ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 9, March 2016

ENERGY AWARE SCHEDULING OF MAP REDUCE JOBS FOR BIG DATA APPLICATIONS

G.K.SRIVATSAN[1],A.ASADULLAH[2],T.GANESH[3], K.KATHIRAVAN[4]
Department Of Information Technology, Jeppiaar Institute Of Technology,India.
[1]srivatsan.g.k@gmail.com,
[2]azar.370@gmail.com, [3]geganesh95@gmail.com,
[4]kathirkerb2410@gmail.com

ABSTRACT

This proposed method is used to optimize the mining results and evaluate Map Reduce using a one-step algorithm and three iterative algorithms with diverse computation characteristics for efficient mining. Naive-Bayes Algorithm is used to find the search from the root cause and it does a detailed study and produces an accurate result. Incremental processing is a promising approach for refreshing mining results. It utilizes previously saved states to avoid the expense of re-computation from scratch. In this paper, we propose Energy Map Reduce Scheduling Algorithm, a novel incremental processing extension to Map Reduce, the most widely used framework for mining big data. Map reduce is a programming model for processing and generating large amount of data in parallel time. In this paper, EMRSA is algorithm provide more energy and less maps. Priority based scheduling is a task will allocate the schedules based on necessary and utilization of the Jobs. For reducing the maps, it will reduce the system work so easily energy has improve. Final results show the experimental comparison of the different algorithms involved in the paper.

KEYWORDS: *Map Reduce, Support Vector Machine (SVM) Algorithm, Naive-Bayes Algorithm.*

1. INTRODUCTION

Today huge amount of digital data is being accumulated in many important areas, including e-commerce, social network, finance, health care, education, and environment. Big data is constantly evolving. As new data and results will become stale and obsolete over time updates are being collected, the input data of a big data mining algorithm will gradually change, and the computed. HIVE is a data warehousing infrastructure based on Hadoop. Hadoop provides massive scale out and fault tolerance capabilities for data storage and processing (using the map-reduce programming paradigm) on commodity hardware.HIVE does not support hadoop as a batch processing system and Hadoop jobs tend to have high latency and incur substantial overheads in job submission and scheduling. As a result - latency for Hive queries is generally very high (minutes) even when data sets involved are very small (say a few hundred megabytes). As a result it cannot be compared with systems such as Oracle where analyses are conducted on a significantly smaller amount of data but the analyses proceed much more iteratively with the response times between iterations being less than a few minutes. Hive aims to provide acceptable (but not optimal) latency for interactive data browsing, queries over small data sets or test queries. Hive is not designed for online transaction processing and does not offer real-time queries and row level updates. It is best used for batch jobs over large sets of immutable data (like web logs). In the following sections we provide a tutorial on the capabilities of the system. We start by describing the concepts of data types, tables and partitions (which are very similar to what you would find in a traditional relational DBMS) and then illustrate the capabilities of the QL language. We focus on improving Map Reduce in this paper. The problem

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 9, March 2016

with Green Computing is the major challenges recent days, due to the size of big data centers it will take more computation for Data Accessing Process. It will increase the time, also it will consume more energy. In this project we are going to solve above mentioned issues.

[1]Jeffrey Dean Et al discussed about 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. Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

[2] MateiZahariaMosharaf Chowdhury Et al focussed about the present Resilient Distributed Datasets (RDDs), a distributed memory abstraction that allows programmers to perform in-memory computations on large clusters while retaining the fault tolerance of data flow models like MapReduce. RDDs are motivated by two types of applications that current data flow systems handle inefficiently: iterative algorithms, which are common in graph applications and machine learning, and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a highly restricted form of shared memory: they are read-only datasets that can only be constructed through bulk operations on other RDDs. However, we show that RDDs are expressive enough to capture a wide class of computations, including MapReduce and specialized programming models for iterative jobs such as Pregel. Our implementation of RDDs can outperform Hadoop by 20× for iterative jobs and can be used interactively to search a 1 TB dataset with latencies of 5–7 seconds.

[3] Russell Power Et al Piccolo 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.

[4]GrzegorzMalewicz Et al focused on Many practical computing problems concern large graphs. Standard examples include the Web graph and various social networks. The scale of these graphs—in some cases billions of vertices, trillions of edges—poses challenges to their efficient processing. In this paper we present a computational model suitable for this task. Programs are expressed as a sequence of iterations, in each of which a vertex can receive messages sent in the previous iteration, send messages to other vertices, and modify its own state and that of its outgoing edges or mutate graph topology. This vertex centric approach is flexible enough to express a broad set of algorithms. The model has been designed for efficient, scalable and fault-tolerant implementation on clusters of

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 9, March 2016

thousands of commodity computers, and its implied synchronicity makes reasoning about programs easier. Distribution related details are hidden behind an abstract API. The result of a framework for processing large graphs that is expressive and easy to program.

[5]Svilen R Et al discussed about In today's Web and social network environments, query workloads include ad hoc and OLAP queries, as well as iterative algorithms that analyze data relationships (e.g., link analysis, clustering, learning). Modern DBMSs support ad hoc and OLAP queries, but most are not robust enough to scale to large clusters. Conversely, "cloud" platforms like MapReduce execute chains of batch tasks across clusters in a fault tolerant way, but have too much overhead to support ad hoc queries. Moreover, both classes of platform incur significant overhead in executing iterative data analysis algorithms. Most such iterative algorithms repeatedly refine portions of their answers, until some convergence criterion is reached. However, general cloud platforms typically must reprocess all data in each step. DBMSs that support recursive SQL are more efficient in that they propagate only the changes in each step — but they still accumulate each iteration's state, even if it is no longer useful. User-defined functions are also typically harder to write for DBMSs than for cloud platforms. We seek to unify the strengths of both styles of platforms, with a focus on supporting iterative computations in which changes, in the form of deltas, are propagated from iteration to iteration, and state is efficiently updated in an extensible way. We present a programming model oriented around deltas, describe how we execute and optimize such programs in our REX runtime system, and validate that our platform also handles failures gracefully. We experimentally validate our techniques, and show speedups over the competing methods ranging from 2.5 to nearly 100 time.

2. PROPOSED APPROACH

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. For scheduling the task, here we will apply priority based task scheduling. It will improve the Scheduling Jobs. Lets take key/value pairs and added in a list, finally the reduce takes the sums into one and produce single output. Energy aware scheduling will decrease the energy consumption ratio.

2.1 .*Collection of Data*

In this stage, data set consists of large number of files 1000 data from geo distributed data out of which 100 are from particular location. The fig 2.1, which certifies the files to be free from spyware. This hosts information about different types of persons and their location information

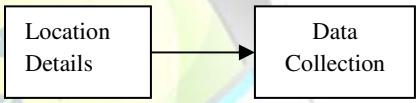

Figure 2.1. representation of data collection

2.2Dataset Creation

In which byte sequences represent fragments of machine code from an executable file. We use xxd, which is a UNIX-based utility for generating hexadecimal dumps of the binary files. From these hexadecimal dumps we may then extract byte sequences, in terms of *n*-grams of different sizes. ARFF databases based on frequency and common features were generated. All input attributes in the data set are represented by Booleans. These ranges are represented by either 1 or 0.

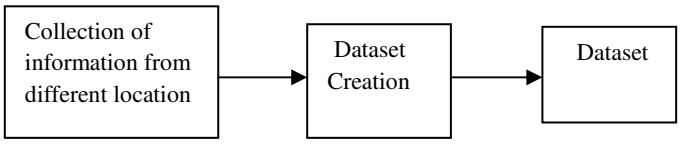

Figure 2.2.Representation of dataset creation

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 9, March 2016

2.3 Feature Extraction

In this stage output from the parsing is further subjected to feature extraction. We extract the features by using following approaches, the Common Feature-based Extraction (CFBE) and Frequency-based Feature Extraction. The occurrence of a feature and the frequency of a feature. Both methods are used to obtain Reduced Feature Sets (RFSs) which are then used to generate the ARFF files.

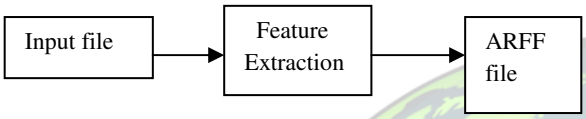

Figure 2.3.Representation of Feature Extraction

2.4. Classification

Here we use SVM classifier, support vector machine (SVM) is a concept in statistics and computer science for a set of related supervised learning methods that analyze data and recognize patterns, used for classification and regression analysis. The standard SVM takes a set of input data and predicts, for each given input, which of two possible classes comprises the input, making the SVM a non-probabilisticbinarylinear classifier. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other.

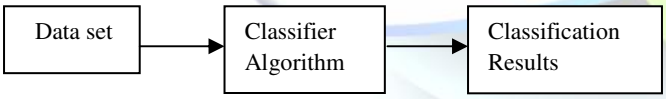

Figure 2,4 Representation of Data Classification

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

3.REFERENCES

1. Y. Bu, V. Borkar, J. Jia, M. J. Carey, and T. Condie, "Pregelix: Bigger graph analytics on a dataflow engine," in Proc. VLDB Endowmen,2015, 8- 2,161–172.

2. Y. Zhang, S. Chen, Q. Wang, and G. Yu, "i2mapreduce: Incremental mapreduce for mining evolving big data," 2015, 150-854.

3. D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and
M. Abadi, "Naiad: A timely dataflow system," 24th ACM Symp. Operating. System Principles, 2013, 439–455.

4. Junzhou Luo ,Fang Dong, Runqun Xiong**, "**SLDP: A Novel Data Placement Strategy for Large-Scale Heterogeneous Hadoop Cluster ",Advanced Cloud and Big Data (CBD), 2014 Second International Conference on 2015,57-10.

5. C. Yan, X. Yang, Z. Yu, M. Li, and X. Li, "IncMR: Incremental data processing based on mapreduce,", IEEE 5th Int. Conf. Cloud Computing, 2012, 534–541.