ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 6, Issue 12, December 2017

A Review on i^{2}Mapreduce: Incremental Mapreduce for Evolving Big Data

Sayali Lonkar

M. E Student, Department of Computer Engineering, JSPM's Imperial College of Engg. & Research, Pune, India

ABSTRACT: Many online data sets grow incrementally over time as new entries are slowly added and existing entries are deleted or modified. As new data and updates are constantly arriving, the results of data mining applications become outdated over time. To keep away from the scenario it is necessary to refreshing mining results so that it can avoid the cost of re-computation from scratch. MapReduce for efficient iterative computations, it is too expensive to perform an entirely new large-scale MapReduce iterative job to timely accommodate new changes to the underlying data sets. The i2MapReduce is an extension to MapReduce that supports key-value pair level incremental processing and also more sophisticated iterative computation, which is widely used in data mining applications .Incremental interactive processing is a small number of updates which propagate to affect a large portion of intermediate states after a number of iterations. The technique helps in improving the job running time and reduces the running time of refreshing the results of big data.

KEYWORDS*:* Big Data, Mapreduce, Incremental Processing, Iterative Computation

I. INTRODUCTION

Modern Internet applications have created a need to manage immense amounts of data quickly. For example devices and communication means like social networking sites, the amount of data produced by mankind is growing rapidly every year. It has become increasingly popular to mine such big data, which helps in taking business decisions or to provide better personalized good quality services. A large number of frameworks have been developed for big data analysis. MapReduce is one of the simple, generalized, framework used in production. Implementations of map-reduce enable many of the most common calculations on large-scale data to be performed on large collections of computers, efficiently and in a way that is tolerant of hardware failures during the computation. Here the main focus is on improving MapReduce technique.

The data lying in the servers of the company was just data until yesterday – sorted and filed. Suddenly, the slang Big Data got popular and now the data in the company is Big Data. The term covers each and every piece of the data that organization has stored till now. Big data involves the data produced by different devices and applications. Big data technologies are important in providing more accurate analysis, which may lead to more concrete decision-making resulting in greater operational efficiencies, cost reductions, and reduced risks for the business. 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 an advanced approach to refreshing mining results. Given the size of the input big data, it is very heavy weighted to return the entire computation from scratch. Incrementally processing the new data of a large data set, takes state as implicit input and combines it with new data. MapReduce [1] programming model is widely used for large scale and one-time data-intensive distributed computing, but it lacks for built-in support for the iterative process.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 6, Issue 12, December 2017

II. MAPREDUCE BACKGROUND

MapReduce is a one of a promising technique of computing that manage large-scale computations in a way that is tolerant of hardware faults .A MapReduce job usually partition the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. MapReduce includes two main functions, called Map and Reduce. MapReduce computation is shown in Figure 1.In the Figure 1 the system manages the parallel execution, coordination of a task that execute Map or Reduce, and also deals with the possibility that one of these tasks will fail to execute. These Map tasks turn the chunk into a sequence of key-value pairs<K, V>. The way key-value pairs are produced from the input data is determined by the code written by the user for the Map function. The key-value pairs from each Map task are composed by a master controller and sorted by key. The keys are divided among all the Reduce tasks, so all key-value pairs with the same key wind up at the same Reduce task. The Reduce tasks work on one key at a time and combine all the values associated with that key in some way. The manner of combination of values is determined by the code written by the user for the Reduce function.

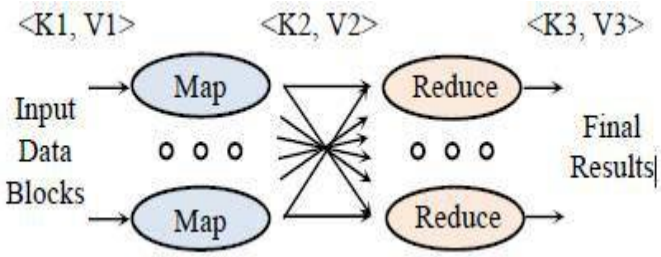

Figure 1.MapReduce computation

III. ITERATIVE ALGORITHMS

Many data mining algorithms have an iterative process to perform data recursively. In this section, it first provides two examples of iterative algorithms, and limitations of implementing these algorithms in MapReduce

i) Single Source Shortest Path (SSSP)

Single Source Shortest Path (SSSP) [2] is a classical problem that derives the shortest distance from a source node S to all other nodes in a graph. Given a weighted, directed graph G = (V, E), is a linear ordering of all its nodes such that if G has an edge (u, v) then u appears before v in the ordering.

To perform the shortest path computation in the MapReduce framework, it can follow breadth -first manner. Starting from a source S. The distance to the source being initialized as 0, i.e., d(s) = 0, and any other node's minimum distance being initially set as , the map function is applied on each node u. The first part is the minimum distance from S to u, i.e., d (u) and the second part is the set of node u's outgoing links' weight values, i.e., W (u, ∗). Using these input

ii) PageRank

The PageRank [3] vector R is defined over a directed graph G = (V, E). Each node v in the graph is associated with a

PageRank score R(v). The initial rank of each node is 1/|V| Each node v updates its rank iteratively as follows:

$$R^{(k+1)}(v) = \frac{1-d}{|V|} + \sum_{u \in N^-(v)} \frac{d \cdot R^{(k)}(u)}{|N^+(u)|},$$

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 6, Issue 12, December 2017

Where N− (v) is the set of nodes pointing to node v, N+ (v) is the set of nodes that v points to, k is the iteration number, and d is a constant representing the damping factor. Here the iterative process continues for a fixed number of iterations or till the difference between the resulting PageRank scores of two consecutive iterations is smaller than a threshold.

a) Map For each node u, output key-value pairs <v, d (R(u) /|N+(u)| >, where v □ N+(u), and output the retained ranking score and its outbound neighbors set, <u, [1−d/|V| , N+(u)].

b) Reduce For each node v, sum 1−d /|V| and d (R(u) /|N+(u)|) received from any u to update R(v), and output key-value pair <v, [R(v), N+(v)]>.

Limitations of MapReduce Implementation

The map task in the iteration starts, after finishing all reduce task in the previous iteration. Map task should be started as soon as their input data are available. The main loop in the MapReduce implementation requires the completion of previous iteration job before starting the next iteration job .The limitation results in the avoidable synchronization overhead. MapReduce implementation starts a new job for each iteration, which involves repeated task initializations and cleanups. These result in the unnecessary scheduling overhead.

IV. RELATED WORK

Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, has conducted experiment by using Distributed graph lab[4].It is a framework for machine learning and data mining in the cloud. Graph Lab abstraction which naturally expresses the asynchronous, dynamic, graph-parallel computation while ensuring data consistency and achieving a high degree of parallel performance in the shared-memory setting. With the exponential growth in the scale of Machine Learning and Data Mining (MLDM) problems and increasing sophistication of MLDM techniques, there is an increasing need for systems that can execute MLDM algorithms efficiently in parallel on large clusters. Simultaneously, the availability of Cloud computing services like Amazon EC2 provide the promise of on-demand access to affordable large-scale computing and storage resources without substantial upfront investments. Unfortunately, designing, implementing, and debugging the distributed MLDM algorithms needed to fully utilize the Cloud can be prohibitively challenging requiring MLDM experts. To address race conditions, deadlocks, distributed state, and communication protocols while simultaneously developing mathematically complex models and algorithms. MLDM have focused on modeling the dependencies between data. By modeling data dependencies, then it is able to extract more signals from noisy data. Unfortunately, data parallel abstractions like MapReduce are not generally well suited for the dependent computation typically required by more advanced MLDM algorithms. Dynamic asynchronous graph parallel computation will be a key component in large-scale machine learning and data-mining systems, and thus further research into the theory and application of these techniques will help in defining the emerging field of BigData learning.

S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl conducted an experiment on parallel data flows [5]. Parallel dataflow systems are an increasingly popular solution for analyzing large data volumes. They offer a simple programming abstraction based on directed acyclic graphs, and relieve the programmer from dealing with the complicated tasks of scheduling computation, transferring intermediate results, and dealing with failures. Most importantly, they allow dataflow programs to be distributed across large numbers of machines which is imperative when dealing with today's data volumes. While dataflow systems were originally built for tasks like indexing, filtering, transforming, or aggregating data, their simple interface and powerful abstraction have made them popular for other kinds of applications, like machine learning or graph analysis. Many of these algorithms are of iterative or recursive nature, repeating some computation until a condition is fulfilled. Naturally, these tasks pose a challenge to dataflow systems, as the flow of data is no longer acyclic. It focuses on two different kind of iteration. Bulk iteration which computes a completely new partial solution from the previous iteration's result, optionally using additional data sets that remain constant in the course of the iteration. Prominent examples are machine learning algorithms like Batch

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 6, Issue 12, December 2017

Gradient Descend and Distributed Stochastic Gradient Descent, many clustering algorithms (such as K-Means), and the well-known PageRank algorithm and another one is Incremental Iteration. The major drawbacks of the system are dataflow systems are practically inefficient for many iterative algorithms. The systems are, however, still required for other typical analysis and transformation tasks.

Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, has conducted experiments using Haloop technique [6] for efficient data processing on large clusters. The need for highly scalable parallel data processing platforms is rising due to an explosion in the number of massive-scale data intensive applications both in industry (e.g., web-data analysis, click-stream analysis, network-monitoring log analysis) and in the sciences (e.g., analysis of data produced by massive-scale simulations, sensor deployments, high-throughput lab equipment). MapReduce is a well-known framework for programming commodity computer clusters to perform large-scale data processing in a single pass. A MapReduce cluster can scale to thousands of nodes in a fault-tolerant manner. Although parallel database systems may also serve these data analysis applications, they can be expensive, difficult to administer, and lack fault-tolerance for long-running queries. Hadoop, an open-source MapReduce implementation, has been adopted by Yahoo!, Facebook, and other companies for large-scale data analysis. The main drawback of the experiment is MapReduce framework does not directly support these iterative data analysis applications. Instead, programmers must implement iterative programs by manually issuing multiple MapReduce jobs and orchestrating their execution using a driver program .Even though much of the data may be unchanged from iteration to iteration, the data must be re-loaded and re-processed at each iteration, wasting I/O, network bandwidth, and CPU resources.

D. Logothetis, C. Olston, B. Reed, K. C. Webb, and K. Yocum, has conducted experiment on statefull bulk processing

[7] for incremental analytics. There is a growing demand for large-scale processing of unstructured data, such as text, audio, and image files. It is estimated that unstructured data is now accumulating in data centers at three times the rate of traditional transaction-based data.These environments often require data processing systems to absorb terabytes of new information every day while running a variety of complex data analytics. The data “deluge” presents major data management challenges, and non-relational information analysis is quickly emerging as a bedrock technology for these large-scale data processing efforts. The work carried out addresses the need for stateful dataflow programs that can rapidly sift through huge, evolving data sets. These data-intensive applications perform complex multi-step computations over successive generations of data inflows, such as weekly web crawls, daily image/video uploads, log files, and growing social networks. While programmers may simply re-run the entire dataflow when new data arrives, which is grossly inefficient, so increasing the result latency and squandering hardware resources and energy. Alternatively, programmers may use prior results to incrementally incorporate the changes. However, current large-scale data processing tools, such as Map-Reduce or Dryad, limits how programmers incorporate and use state in data-parallel programs. Straightforward approaches to incorporating state can result in custom, fragile code and disappointing performance.

Y. Zhang, Q. Gao, L. Gao, and C. Wang, conducted an experiment to Accelerate large-scale iterative computation [8] through asynchronous accumulative updates. The advances in sensing, storage, and networking technology have created huge collections of high-volume, high dimensional data. Making sense of these data is critical for companies and organizations to make better business decisions, and even bring convenience to the daily life. Recent advances in scientific computing, such as computational biology, have led to a flurry of data analytic techniques that typically require an iterative refinement process. Common to these proposed distributed frameworks, iterative updates are performed iteration by iteration. Specifically, an iterative update in iteration k is performed after all updates in iteration $k-1$ have completed. Within the same iteration, the results from an earlier update cannot be used for a later update. While it is the traditional model for iterative computations, such a model does slow down the progress of the computation. First, the concurrent update model does not utilize the already retrieved partial results from the same iteration, which prolongs the iteration process and lead to slow convergence. Second, a synchronization step is required for every iteration. And it is a time consuming in a heterogeneous distributed environment.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Visit: www.ijirset.com

Vol. 6, Issue 12, December 2017

V. CONCLUSION

The conclusion of the survey carried out is as follows, i2MapReduce that explicitly supports the iterative processing of large relational data, and addresses all the issues in the MapReduce implementation of iterative processing. It provides a framework for programmers to explicitly model iterative algorithms and proposes the concept of persistent tasks to perform the iterative computation to avoid repeatedly creating, destroying, and scheduling tasks. It facilitates asynchronous execution of tasks within the same iteration, to accelerate the processing speed.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in Proc. of OSDI '04, 2004.
[2] Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, 2nd edn. McGraw-Hill Higher Education (2001)
[3] Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Comput Networks ISDN 30, 107–117 (1998)
[4] 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," PVLDB, vol. 5, no. 8, pp. 716–727, 2012.
[5] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl, "Spinning fast iterative data flows," PVLDB, vol. 5, no. 11, pp. 1268–1279, 2012.
[6] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: efficient iterative data processing on large clusters," PVLDB, vol. 3, no. 1- 2, pp. 285–296, 2010.
[7] D. Logothetis, C. Olston, B. Reed, K. C. Webb, and K. Yocum, "Stateful bulk processing for incremental analytics," in Proc. of SOCC '10, 2010.