I2 MAP REDUCE: INCREMENTAL MAP REDUCE FOR EFFICIENT MINING IN BIG DATA

Dr.T.POONGOTHAI, M.E., P.hD., (1)

B DHAVAPRIYA[2], M SUJIDA DEVI[3], S VIDHYA[4], S VIJAY SRINIVASAN[5]

Associate Professor[1], UG Scholar[2, 3, 4, 5]

Department of Information Technology

K.S.R College of Engineering, Namakkal, Tamilnadu, India.

ABSRACT

New data and updates are constantly arriving , the result of data mining applications become stale and obsolete over time.Incremental processing is a promising approach to refreshing mining result which is widely used for large scale and one-time data-intensive distributed computing , but lacks flexibility and efficiency of processing small incremental data. I^{2}MapReduce 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 fromdesignate nodes or local nodes.

Key words: *iterative computation,I map reduce,big data.*

1.INTRODUCTION

Incremental Map Reduce is also called **I^2 Map Reduce** which is based on the concept of both plain and iterative Map Reduce performing **Re-computation.** Map Reduce enables easy development of scalable parallel applications to process immense amount of data on large clusters of commodity machines.

It supports not only one step computation but also more sophisticated iterative computation which is used in Data mining.I^2 Map Reduce makes some contributions such as Incremental Map Reduce framework, Dynamic resource allocation based on the state, Friendly APIs for application.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 fromscratch. Incremental processing exploits the fact that theinput data of two subsequent computations A and B aresimilar. 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-computation only for states that are affected by the changed input data.In this paper, we investigate the realization of this principlein the context of the MapReduce computing framework.

A number of previous studies (including Percolator [22],CBP [16], and Naiad [20]) have followed this principle and designed new programming models to support incrementa lprocessing. 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 programmers to completely re-implement their algorithms. On the other hand, Incoop [4] extends MapReduce to support incremental processing. However, it has two main limitations. First, Incoop supports only task-level incremental processing. That is, it saves and reuses states at the granularity of individual Map and Reduce tasks. Each task typically processes a large number of key-value pairs (kvpairs). If Incoop detects any data changes in the input of a task, it will rerun the entire task. While this approach easily leverages existing MapReduce features for state savings,it may incur a large amount of redundant computationif only a small fraction of kv-pairs have changed in a task.Second, Incoop supports only one-step computation, while important mining algorithms, such as PageRank, require iterative computation. Incoop would treat each iteration as a separate MapReduce job. However, a small number of input data changes may gradually propagate to affect a large portion of intermediate states after a number of iterations, resulting in expensive global re-computation afterwards. We propose i2MapReduce, an extension to MapReducethat supports fine-grain incremental processing for both one-step and iterative computation. Compared to previous solutions,i2MapReduce incorporates the following three novel.

• Fine-grain Incremental Processing using MRBG-Store: Unlike Incoop, i2MapReduce supports kv-pair level fine-grain incremental processing in order to minimize the amount of re-computation as much as possible. We model the kv-pair level data flow and data dependence in aMapReduce computation as a bipartite graph, called MRBGraph. AMRBG-Store is designed to preserve the fine-grain states in the MRBGraph and support efficient queries to retrieve fine-grain states for incremental processing.

• General-Purpose Iterative Computation withModest Extension toMapReduce API: Our previous workproposed iMapReduce to efficiently support iterative computation on the MapReduce platform. However, ittargets types of iterative computation where there is a oneto-
one/all-to-one correspondence from Reduce output to Map input. In comparison, our current proposal provides general-purpose support, including not only one-to-one, but also one-to-many, many-to-one, and many-to-many correspondence. We enhance the Map API to allow users to easily express loop-invariant structure data, and we propose a Project API function to express the correspondence from Reduce to Map. While users need to slightly modify their algorithms in order to take full advantage of i2MapReduce, such modification is modest compared to the effort to re-implement algorithms on a completely different programming paradigm, such as in Percolator , and Naiad .

• Incremental Processing for iterative computation: iterative algorithm typically performs the same computation on a data set in every iteration, generating a sequence of improving results. The computation of an iteration can be represented by an update function F: $vk = F(vk-1;D)$;where D is the input data set, and v is the result set being computed. After initializing v with a certain $v0$, the iterative algorithm computes an improved vk from $vk-1$ and D in the k-th iteration.This process continues until it converges to a fixed point $v*$. Inpractice, this means that the difference between the result sets of two consecutive iterations is small enough. Then the iterative computation will return the converged result

$v*$. Note that while v is updated in every iteration, D is static in the computation. We refer to D as the static *structure data*, and v as the dynamic *state data*.
For example, the well-known PageRank algorithm [4] iteratively computes the PageRank vector R that contains the ranking scores
of all pages in a web graph, using the following update function:

$$R(k) = dWR(k-1) + (1 - d)E;$$

Use the same computation logic (update function) to process the data many times
•The previous iteration's output is the next iteration's input .
•Stop when the iterated result converges to a fixed point .

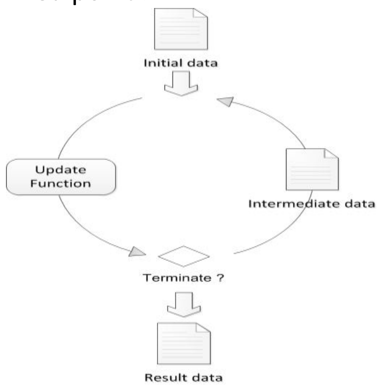


Figure 1:iterative computation.

2. I MAP REDUCE BACKGROUND:

Start from the previously converged state data Reduce the number of iterations Only execute the changed mappers/reducers and utilize the converged MR-Edge/RM-Edge state Reduce the workload of each iteration Filter the converged reducers Avoid changes propagation.

.

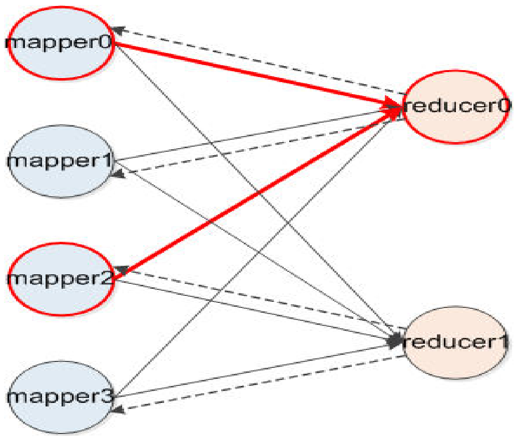
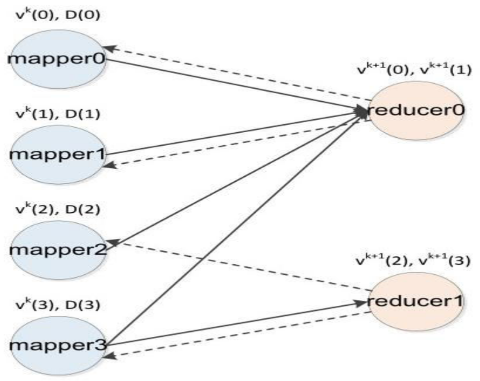


Figure 2:I map reduce.

A I map reduce program is composed of a map function and a reduce function as shown in fig 2.

Building MRBGraph:

Figure 3:MapReduce extension

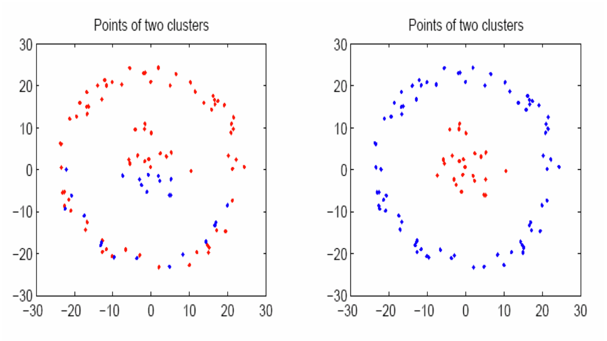


The MRBG-Store supports the preservation and retrieval
of fine-grain MRBGraph states for incremental processing.We see two main requirements on the MRBG-Store. First,
the MRBG-Store must incrementally store the evolving MRBGraph.Consider a sequence of jobs that incrementally refresh
the results of a 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, 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 Reduce instances.

SPECTRAL CLUSTERING:

In multivariate statistics and the clustering of data, **spectral clustering** techniques make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. The similarity matrix is provided as an input and consists of a quantitative assessment of the relative
Figure
Figure 4:spectral clustering vs k-means



K means spectral
similarity of each pair of points in the dataset

FAULT TOLERANCE AND LOAD BALANCING

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 consideringtask dependencies in three cases. (i) In case a primeMap task fails, the master reschedules the Map task on the worker where its dependent Reduce task resides. The primeMap task reloads the its structure data and resumes computation from its dependent state data (checkpoint). (ii) Incase 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 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.

Performance:

We 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.
"1" corresponds to the runtime of PlainMR recomp.For PageRank, iterMR reduces the runtime of PlainMR recomp by 56%. The main saving comes from the caching of structure data and the saving of the MapReduce startup 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). We also show that without change propagation control the changes it 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. The detail comparison with HaLoop is provided.For SSSP, the performance gain of i2MapReduce is similar to that for PageRank. We 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 Kmeans, small portion of changes in input will lead to global re-computation. Therefore, we turn off the MRBGraph functionality. As a result, i2MapReduce falls back to iterMR recomp. We 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 (i.e., matrix) and the state data (i.e., vector). 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.

CONCLUSION

We have described i2MapReduce, 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 runtime for refreshing big data mining results compared to recomputation on both plain and iterative MapReduce.

REFERENCES:

[1]Y.Zhang, S.chen,Q .Wang and Ge Yu.I^2 Map Reduce:Incremental Map Reduce for mining Evolving Big Data.arXiv:1501.0485[cs.DC] 1,2015

[2]P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar , and R. Pasquin. Incoop: Mapreduce for incremental computations. In Proc. of SOCC 11, 2011.

[3]J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H.Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative map Reduce. In Proc. of MAPREDUCE ’10,2010.

[4]Y. Zhang, Q. Gao, L. Gao, and C. Wang. Accelerate large-scale iterative computation through asynchronous accumulative updates. In Proc. Of ScienceCloud ’12, 2012.

[5]C. Yan, X. Yang, Z. Yu, M. Li, and X. Li. Incmr: Incremental data processing based on mapreduce. In Proc. of CLOUD ’12, 2012.

[6]Y. Bu, B. Howe, M. Balazinska, and M. Ernst, “Haloop: efficient iterative data processing on large clusters,” in *34th International Conference on Very Large Data Bases (VLDB)*, 2010.