of the in-edges of the associated K2, and merge the preserved states with the newly computed edge changes.

D. PAGERANK

PageRank is a well-known iterative graph algorithm for ranking web pages. It computes a ranking score for each vertex in a graph. After initializing all ranking scores, the computation performs a MapReduce job per iteration. i and j are vertex ids, Ni is the set of out-neighbor vertices of i, Ri is i’s ranking score that is updated iteratively. ‘|’ means
concatenation. All Ri’s are initialized to one. The Reduce
instance on vertex j updates Rj by summing the Ri,j received
from all its in-neighbors i, and applying a damping factor d.

Map Phase input: < i, Ni|Ri >
1: output < i, Ni >
2: for all j in Ni do
3: Ri,j = Ri |Ni|
4: output < j, Ri,j >
5: end for
Reduce Phase input: < j, {Ri,j ,Nj} >
6: Rj = dPi Ri,j + (1 − d)
7: output < j, Nj |Rj >
Algorithm PageRank in MapReduce

E. KMEANS

Kmeans is a commonly used clustering algorithm that
partitions points into k clusters. User denote the ID of a point
as pid, and its feature values pval. The computation starts with
selecting k random points as cluster centroids set {cid, cval}.
As shown in Algorithm 3, in each iteration, the Map instance
on a point pid assigns the point to the nearest centroid. The
Reduce instance on a centroid cid updates the centroid by
averaging the values of all assigned points {pval}.

Map Phase input: < pid, pval|{cid, cval} >
1: cid ← find the nearest centroid of pval in {cid, cval}
2: output < cid, pval > Reduce Phase input: < cid, {pval} >
3: cval ← compute the average of {pval}
4: output < cid, cval >
Algorithm Kmeans in MapReduce

F. GIM-V

Generalized Iterated Matrix-Vector multiplication (GIM-V)
is an abstraction of many iterative graph mining operations.
These graph mining algorithms can be generally represented
by operating on an n x n matrix M and a vector v of size n.
Suppose both the matrix and the vector are divided into
sub-blocks. Let mi,j denote the (i, j)-th block of M and vj
denote the j-th block of v. The computation steps are similar
to those of the matrix-vector multiplication and can be
abstracted into three operations: (1) mvi,j = combine2(mi,j ,
vj); (2) v'i = combineAll({mvi,j}); and (3) vi = assign(vi, v'i).

User can compare combine2 to the multiplication between
mi,j and vj, and compare combineAll to the sum of mvi,j for
row i. Algorithm 4 shows the MapReduce implementation
with two jobs for each iteration. The first job assigns vector
block vj to multiple matrix blocks mi,j (Vi) and performs
combine2(mi,j , vj ) to obtain mvi,j . The second job groups
the mvi,j and vi on the same i, performs the
combineAll({mvi,j}) operation, and updates vi using assign
(vi , v'i).

Map Phase 1 input: < (i, j),mi,j > or < j, vj >
1: if kv-pair is < (i, j),mi,j > then
2: output < (i, j),mi,j >
3: else if kv-pair is < j, vj > then
4: for all i blocks in j’s row do
5: output < (i, j), vj >
6: end for
Reduce Phase 1 input: < (i, j), {mi,j , vj} >
8: mvi,j = combine2(mi,j , vj )
9: output < i, mvi,j >, < j, vj >
Map Phase 2: output all inputs
Reduce Phase 2 input: < i, {mvi,j , vi} >
10: v'i ← combineAll({mvi,j})
11: vi ← assign(vi, v'i)
12: output < i, vi >
Algorithm GIM-V in MapReduce

G. API CHANGES TO MAPREDUCE

User implement a prototype of i2MapReduce by modifying
Hadoop-1.0.3. In order to support incremental and iterative
processing, a few MapReduce APIs are changed or added.
User summarize these API changes in Table 2. User briefly
explain the key APIs and their usage in this section.

• For incremental one-step processing, programmers
need to specify the delta input, in which the inserted and
deleted input kv-pairs are marked with ‘+’ and ‘−’,
respectively.

• For the special case of accumulator reduce, an
accumulate function needs to be specified, which aggregates
reducer input values with the same key.

• For iterative computation, programmers must specify
the structure kv-pairs hSK, SV i, the state kv-pairs hDK,DV i,
and the Project function. Besides, a new mapper interface
should be implemented, and the new map function will take
both the structure and state kv-pairs as input. The initial state
value DV should also be set.

IV. PERFORMANCE STUDY

A. Incremental one-step processing:
User use APriori to understand the benefit of incremental
one-step processing in i2MapReduce. MapReduce
re-computation takes 1608 seconds. In contrast, i2MapReduce takes only 131 seconds. Fine-grain
incremental processing leads to a 12x speedup.

B. Incremental Iterative Processing:
The above graph shows the normalized runtime of the four iterative algorithms while 10% of input data has been changed. “1” corresponds to the runtime of PlainMR recomputation. For PageRank, iterMR reduces the runtime of PlainMR recomputation by 56%. The main saving comes from the caching of structure data and the saving of the MapReduce tuple costs. i2MapReduce improves the performance further with fine-grain incremental processing and change propagation control (CPC), achieving a speedup of 8 folds (i2MR w/o CPC). Users also show that without change propagation control, the changes will return the exact updated result but at the same time prolong the runtime (i2MR w/o CPC). The change propagation control technique is critical to guarantee the performance. Section 8.5 will discuss the effect of CPC in more details.

On the other hand, it is surprising to see that HaLoop performs worse than plain MapReduce. This is because HaLoop employs an extra MapReduce job in each iteration to join the structure and state data. The profit of caching cannot compensate for the extra cost when the structure data is not big enough. Note that the iterative model in i2MapReduce avoids this overhead by exploiting the Project function to co-partition structure and state data. For SSSP, the performance gain of i2MapReduce is similar to that for PageRank. Users set the filter threshold to 0 in the change propagation control. That is, nodes without any changes will be filtered out. Therefore, unlike PageRank, the SSSP results with CPC are precise. For K-means, small portion of changes in input will lead to global re-computation. Therefore, users turn off the MRBGraph functionality. As a result, i2MapReduce falls back to iterMR recomputation. Users see that HaLoop and iterMR exhibit similar performance. They both outperform plainMR because of similar optimizations, such as caching structure data. For GIM-V, both plainMR and HaLoop run two MapReduce jobs in each iteration, one of which joins the structure data and the state data.

In contrast, our general-purpose iterative support removes the need for this extra job. iterMR and i2MapReduce see dramatic performance improvements. i2MapReduce achieves a 10.3x speedup over plainMR, and a 1.4x speedup over HaLoop.

V. CONCLUSION AND FUTURE WORK
i2MapReduce achieves better performance when input data is large. Pregel follows the Bulk Synchronous Processing (BSP) model. The computation is broken down into a sequence of supersteps. In each superstep, a Compute function is invoked on each vertex. It communicates with other vertices by sending and receiving messages and performs computation for the current vertex. This model can efficiently support a large number of iterative graph algorithms. Open source implementations of Pregel include Giraph, Hama, and Pregelix. Compared to i2MapReduce, the BSP model in Pregel is quite different from the MapReduce programming paradigm. It would be interesting future work to exploit similar ideas in this thesis to support incremental processing in Pregel-like systems.

Users have described i2MapReduce, a MapReduce-based framework for incremental big data processing. i2MapReduce 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 i2MapReduce can significantly reduce the run time for refreshing big data mining results compared to recomputation on both plain and iterative MapReduce.

REFERENCES