International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 6, Issue 4, April 2017, ISSN: 2278 – 1323

An Illustrative Approach to Create and Compare a Hybrid Big Data Integration Engine with MySQL

Aqshata Gade
Student
A.C. Patil College of Engineering
Kharghar, Navi Mumbai, India

Aakansha Gupta
Student
A.C. Patil College of Engineering
Kharghar, NaviMumbai, India

Purva Chaudhary
Student
A.C. Patil College of Engineering
Kharghar, NaviMumbai, India

Amruta Anbhule
Student
A.C. Patil College of Engineering
Kharghar, Navi Mumbai, India

Prof. Shaila Deore
Professor
A.C. Patil College of Engineering
Kharghar, Navi Mumbai, India

Abstract: **Nowadays large enterprises maintain a huge amount of data in multiple backend systems including traditional database systems and recently popular big data systems. How to integrate such data and provide a consolidate query and analytic becomes a challenging task. Neither traditional database warehouse nor recent Big Data system (e.g. Apache Spark) can fully leverage the power of each backend system. In this paper, we build a hybrid data processing engine, to fully integrate backend systems. Given the backend systems, data is distributed at multiple locations. The proposed system focuses on the optimization of the amount of data movement. To this end, it proposes a technique of query pushdown for such optimization. A proof-of-concept prototype of the proposed system successfully verifies that it can achieve much faster running time than MySQL.**

Index Terms—query pushdown, data movement optimization.

I. INTRODUCTION

Nowadays large enterprises maintain a huge amount of data in multiple backend systems including traditional database systems and recently popular big data systems. How to integrate, query and analyze such data becomes a challenging task.
We take a typical telecom provider as an example. First, the telecom provider frequently maintains the Billing information by Oracle or recent Vertical database. Beyond this, due to the increasing prevalence of mobile cloud services, mobile devices generate a huge amount of spatiotemporal log data, such as the signaling log that is used to record user ID (i.e., International Mobile Subscriber Identity), location (longitude, latitude), time-stamp, mobile device type, mobile App type, and etc. Given such log data, HDFS with Hive is perhaps the most common example to maintain such data for storage,

query and analytics.
Most data warehouse applications are implemented using relational databases that use SQL as the query language. Hive lowers the barrier for moving these applications to Hadoop. People who know SQL can learn Hive easily. Without Hive, these users must learn new languages and tools to become productive again. Similarly, Hive makes it easier for developers to port SQL-based applications to Hadoop, compared to other tool options. Without Hive, developers would face a daunting challenge when porting their SQL applications to Hadoop. Still, there are aspects of Hive that are different from other SQL-based environments. Documentation for Hive users and Hadoop developers has been sparse. It is nontrivial to perform efficient query and analytic tasks which need to access both critical business data and signaling log data together for insightful analytics. First, given the billing business data in database, we might follow a data warehouse approach (e.g., using Oracle and DB2 data warehouse solution) for query and analytics. Then Extract Transformation and Loading (ETL) tools moves voluminous log data (together with billing data) into the warehouse system. The warehouse system next processes the analytic tasks (e.g., SQL queries) by using parallel database engines. Nevertheless, the log data volume is significantly larger than billing data, and is not so valuable when compared with the critical business data (e.g., billing information). Consequently, moving such large volume of data to the computation in a centralized data warehouse is wasteful.

Second, with HDFS to maintain log data, we could use popular big data systems (e.g., Apache Hadoop and Spark) as an alternative query processing engine. In particular, Spark leverages a unified data abstraction, Resilient Distributed Dataset (RDD), to comfortably support SQL, Machine Learning, etc. which are easily intermixed within Spark programs. When analytic tasks need to access both data, we can still move billing data (together with log data) onto the HDFS. The Spark engine can be used for analytic tasks. However, the powerful database query processing engine is long treated as the “work horse” storage and computation engine of choice for data warehousing applications, and is not fully utilized.

Based on our experiments with real data set from a network traffic anomaly analytic application, we compare the proposed system with a MySQL system. It is a middleware engine to integrate backend systems such as Spark, database (we use MySQL as an example).

The proposed system accepts user programs to query and analyze the data distributed in backend systems. The user program is written by a SQL-like program. The system processes a user program by two main components, *parser* and *optimizer*. The parser generates a logical operator tree based on user programs. The logical operators, such as selection, projection, are to support traditional SQL queries. Based on the generated logical operator tree by the parser, the optimizer then generates optimal query plans (typically a directed acyclic graph DAG as a sub query) for execution. It requires the input information including optimization rules, cardinality estimation, cost estimation, data location, and etc. Based on the input information, the optimizer outputs a set of query plans (sub queries) which are pushed down and processed by backend systems where the data resides.

The *Hadoop* ecosystem emerged as a cost-effective way of working with such large data sets. It imposes a particular programming model, called *Map Reduce[4]* , for breaking up computation tasks into units that can be distributed around a cluster of commodity, server class hardware, thereby providing cost-effective, horizontal scalability. Underneath this computation model is a distributed file system called the *Hadoop Distributed File system* (HDFS). Although the file system is “pluggable,” there are now several commercial and open source alternatives.

However, a challenge remains; how do you move an existing data infrastructure to Hadoop, when that infrastructure is based on traditional relational databases and the *Structured Query Language* (SQL)? What about the large base of SQL users, both expert database designers and administrators, as well as casual users who use SQL to extract information from their data warehouses?

This is where *Hive* comes in. Hive provides an *SQL* dialect, called *Hive Query Language* (abbreviated *Hive QL* or just *HQL*) for querying data stored in a Hadoop cluster. SQL knowledge is widespread for a reason; it’s an effective, reasonably intuitive model for organizing and using data. Mapping these familiar data operations to the low-level Map Reduce Java API can be daunting, even for experienced Java developers. Hive does this dirty work for you, so you can focus on the query itself. Hive translates most queries to Map Reduce jobs, thereby exploiting the scalability of Hadoop, while presenting a familiar SQL abstraction.

II. LITERATURE SURVEY

A. BIG TABLE

Chang et al. gave the concept of Bigtable [1], a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp and each value in the map is an uninterrupted array of bytes.It is a distributed storage system for managing structured data that is designed to scale to a very large size. Many projects of Google store data in Bigtable, including web indexing, Google Earth, and Google Finance.

B. BDM TECHNOLOGY

Ji et al. proposed a general view of Big Data Management (BDM) [2] technologies and applications. They categorized BDM system into three parts namely distributed file system, non-structural and semi-structured data storage and open source cloud platform. Distributed File System include Google File System, Hadoop Distributed File System, Amazon Simple Storage Service, and Elastic Storage System. Non-structural and Semi-structured Data Storage include Big Table, PNUTS, Dynamo, and Llama.

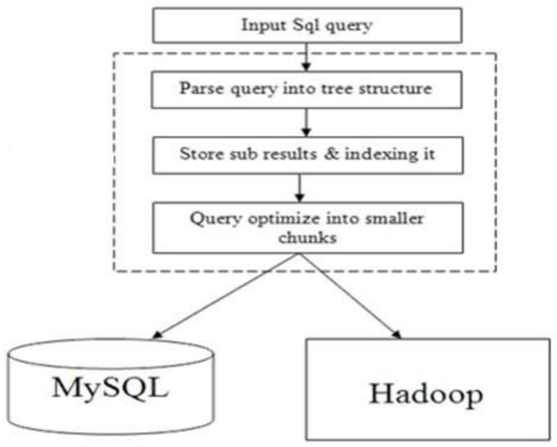
C. THUMP STORAGE

Tao et al. proposed a management and analysis system for structured big data called Thump Storage [3] by integrating the bottom distributed structure of Hadoop distributed file system (HDFS) and the partitioning and scheduling technology of the massive parallel processing database. This system shows high efficiency, low latency and high scalability. It is applicable for managing and analyzing the massive structured data at PB level above.

III. PROPOSED SYSTEM AND WORKING

In this project, we propose a hybrid data processing engine, in order to fully integrate the powers of various back end systems, such as database and Spark, for query processing. When data is on multiple locations (i.e., backend systems), the key of the system is to optimize the amount of data movement across backend systems. To this end, given a query task, Octopus divides the task into multiple sub queries. The sub queries are next pushed down to backend systems and then processed inside the backend systems. Thus the proposed system reduces the amount of data movement. It has two main modules viz. Parser and Optimizers.
Parser processes multi-structured and industry-standard data on Hadoop, including industry standards,documents, and log files, and complex file formats.It can access and parse a varietyof industry standards, documents, log files, and complex file formats in Hadoop.

It supports multiple industry standards.It can extract data from binary documents such as Microsoft Office and Adobe PDF on Hadoop. It also processes hierarchical structures on Hadoop such as XML and JSON formats.Parser can extract data from a variety of log files (web, call-detail records, mainframe, and proprietary) on Hadoop. It also exhibits elastic scaling i.e. process parsing transformations regardless of data format or size that scale with the topology of the Hadoop cluster.


Input Sql query
Parse query into tree structure
Store sub results & indexing it
Query optimize into smaller chunks
MySQL
Hadoop

Most query optimizers represent query plans as a tree of “plan nodes”. A plan node encapsulates a single operation that is required to execute the query. The nodes are arranged as a tree, in which intermediate results flow from the bottom of the tree to the top. Each node has zero or more child nodes those are nodes whose output is fed as input to the parent node. For example, a join node will have two child nodes, which represent the two join operands, whereas a sort node would have a single child node (the input to be sorted). The leaves of the tree are nodes which produce results by scanning the disk, for example by performing an index scan or a sequential scan.

IV. CONCLUSION

Big Data is defined in terms of variety, volume, velocity, variability, complexity and value. Two challenges are associated with data in terms of storage and processing. One is to handle or store large amount of data efficiently and effectively. Second is to filter the most important data from all the data collected by the organization. In this project, we propose build a hybrid data processing engine, to fully integrate backend systems that store the big data. Given the backend systems, data is distributed at multiple locations. The proposed system focuses on the optimization of the amount of data movement. To this end, it proposes a technique of query pushdown for such optimization.

References

[1]Fay Chang, Jeffery Dean, Sanjay Ghemawat, Wilson C. Hsiesh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber. "Bigtable: A Distributed Storage System for Structured Data."

[2]M. Chen, S. Mao, and Y. Liu, "Big data: a survey," Mobile Networks and Applications, vol. 19, no. 2, pp. 171–209, 2014.

[3] Xu Tao, Fu Ge, Tan Huaiyuan, Zhang Hong and Liu Xinran, "Thump Storage: A Management and Analysis System for Structured Big Data", International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC), pp.2424 - 2427,2013

[4] Donald Miner "Map Reduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop".

First Author:**Aqshata Gade**, Student in A. C. Patil College of Engineering, pursuing BE in information technology from Mumbai University.

Second Author:**Aakansha Gupta**,Student in A. C. Patil College of Engineering, pursuing BE in information technology from Mumbai University.

Third Author:**Purva Chaudhary**,Student in A. C. Patil College of Engineering, pursuing BE in information technology from Mumbai University.

Fourth Author:**Amruta Anbhule**,Student in A. C. Patil College of Engineering, pursuing BE in information technology from Mumbai University.