International Journal Of Core Engineering & Management (ISSN: 2348-9510)
Special Issue, NCETICT -2017, St. Johns College of Engineering and Technology, Yemmiganur

HADOOPDB: AN ARCHITECTURAL HYBRID OF MAPREDUCE AND DBMS TECHNOLOGIES FOR ANALYTICAL WORKLOADS

U.U.Veerendra
Associate. Professor, CSE Dept, SJCET, Yemmiganur,
Andhra pradesh.
veerendra.uppara@gmail.com

B.Reshma Banu
PGStudent M.Tech(CSE), SJCET, Yemmiganur,
Andhra Pradesh
reshma.reshu014@gmail.com

K.Hymavathi
PGStudent M.Tech(CSE), SJCET, Yemmiganur,
Andhra pradesh
hymavathi455@gmail.com

Abstract

The production environment for analytical data management applications is rapidly changing. Many enterprises are shifting away from deploying their analytical databases on high-end proprietary machines, and moving towards cheaper, lower-end, commodity hardware, typically arranged in a shared-nothing MPP architecture, often in a virtualized environment inside public or private "clouds". At the same time, the amount of data that needs to be analyzed is exploding, requiring hundreds to thousands of machines to work in parallel to perform the analysis. There tend to be two schools of thought regarding what technology to use for data analysis in such an environment. Proponents of parallel databases argue that the strong emphasis on performance and efficiency of parallel databases makes them wellsuited to perform such analysis. On the other hand, others argue that MapReduce-based systems are better suited due to their superior scalability, fault tolerance, and flexibility to handle unstructured data. In this paper, we explore the feasibility of building a hybrid system that takes the best features from both technologies; the prototype we built approaches parallel databases in performance and efficiency, yet still yields the scalability, fault tolerance, and flexibility of Map Reduce-based systems.

I. INTRODUCTION

The analytical database market currently consists of $3.98 billion of the $14.6 billion database software market 27%, and is growing at a rate of 10.3% annually . As business "bestpractices" trend increasingly towards basing decisions off data and hard facts rather than instinct and theory, the corporate thirst for systems that can manage, process, and granularly analyze data is becoming insatiable. Venture capitalists are very much aware of this trend, and have funded no fewer than a dozen new companies in recent years that build specialized analytical data management software (e.g., Netezza, Vertica, DATAllegro, Greenplum, Aster Data, Infobright, Kickfire, Dataupia, ParAccel, and Exasol), and continue to fund them, even in pressing economic times . At the same time, the amount

of data that needs to be stored and processed by analytical database systems is exploding
This is partly due to the increased automation with which data can be produced (more business processes are becoming digitized), the proliferation of sensors and data-producing devices, Web-scale interactions with customers, and government compliance demands along with strategic corporate initiatives requiring more historical data to be kept online for analysis. It is no longer uncommon to hear of companies claiming to load more than a terabyte of structured data per day into their analytical database system and claiming data warehouses of size more than a petabyte .Given the exploding data problem, all but three of the above mentioned analytical database start-ups deploy their DBMS on a shared-nothing architecture (a collection of independent, possibly virtual, machines, each with local disk and local main memory, connected together on a high-speed network).
This architecture is widely believed to scale the best , especially if one takes hardware cost into account. Furthermore, data analysis workloads tend to consist of many large scan operations, multidimensional aggregations, and star schema joins, all of which are fairly easy to parallelize across nodes in a shared-nothing network. Analytical DBMS vendor leader, Teradata, uses a shared-nothing architecture. Oracle and Microsoft have recently announced shared-nothing analytical DBMS products in their Exadata(1) and Madison projects, respectively. For the purposes of this paper, we will call analytical DBMS systems that deploy on a shared-nothing architecture parallel databases. Parallel databases have been proven to scale really well into the tens of nodes (near linear scalability is not uncommon). However, there are very few known parallel databases deployments consisting of more than one hundred nodes, and to the best of our knowledge, there exists no published deployment of a parallel database with nodes numbering into the thousands.
As the data that needs to be analyzed continues to grow, the number of applications that require more than one hundred nodes is beginning to multiply. Some argue that Map Reduce-based systems are best suited for performing analysis at this scale since they were designed from the beginning to scale to thousands of nodes in a shared-nothing architecture, and have had proven success in Google's internal operation

II. RELATED WORK

There has been some recent work on bringing together ideas from Map Reduce and database systems, however this work focuses mainly on language and interface issues.
The big project at Yahoo , the SCOPE project at Microsoft , and the open source Hive project aim to integrate declarative query constructs from the database community into Map Reduce like software to allow greater data independence, code reusability, and automatic query optimization. Green plum and Aster Data have added the ability to write Map Reduce functions over data stored in their parallel database products . Although these five projects are without question an important step in the hybrid direction, there remains a need for a hybrid solution at the systems level in addition to at the language and interface levels. This paper focuses on such a systems-level hybrid.

III. DESIRED PROPERTIES

In this section we describe the desired properties of a system designed for performing data analysis at the (soon to be more common) petabyte scale.In the following section, we discuss how parallel database systems and Map Reduce-based systems do not meet some subset of these desired properties.

Performance:
Performance is the primary characteristic that commercial database systems use to distinguish themselves from other solutions, with marketing literature often filled with claims that a particular solution is many times faster

than the competition. A factor of ten can make a big difference in the amount, quality, and depth of analysis a system can do. High performance systems can also sometimes result in cost savings. Upgrading to a faster software product can allow a corporation to delay a costly hardware upgrade, or avoid buying additional compute nodes as an application continues to scale. On public cloud computing platforms, pricing is structured in a way such that one pays only for what one uses, so the vendor price increases linearly with the requisite storage, network bandwidth, and compute power.

Hence, if data analysis software product A requires an order of magnitude more compute units than data analysis software product B to perform the same task, then product A will cost (approximately) an order of magnitude more than B. Efficient software has a direct effect on the bottom line

Fault Tolerance :
Fault tolerance in the context of analytical data workloads is measured differently than fault tolerance in the context of transactional workloads. For transactional workloads, a fault tolerant DBMS can recover from a failure without losing any data or updates from recently committed transactions, and in the context of distributed databases, can successfully commit transactions and make progress on a workload even in the face of worker node failures. For read-only queries in analytical workloads, there are neither write transactions to commit, nor updates to lose upon node failure.
Hence, a fault tolerant analytical DBMS is simply one that does not have to restart a query if one of the nodes involved in query processing fails.

IV. BACKGROUND AND SHORTFALLS OF AVAILABLE APPROACHES

Parallel DBMSs MapReduce :
Parellel DBMSs:
Parallel database systems stem from research performed in the late 1980s and most current systems are designed similarly to the early Gamma and Grace parallel DBMS research projects.
These systems all support standard relational tables and SQL, and implement many of the performance enhancing techniques developed by the research community over the past few decades, including indexing, compression (and direct operation on compressed data), materialized views, result caching, and I/O sharing. Most (or) even all tables are partitioned over multiple nodes in a shared nothing cluster; however, the mechanism by which data is partitioned is transparent to the end-user. Parallel databases use an optimizer tailored for distributed workloads that turn SQL commands into a query plan whose execution is divided equally among multiple nodes.
Parallel databases best meet the "performance property" due to the performance push required to compete on the open market, and the ability to incorporate decade's worth of performance tricks published in the database research community.
Parallel databases can achieve especially high performance when administered by a highly skilled DBA who can carefully design, deploy, tune, and maintain the system, but recent advances in automating these tasks and bundling the software into appliance (pre-tuned and pre-configured) offerings have given many parallel databases high performance out of the box.
Parallel databases also score well on the flexible query interface property. Implementation of SQL and ODBC is generally a given, and many parallel databases allow UDFs (although the ability for the query planner and optimizer to parallelize UDFs well over a shared-nothing cluster varies across different implementations)

However, parallel databases generally do not score well on the fault tolerance and ability to operate in a heterogeneous environment properties. Although particular details of parallel database implementations vary, their historical assumptions that failures are rare events and "large" clusters mean dozens of nodes (instead of hundreds or thousands) have resulted in engineering decisions that make it difficult to achieve these properties

MapReduce:

MapReduce was introduced by Dean et. al. in 2004 . Understanding the complete details of how Map Reduce works is not a necessary prerequisite for understanding this paper. In short, Map Reduce processes data distributed (and replicated) across many nodes in a shared-nothing cluster via three basic operations.

First, a set of Map tasks are processed in parallel by each node in the cluster without communicating with other nodes. Next, data is repartitioned across all nodes of the cluster. Finally, a set of Reduce tasks are executed in parallel by each node on the partition it receives. This can be followed by an arbitrary number of additional Map-repartition-Reduce cycles as necessary.

Map Reduce does not create a detailed query execution plan that specifies which nodes will run which tasks in advance; instead, this is determined at runtime. This allows MapReduce to adjust to node failures and slow nodes on the fly by assigning more tasks to faster nodes and reassigning tasks from failed nodes. MapReduce also checkpoints the output of each Map task to local disk in order to minimize the amount of work that has to be redone upon a failure.

By breaking tasks into small, granular tasks, the effect of faults and "straggler" nodes can be minimized

MapReduce has a flexible query interface; Map and Reduce functions are just arbitrary computations written in a general-purpose language. Therefore, it is possible for each task to

do anything on its input, just as long as its output follows the conventions defined by the model

In general, most MapReduce-based systems (such as Hadoop, which directly implements the systems-level details of the MapReduce paper) do not accept declarative SQL. However, there are some exceptions

The biggest issue with MapReduce is performance. By not requiring the user to first model and load data before processing, many of the performance enhancing tools listed above that are used by database systems are not possible. Traditional business data analytical processing, that have standard reports and many repeated queries, is particularly, poorly suited for the one-time query processing model of MapReduce.

Ideally, the fault tolerance and ability to operate in heterogeneous environment properties of MapReduce could be combined with the performance of parallel databases systems. In the following sections, we will describe our attempt to build such a hybrid system.

HADOOPDB :

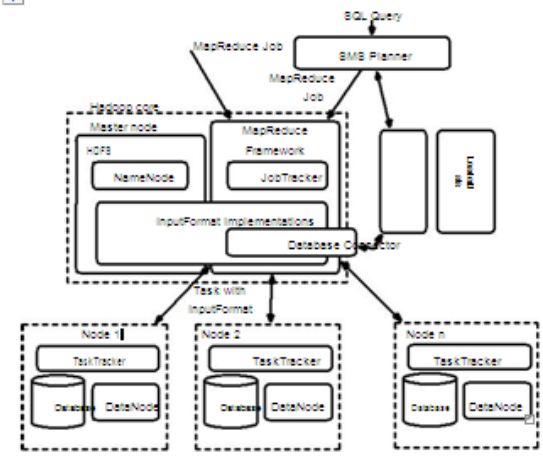
We describe the design of HadoopDB. The goal of this design is to achieve all of the properties described in desired properties:

The basic idea behind HadoopDB is to connect multiple singlenode database systems using Hadoop as the task coordinator and network communication layer. Queries are parallelized across nodes using the MapReduce framework; however, as much of the single node query work as possible is pushed inside of the corresponding node databases. HadoopDB achieves fault tolerance and the ability to operate in heterogeneous environments by inheriting the scheduling and job tracking implementation from Hadoop, yet it achieves the performance of parallel databases by doing much of the query processing inside of the database engine

International Journal Of Core Engineering & Management (ISSN: 2348-9510)
Special Issue, NCETICT -2017, St. Johns College of Engineering and Technology, Yemmiganur

Hadoop Implementation Background :

Hadoop Implementation Background At the heart of HadoopDB is the Hadoop framework,
Hadoop consits of two layers:



HadoopDB's Components

Database Connector:

The Database Connector is the interface between independent database systems residing on nodes in the cluster and Task Trackers. It extends Hadoop's Input Format class and is part of the Input Format Implementations library.

Each MapReduce job supplies the Connector with an SQL query and connection parameters such as: which JDBC driver to use, query fetch size and other query tuning parameters. The Connector connects to the database, executes the SQL query and returns results as key-value pairs. The Connector could theoretically connect to any JDBC-compliant database that resides in the cluster.

However, different databases require different read query optimizations. We implemented connectors for MySQL and PostgreSQL. In the future we plan to integrate other databases including open-source column-store databases such as MonetDB and InfoBright. By extending Hadoop's Input Format, we integrate seamlessly with Hadoop's MapReduce Framework. To the framework, the databases are data sources similar to data blocks in HDFS.

Catalog :

The catalog maintains meta information about the databases.

This includes the following**:**

(i) Connection parameters such as database location, driver class and credentials

(ii) Metadata such as data sets contained in the cluster, replica locations, and data partitioning properties

The current implementation of the HadoopDB catalog stores its meta information as an XML file in HDFS. This file is accessed by the Job Tracker and Task Trackers to retrieve information necessary to schedule tasks and process data needed by a query. In the future, we plan to deploy the catalog as a separate service that would work in a way similar to Hadoop's Name Node

Data Loader :
The Data Loader is responsible for globally repartitioning data on a given partition key upon loading and breaking apart single node data into multiple smaller partitions or chunks and finally bulk-loading the single-node databases with the chunks.

Global Hasher and Local Hasher :
The Global Hasher executes a custom made Map Reduce job over Hadoop that reads in raw data files stored in HDFS and repartitions them into as many parts as the number of nodes in the cluster. The repartitioning job does not incur the sorting overhead of typical MapReduce jobs
The Local Hasher then copies a partition from HDFS into the local file system of each node and secondarily partitions the file into smaller sized chunks based on the maximum chunk size setting
The hashing functions used by both the Global Hasher and the Local Hasher differ to ensure chunks are of a uniform size. They also differ from Hadoop's default hash-partitioning function to ensure better load balancing when executing Map Reduce jobs over the data.

Summary :
Hadoop DB does not replace Hadoop. Both systems coexist enabling the analyst to choose the appropriate tools for a given dataset and task. Through the performance benchmarks in the following sections, we show that using an efficient database storage layer cuts down on data processing time especially on tasks that require complex query processing over structured data such as joins. We also show that Hadoop DB is able to take advantage of the fault-tolerance and the ability to run on heterogeneous environments that comes naturally with Hadoop-style systems