ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

Incremental Map Reduce For Efficient Data Retrieval

P. Chitra[1], L.Sam Joseph[2], N.Vimal[2], B.yuvaraj[2]

Professor, Department of Information Technology, Prathyusha Engineering College, India[1]

Student, Department of Information Technology, Prathyusha Engineering College, India[2]

ABSTRACT: Programming paradigm allows for massive scalability across hundreds or thousands of servers which leads to the collection of enormous volume of data. Processing this voluminous data requires an advanced techniques like Map Reduce for efficient processing of data. MapReduce refers to two separate and distinct tasks. The first is the map job, which takes a set of data and converts it into another set of data. The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As new data and updates are constantly arriving, the results of data mining applications become stale and obsolete over time. This paper proposes incremental-MapReduce a novel incremental processing extension to MapReduce, the most widely used framework for mining big data. It utilizes previously saved states to avoid the expense and time required for reprocessing the same set of data.We analysis Financial accounting data is used for analyzing the proposed approach and the experimental outcome shows the effectiveness of the proposed technique when compared to the existing system with simple MapReduce method.

KEYWORDS: Incremental processing, MapReduce, iterative computation

I.INTRODUCTION

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. For example, the PageRank algorithm computes ranking scores of web pages based on the web graph structure for supporting web search. However, the web graph structure is constantly evolving; Web pages and hyper-link are created, deleted, and updated. As the underlying web graph evolves, the PageRank ranking results gradually become stale, potentially lowering the quality of web search. Therefore, it is desirable to refresh the PageRank computation regularly.

Incremental processing is a promising approach to refreshing mining results. Given the size of the input big data, it is often very expensive to return the entire computation from scratch. Incremental processing exploits the fact that the input data of two subsequent computations are similar. Only very small fraction of input data has changed. The idea is to save the states in previous computation reuse the states in next computation and perform re-computation for states that are affected by changed input data.

On the other hand, Incoop 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 (kv-pairs). If Incoop detects any data changes in the input of a task, it will return the entire task. While this approach easily leverages existing MapReduce features for state savings, it may incur a large amount of redundant computation if 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 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.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

II.LITERATURE SURVEY

MapReduce: Simplified Data Processing on Large Clusters

J. Dean and S. Ghemawat [1] described that MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Incoop : MapReduce for incremental computations

Pramod Bhatotia, Alexander Wieder, Rodrigo Rodrigues, Umut A. Acar, Rafael Pasquini [11] described the architecture, implementation, and evaluation of a generic MapReduce framework, named Incoop, for incremental computations. Incoop detects changes to the input and enables the automatic up-date of the outputs by employing an efficient, fine-grained result re-use mechanism. To achieve efficiency without sacrificing transparency, we adopt recent advantages in the area of programming languages to identify systematically the short-comings of task-level memorization approaches, and address them using several novel techniques such as a storage system to store the input of consecutive runs, a contraction phase that make efficient, and a scheduling algorithm for Hadoop that is aware of the location of previously computed results.

Pregel: A System for Large-Scale Graph Processing

G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski [4] proposed Pregel,This computations consist of a sequence of iterations, called supersteps. During a superstep the framework invokes a user defined function for each vertex, conceptually in parallel. The function specifies behavior at a single vertex V and a single superstep S. It can read messages sent to V in superstep S 1, send messages to other vertices that will be received at superstep S + 1, and modify the state of V and its outgoing edges. Messages are typically sent along outgoing edges, but a message may be sent to any vertex whose identifier is known. The vertex-centric approach is reminiscent of Map Reduce in that users focus on a local action, processing each item independently, and the system composes these actions to lift computations.

Piccolo: Building Fast, Distributed Programs with Partitioned Tables

R. Power and J. Li [3] proposed Piccolo, which is a new data-centric programming model for writing parallel in-memory applications in data centers. Unlike existing data-flow models, Piccolo allows computation running on different machines to share distributed, mutable state via a key-value table interface. Piccolo enables efficient application implementations. In particular, applications can specify locality policies to exploit the locality of shared state access and Piccolo's run-time automatically resolves write-write conflicts using user-defined accumulation functions. Using Piccolo, we have implemented applications for several problem domains, including the PageRank algorithm, *k*-means clustering and a distributed crawler. Experiments using 100 Amazon EC2 instances and a 12 machine cluster show Piccolo to be faster than existing data flow models for many problems, while providing similar fault-tolerance guarantees and a convenient programming interface.

Twister: A Runtime for Iterative MapReduce

J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox [9] Proposed Twister, which is an enhanced MapReduce runtime with an extended programming model that supports iterative MapReduce computations

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

efficiently. It uses a publish/subscribe messaging infrastructure for communication and data transfers and supports long running map/reduce tasks which can be used in "configure once and use many times" approach. In addition it provides programming extensions to MapReduce with "broadcast" and "scatter" type data transfers. These improvements allow Twister to support iterative MapReduce computations highly efficiently compared to other MapReduce runtimes.

HaLoop: Efficient Iterative Data Processing on Large Clusters

Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst [8] designed HaLoop, that is used to efficiently handle the above types of applications. HaLoop extends MapReduce and is based on two simple intuitions. First, a MapReduce cluster can cache the invariant data in the first iteration,and then reuse them in later iterations. Second, a MapReduce cluster can cache reducer outputs, which makes checking for a fixpoint more efficient, without an extra MapReduce job.

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica [2] proposed an abstraction called resilient distributed datasets (RDDs) that enables efficient data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators. RDDs provide an interface based on coarse-grained transformations (e.g., map, filter and join) that apply the same operation to many data items. This allows them to efficiently provide fault tolerance by logging the transformations used to build a dataset (its lineage) rather than the actual data.1 If a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to recomputed just that partition. Thus, lost data can be recovered, often quite quickly, without requiring costly replication.

REX: Recursive, DeltaBased DataCentric Computation

S. R. Mihaylov, Z. G. Ives, and S. Guha[5] Developed the REX system, which includes: (1) support for high-level programming using declarative SQL, (2) the ability to do pipelined, ad hoc queries as in DBMSs, (3) the failover capabilities and easy integration of user-defined code from cloud platforms, and (4) efficient support for incremental iterative computation with arbitrary termination conditions and explicit creation of custom delta operations and handlers. REX runs efficiently on clusters. Its generalized treatment of streams of incremental updates is, to our knowledge, unique, and as we show experimentally, extremely beneficial. REX supports extended SQL with user-defined Java code, and performs cost-based optimization. We take the further step of allowing the programmer to directly use compiled code for Hadoop in our environment: while adding some performance overhead versus a native approach, it provides an easy transition.

III.ARCHITECTURE DIAGRAM

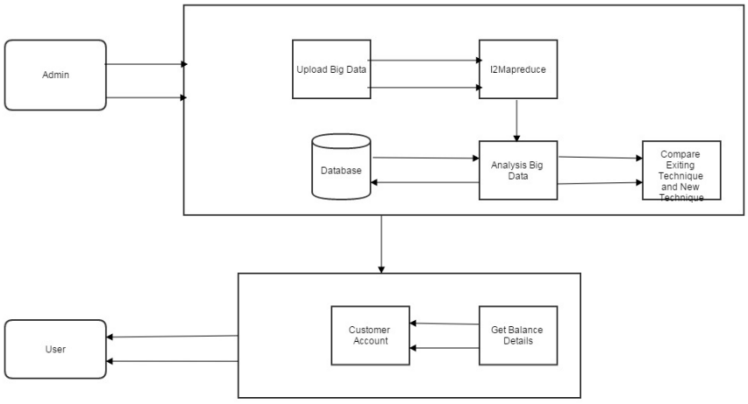
The architecture diagram shows that there are two logins in our project, a user and an admin. The user is a normal bank user who does his transactions through online. The users should first register with a username and password. Then the users can create a bank account based on their needs such as a saving account or a current account. The users can also perform various transactions in their account such as deposit and withdrawal. Next is the admin login which plays a major role. The admin may be a bank employee who monitors the various transactions taking place in a bank .The admin takes care of uploading the data to the cloud, the data uploaded is encrypted using AES algorithm and a MD5 hashing algorithm is used to check the integrity of the data. In our project we compare the existing map reduce and the proposed incremental map reduce concepts. We analyze the time taken to retrieve the same data using both the concepts. The result shows that incremental map reduce takes a lesser time than the existing map reduce technique.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016



Architecture diagram

IV.AUTHENTICATION AND AUTHORIZATION

In this module the user have to register first, then only he/she has to access the data base in online bank application. After registration the User can login to the site. The authorization and authentication process facilitates the system to protect itself and besides it protects the whole mechanism from unauthorized usage. The Registration involves in getting the details of the users who wants to use this application. Admin can login to the site to manage over all process.

V.MANAGING BANK ACCOUNTS

This provides two modules for managing a bank account. One is intended to be used by the bank, and the other by the customer. The approach is to implement a general-purpose parameterized function providing all the needed operations, then apply it twice to the correct parameters, constraining it by the signature corresponding to its final user: the bank or the customer. This set of functions provide the minimal operations on an account. The creation operation takes as arguments the initial balance and the maximal overdraft allowed. Excessive withdrawals may raise the Bad Operation exception. We keep unspecified for now the types of the log keys (type *tkey*) and of the associated data (type *tinfo*), as well as the data structure for storing logs (type *t*). We assume that new information added with the add function are kept in sequence.

VI.UPLOAD BIG DATA

Analyzing transactional data is at the core of the data at a financial institution's disposal. Transaction data can uncover powerful insights into customer needs, preferences and behaviors. However, transaction data represents only one type of insight that financial institutions possess. Other types of insight that reside within an organization include both structured data (demographic profiles, product ownership, balances, etc.) and unstructured internal data (call center logs, channel interactions, correspondence, etc.).In addition to internal data sources, banks and credit unions can also take advantage of external data. Social media represents a largely untapped source of insight that financial organizations can use to develop a more holistic view of their customers. financial organizations and their executives to improve their customer experience levels to differentiate themselves and to stay ahead of competitors. This, in turn, will improve acquisition results, engagement and cross-sell effectiveness as well as customer loyalty and growth.

VII.I2MAP REDUCE AND ANALYZING ITERATIVE COMPUTATIONS

Cloud intelligence applications often perform iterative computations (e.g., Online Bank) on constantly changing data sets (e.g., Web graph). While previous studies extend 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. In this paper, we propose i2MapReduce to support incremental iterative computation. We observe

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

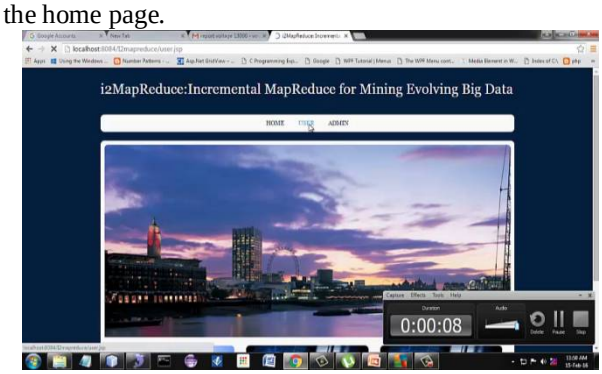
(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

that in many cases, the changes impact only a very small fraction of the data sets, and the newly iteratively converged state is quite close to the previously converged state. i2MapReduce exploits this observation to save re-computation by starting from the previously converged state, and by performing incremental updates on the changing data. Our preliminary result is quite promising.

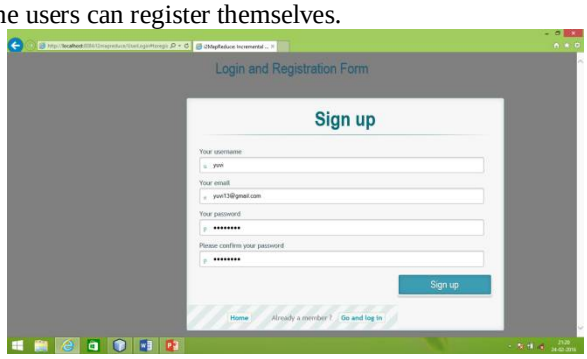
VIII.SCREEN SHOTS

The following are the screenshots of the implemented project which shows the various modules, users and how they are implemented. The following is the home page.



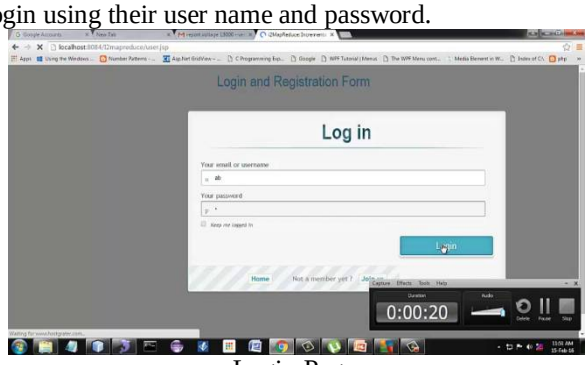
Home Page

This is the register page which the users can register themselves.



Register Page

After registering the users can login using their user name and password.



Login Page

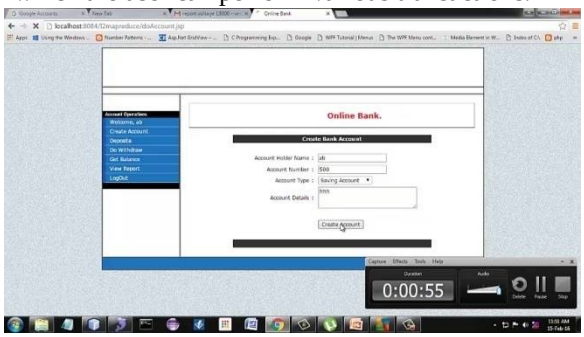
ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

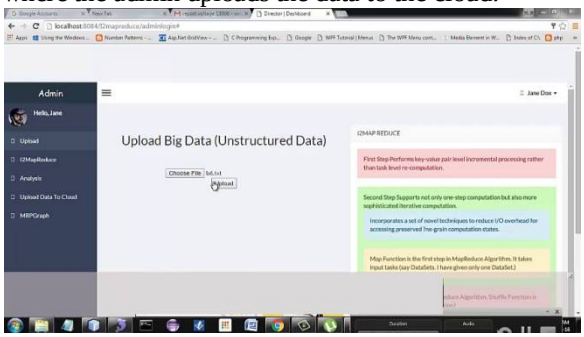
Vol. 4, Issue 3, March 2016

The following is the User page in which the user can perform various transactions.



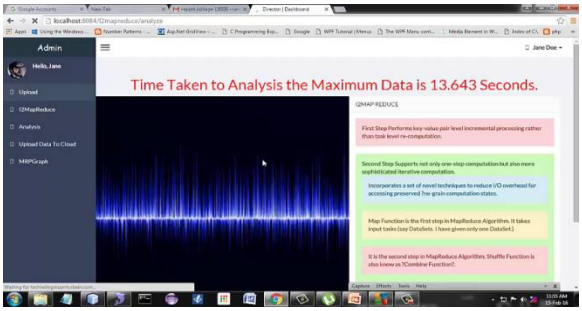
User page

The following is the admin page where the admin uploads the data to the cloud.



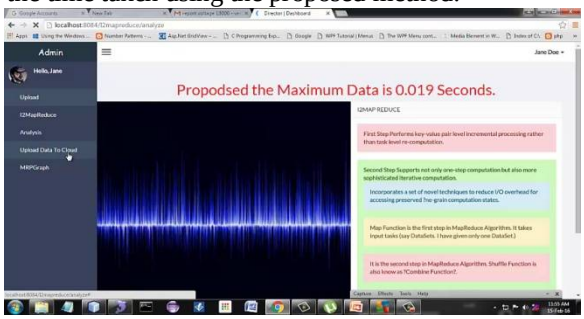
Admin uploading file

The following screen shot shows the time taken using normal existing method.



Time taken using map reduce

The following screen shot shows the time taken using the proposed method.



Time taken using incremental map reduce

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

IX.CONCLUSION

We 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 re-computation on both plain and iterative MapReduce.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on 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 distributed datasets: A fault-tolerant abstraction for, in-memory cluster computing," in Proc. 9th USENIX Conf. Netw. Syst. Des. Implemen-tation, 2012, p. 2.
[3] R. Power and J. Li, "Piccolo: Building fast, distributed programs with 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-scale graph processing," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2010, pp. 135–146.
[5] 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.
[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] 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.
[8] 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.
[9] 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.
[10] 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.
[11] Pramod Bhatotia, Alexander Wieder, Rodrigo Rodrigues, Umut A. Acar, Rafael Pasquini "Incoop: MapReduce for Incremental" Technical Report: MPI-SWS-2011-003.