Hive – A Petabyte Scale Data Warehouse Using Hadoop

Thusoo et. Al.

Presented by: Manoj Sharma

UNIVERSITY OF WATERLOO

Basis

- Data is everywhere and increasing day by day

There are many sources that predict exponential data growth toward 2020 and beyond.

Reference - https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/

Basis

- Increasing data size $\propto$ Increased complexity to handle it
- Increased processing times
- Drive for parallel database models.

RoadMap -

- Hadoop Outline and Challenges
- Hive and its need
- Data Model and Type System
- HiveQL
- Data Storage and access
- System Architecture
- Conclusion & Few Thoughts

Hadoop Outline

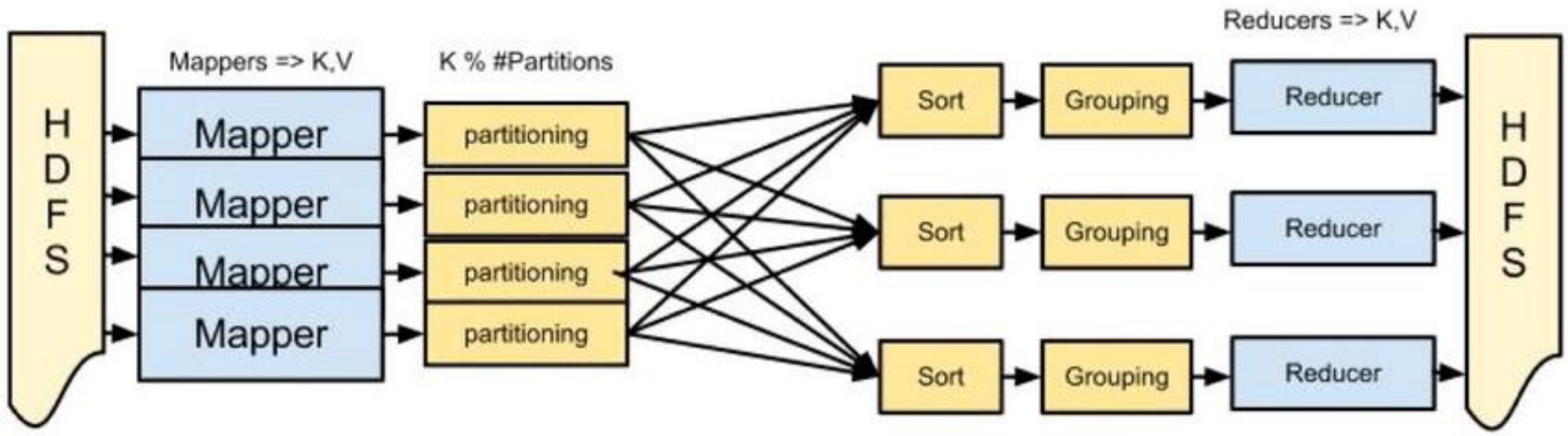
- What is Hadoop ?
- Coordinated distributed parallel processing of data.
- Actively used by many companies like Facebook, Yahoo etc.,

Ref – Trademark logo of haddop inc

UNIVERSITY OF WATERLOO

Hadoop Outline

- **MapReduce** is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.



Ref - https://www.researchgate.net/figure/The-Hadoop-MapReduce-Pipeline_fig3_310036684

Hadoop Challenges

- HDFS is used to store the data and MR is used to process the data
- MR are set of programs written by users
- No Flexibility – error prone codes, programmer need knowledge of system architecture.

- At Facebook

  - HDFS was providing proper storage abstraction and scaling.

  - Most of data was unstructured
- Need for HIVE !!

HIVE

- Hive was developed by Facebook (around Jan 2007)
- Requisites

  - Unstructured data handing

  - Faster processing

  - Minimum/No user intervention

  - Flexible SQL type support

HIVE

- Query processing engine for HDFS. (Can run as a layer on HDFS)
- HiveQL – Supports queries expressed in SQL like declarative language
- Extensible framework to support customizable file and data formats

HiveQL

- Subset of SQL queries are supported
- SQL clauses like – FROM, joins -INNER, OUTER, RIGHT OUTER, GROUP BY, UNION ALL, aggregations etc., are supported
- Example
  - SELECT t1.a, t2.b FROM t1 JOIN t2 ON (t1.a = t2.b);

  Equality predicates were only supported in a join query

  Recent HIVE releases have support for resolving implicit joins

HiveQL – MR support

- Supports MapReduce based analysis of data
- Example - Canonical word count on a table of documents

FROM (MAP doctext USING 'python wc_mapper.py' AS (word, cnt) FROM docs CLUSTER BY word) a REDUCE word, cnt USING 'python wc_reduce.py';

Example - Find all the actions in a session sorted by time

FROM (FROM session_table SELECT sessionid, tstamp, data DISTRIBUTE BY sessionid SORT BY tstamp) a REDUCE sessionid, tstamp, data USING 'session_reducer.sh';

Data Model and Type System

- Hive provides data abstraction to user.

  - abstraction via row and column layout of data (similar to RDBMS tables)

- Supports

  - primitive data types – int, float, double, string

  - complex types – maps, lists and struct

  - nested structures

  - provides '.' and '[ ]' operator support to access attributes of structured datatypes.

Example

```
CREATE TABLE T(a int, b list<map<string, struct<p1:int, p2:int>>);
```

Data Storage

- Tables are logical units in Hive
- Table metadata associates the data in a table to hdfs directories
- Primary data units and their mappings –

  Tables – stored in a directory in hdfs

  Partitions – stored in the sub-directory of table's directory

  Buckets – stored in a file within the partition's or table's directory

  Example – Creating a partitioned table

```
CREATE TABLE test_part(c1 string, c2 int) PARTITIONED BY (ds string, hr int);
```

Serialization/DeSerialization (SerDe)

- Tables are serialized and deserialized using default serializers and deserializers in Hive. Default is LazySerDe
- Custom SerDe can be provided by users.

  - customized delimiters, regex support for parsing columns from rows.
- Any arbitrary data format and types encoded can be plugged into Hive
- Example:

```
add jar /jars/myformat.jar;
CREATE TABLE t2 ROW FORMAT SERDE 'com.myformat.MySerDe';
```

File Formats

- Hadoop files can be stored in different formats

  Example –

  TextInputFormat for text files, SequenceFileInputFormat for binary files, etc.

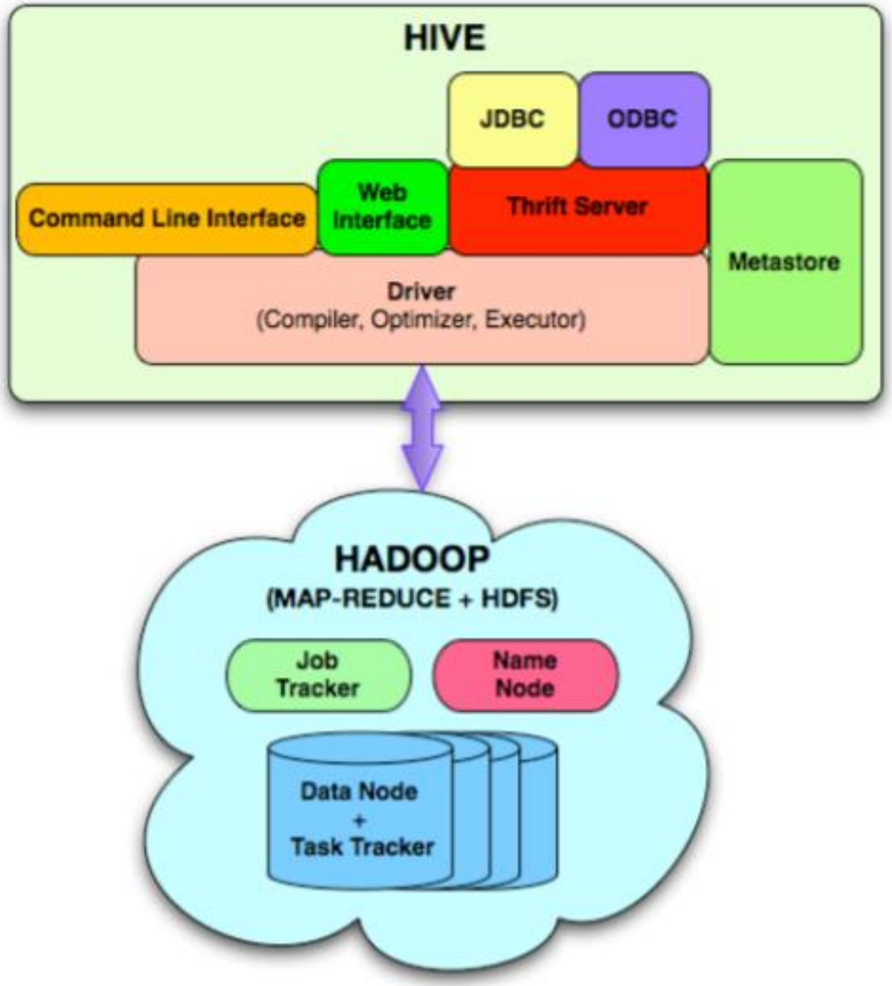
- Users can implement their own formats and associate them to a table

- Format can be specified when the table is created and no restrictions are imposed by Hive

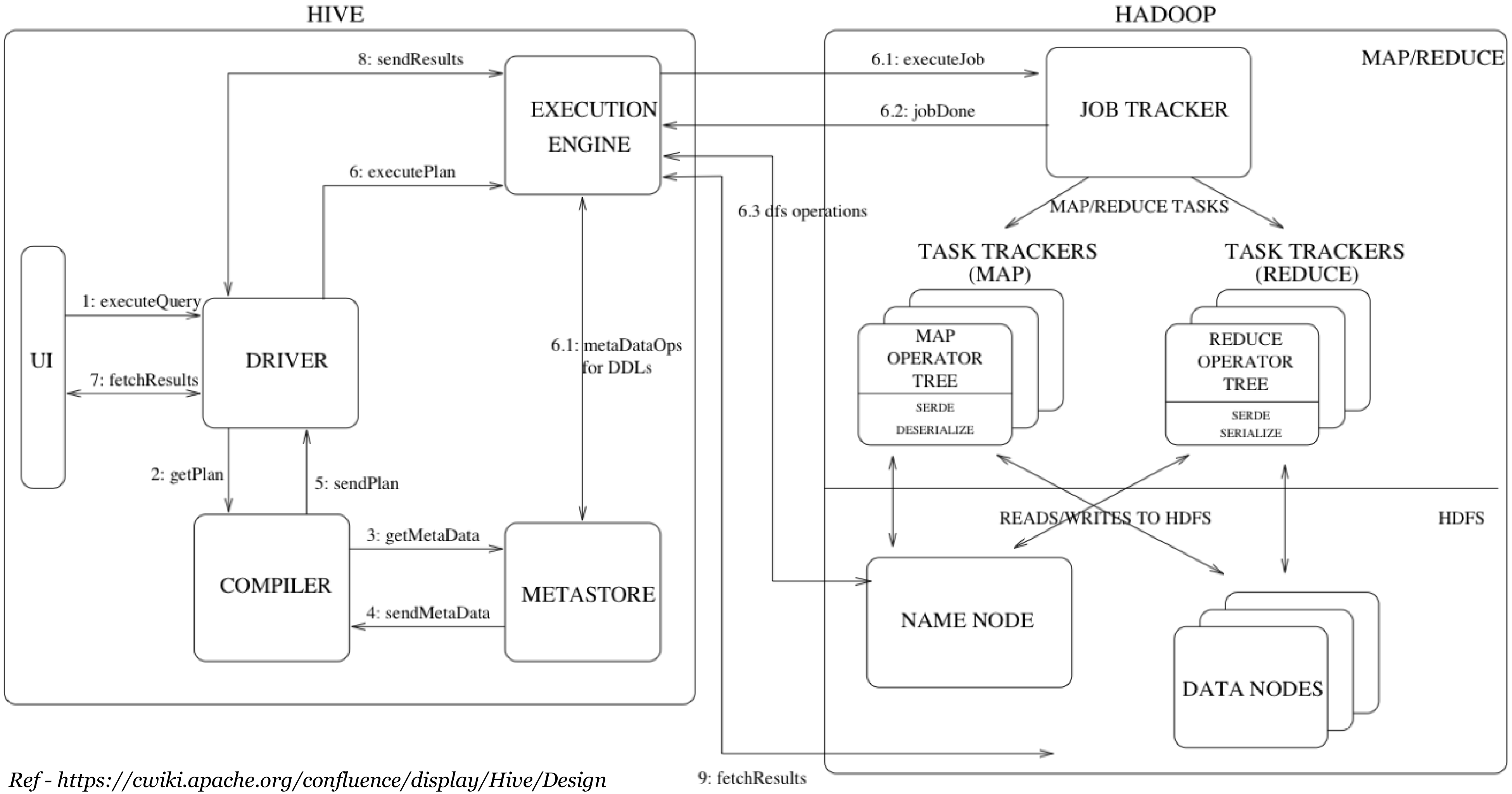
  Example –

  CREATE TABLE dest(key INT, value STRING) STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.SequenceFileInputFormat' OUTPUTFORMAT 'org.apache.hadoop.mapred.SequenceFileOutputFormat‘;

Hive Components and Architecture



Query Flow



Ref - https://cwiki.apache.org/confluence/display/Hive/Design

Query Flow

- HiveQL statement submitted via the CLI, the web UI or external client using thrift, odbc or jdbc interfaces.
- Driver first passes the query to compiler – Typical parse, type check and semantic analysis is done using the metadata
- Compiler generates a logical plan
- It is then optimized through rule based optimizer to generate a DAG of map-reduce and hdfs tasks
- Execution engine then execute these tasks in the order of their dependencies using Hadoop

MetaStore

- System catalog for Hive. Stores all the information about the tables, their partitions, schemas, columns and types, table locations, SerDe information etc.
- Can be queried or modified using a thrift interface
- This information is stored on traditional RDBMS
- Uses an open source ORM layer called DataNucleus to convert object representations to relational schema and vice versa
- Scalability of the Metastore server is ensured by making sure no metadata calls are made from mappers or reducers of a job
- Xml plan files are generated by compiler containing all the runtime information

Query Compiler

- Parser: Uses Antlr to generate abstract syntax tree (AST) for the query
- Semantic Analyser

  - Compiler fetches all the required information from metastore

  - Verifying column names, type-checking and implicit type conversions are done

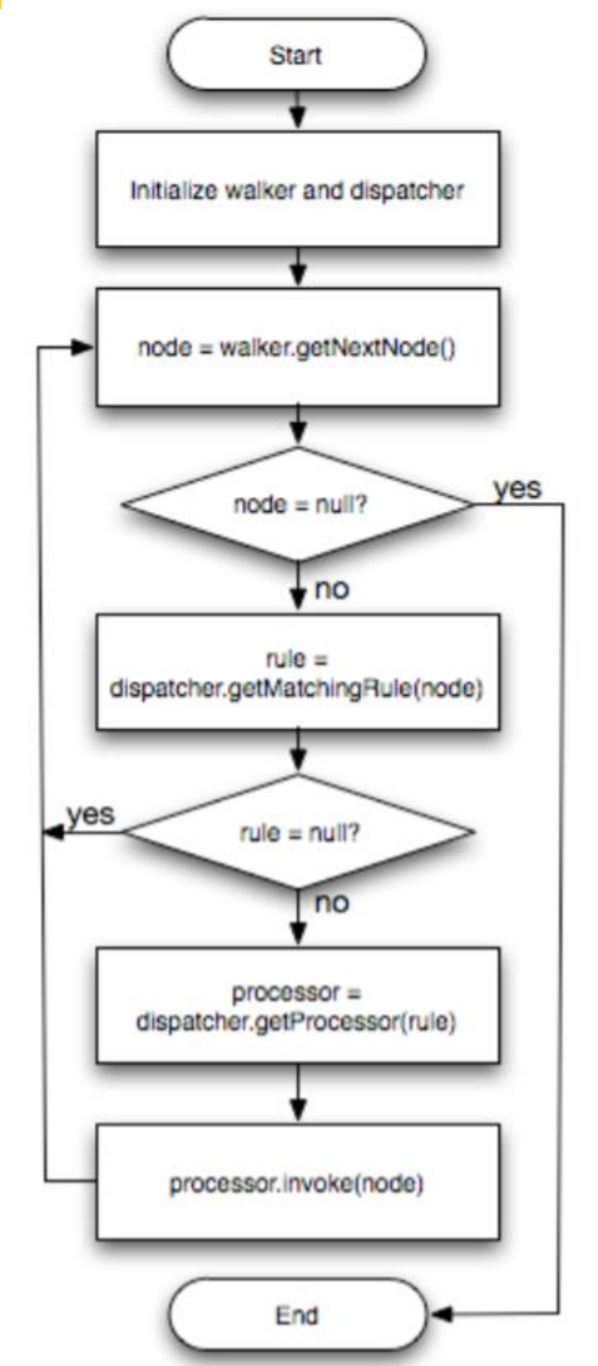
  - Transforms the AST to an internal query representation – Query Block (QB) tree.
- Logical Plan generator –

  - Convert internal query to logical plan – tree of operators or operator DAG.

  - Some operators are relational algebra operators like ‘filter’, ‘join’, etc. Some are Hive specific say, reduceSink operator – occurs at map-reduce boundary.

Query Compiler

- Optimizer - Contains a chain of transformations to transform the plan for improved performance
- Walks on the operator DAG and does processing actions when certain rules or conditions are satisfied
- Five main interfaces involved during the walk - Node, GraphWalker, Dispatcher, Rule and Processor.



RSITY OF
ERLOO

Query Compiler

- Typical Transformations

  - Column pruning - only the columns that are needed in the query processing are actually projected out of the row

  - Predicate pushdown - Predicates are pushed down to the scan if possible so that rows can be filtered early in the processing

  - Partition pruning - Predicates on partitioned columns are used to prune out files of partitions that do not satisfy the predicate

  - Map side joins – Small tables in a join are replicated in all the mappers and joined with other tables. Eg: SELECT /*+ MAPJOIN(t2) */ t1.c1, t2.c1 FROM t1 JOIN t2 ON(t1.c2 = t2.c2);

  - Join reordering – Larger tables are streamed in the reducer and smaller tables are kept in memory

Query Compiler

- Supports few optimizations
- Repartitioning of data to handle skews in GROUP BY processing

  - Most of the data might get sent to few reducers

  - Use two-stage map-reduce

    Stage one - Random distribution of data to the reducers to compute partial aggregations

    Stage two - Partial aggregations are distributed on the GROUP BY columns to the reducers in the second MR stage

    Triggered in Hive by setting a parameter – set hive.groupby.skewindata=true;

- Hash based partial aggregations in the mappers – Hive does hash based partial aggregations within the mappers to reduce the data sent to the reducers

  - This reduces the time spent in sorting and merging data and gives a performance gain.

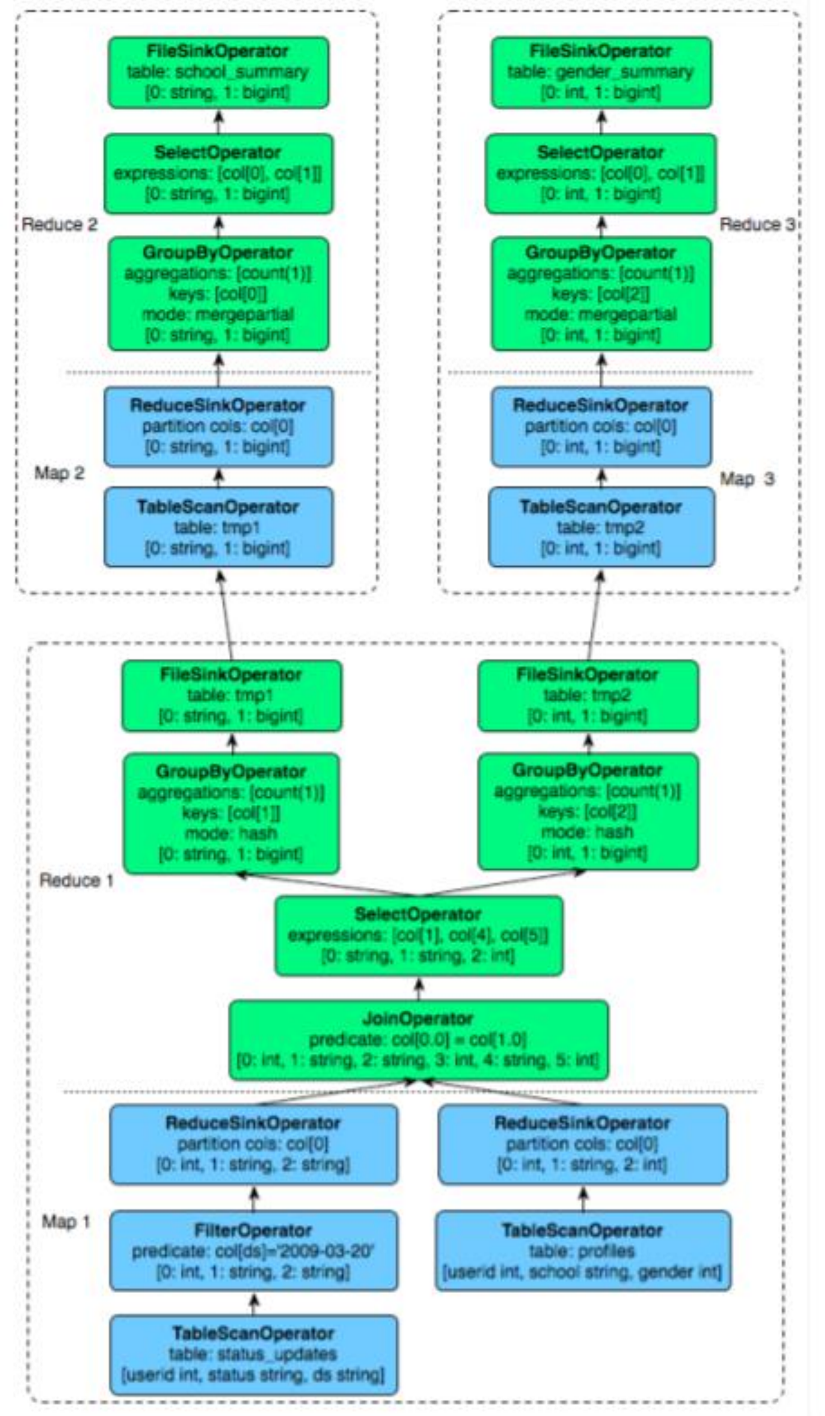
  - Controlled by parameter – hive.map.aggr.hash.percentmemory

Query Plan

- Physical plan generator – Logical plan after optimization is split into multiple map/reduce and hdfs tasks

- A Multi-table insert query –

  FROM
  (SELECT a.status, b.school, b.gender FROM
  status_updates a JOIN **profiles** b ON
  (a.userid = b.userid AND a.ds='2009-03-20')) subq1
  INSERT OVERWRITE TABLE **gender_summary**
  PARTITION(ds='2009-03-20')
  SELECT subq1.gender, COUNT(1) GROUP BY subq1.gender
  INSERT OVERWRITE TABLE school_summary
  PARTITION(ds='2009-03-20')
  SELECT subq1.school, COUNT(1) GROUP BY subq1.school;



Execution Engine

- The tasks are executed in the order of their dependencies
- A map/reduce task first serializes its part of the plan into a plan.xml file
- This file is added to the job cache for the task and ExecMapper and ExecReducer instances are spawned using Hadoop
- Each of these classes executes relevant parts of the DAG
- Final results are stored in a temporary location
- At the end of entire query, final data is either moved to a desired location or fetched from the temporary location

Related Work

SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets

Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey,
Darren Shakib, Simon Weaver, Jingren Zhou
Microsoft Corporation
{rchaiken, bobjen, palarson, brams, darrens, sweaver, jrzhou}@microsoft.com

ABSTRACT

Companies providing cloud-scale services have an increasing need to store and analyze massive data sets such as search logs and click streams. For cost and performance reasons, processing is typically done on large clusters of shared-nothing commodity machines. It is imperative to develop a programming model that hides the complexity of the underlying system but provides flexibility by allowing users to extend functionality to meet a variety of requirements.

In this paper, we present a new declarative and extensible scripting language, SCOPE (Structured Computations Optimized for Parallel Execution), targeted for this type of massive data analysis. The language is designed for ease of use with no explicit parallelism, while being amenable to efficient parallel execution on large clusters. SCOPE borrows several features from SQL. Data is modeled as sets of rows composed of typed columns. The select statement is retained with inner joins, outer joins, and aggregation allowed. Users can easily define their own functions and implement their own versions of operators: extractors (parsing and constructing rows from a file), processors (row-wise processing), reducers (group-wise processing), and combiners (combining rows from two inputs). SCOPE supports nesting of expressions but also allows a computation to be specified as a series of steps, in a manner often preferred by programmers. We also describe how scripts are compiled into efficient, parallel execution plans and executed on large clusters.

1. INTRODUCTION

Internet companies store and analyze massive data sets, such as search logs, web content collected by crawlers, and click streams collected from a variety of web services. Such analysis is becoming increasingly valuable for business in a variety of ways, for example, to improve service quality and support novel features, to detect changes in patterns over time, and to detect fraudulent activity.

Due to the size of these data sets, traditional parallel database solutions can be prohibitively expensive. To be able to perform this type of web-scale analysis in a cost-effective manner, several

companies have developed distributed data storage and processing systems on large clusters of shared-nothing commodity servers, including Google's File System [8] , Bigtable [3], Map-Reduce [5], Hadoop [1], Yahoo!'s Pig system [2], Ask.com's Neptune [4], and Microsoft's Dryad [6]. A typical cluster consists of hundreds or thousands of commodity machines connected via a high-bandwidth network. It is challenging to design a programming model that enables users to easily write programs that can efficiently and effectively utilize all resources in such a cluster and achieve maximum degree of parallelism.

The *Map-Reduce* programming model provides a good abstraction of group-by-aggregation operations over a cluster of machines. The programmer provides a map function that performs grouping and a reduce function that performs aggregation. The underlying run-time system achieves parallelism by partitioning the data and processing different partitions concurrently using multiple machines.

However, this model has its own set of limitations. Users are forced to map their applications to the map-reduce model in order to achieve parallelism. For some applications this mapping is very unnatural. Users have to provide implementations for the map and reduce functions, even for simple operations like projection and selection. Such custom code is error-prone and hardly reusable. Moreover, for complex applications that require multiple stages of map-reduce, there are often many valid evaluation strategies and execution orders. Having users implement (potentially multiple) map and reduce functions is equivalent to asking users specify physical execution plans directly in database systems. The user plans may be suboptimal and lead to performance degradation by orders of magnitude.

In this paper, we present a new scripting language, SCOPE (Structured Computations Optimized for Parallel Execution), targeted for large-scale data analysis that is under development at Microsoft. Many users are familiar with relational data and SQL. SCOPE intentionally builds on this knowledge but with simplifications suited for the new execution environment. Users familiar with SQL require little or no training to use SCOPE. Like SQL, data is modeled as sets of rows composed of typed columns. Every rowset has a well-defined schema. The SCOPE runtime provides implementations of many standard physical operators, saving users from implementing similar functionality repetitively. SCOPE is being used daily for a variety of data analysis and data mining applications inside Microsoft.

SCOPE is a declarative language. It allows users to focus on the

Pig Latin: A Not-So-Foreign Language for Data Processing

Christopher Olston[*] Yahoo! Research
Benjamin Reed[†] Yahoo! Research
Utkarsh Srivastava[‡] Yahoo! Research
Ravi Kumar[§] Yahoo! Research
Andrew Tomkins[¶] Yahoo! Research

ABSTRACT

There is a growing need for ad-hoc analysis of extremely large data sets, especially at internet companies where innovation critically depends on being able to analyze terabytes of data collected every day. Parallel database products, e.g., Teradata, offer a solution, but are usually prohibitively expensive at this scale. Besides, many of the people who analyze this data are entrenched procedural programmers, who find the declarative, SQL style to be unnatural. The success of the more procedural *map-reduce* programming model, and its associated scalable implementations on commodity hardware, is evidence of the above. However, the map-reduce paradigm is too low-level and rigid, and leads to a great deal of custom user code that is hard to maintain, and reuse.

We describe a new language called *Pig Latin* that we have designed to fit in a sweet spot between the declarative style of SQL, and the low-level, procedural style of map-reduce. The accompanying system, Pig, is fully implemented, and compiles Pig Latin into physical plans that are executed over *Hadoop*, an open-source, map-reduce implementation. We give a few examples of how engineers at Yahoo! are using Pig to dramatically reduce the time required for the development and execution of their data analysis tasks, compared to using Hadoop directly. We also report on a novel debugging environment that comes integrated with Pig, that can lead to even higher productivity gains. Pig is an open-source, Apache-incubator project, and available for general use.

Categories and Subject Descriptors:
H.2.3 Database Management: Languages

General Terms: Languages.

[*]olston@yahoo-inc.com
[†]breed@yahoo-inc.com
[‡]utkarsh@yahoo-inc.com
[§]ravikuma@yahoo-inc.com
[¶]atomkins@yahoo-inc.com

1. INTRODUCTION

At a growing number of organizations, innovation revolves around the collection and analysis of enormous data sets such as web crawls, search logs, and click streams. Internet companies such as Amazon, Google, Microsoft, and Yahoo! are prime examples. Analysis of this data constitutes the innermost loop of the product improvement cycle. For example, the engineers who develop search engine ranking algorithms spend much of their time analyzing search logs looking for exploitable trends.

The sheer size of these data sets dictates that it be stored and processed on highly parallel systems, such as shared-nothing clusters. Parallel database products, e.g., Teradata, Oracle RAC, Netezza, offer a solution by providing a simple SQL query interface and hiding the complexity of the physical cluster. These products however, can be prohibitively expensive at web scale. Besides, they wrench programmers away from their preferred method of analyzing data, namely writing imperative scripts or code, toward writing declarative queries in SQL, which they often find unnatural, and overly restrictive.

As evidence of the above, programmers have been flocking to the more procedural *map-reduce* [4] programming model. A map-reduce program essentially performs a group-by-aggregation in parallel over a cluster of machines. The programmer provides a map function that dictates how the grouping is performed, and a reduce function that performs the aggregation. What is appealing to programmers about this model is that there are only two high-level declarative primitives (map and reduce) to enable parallel processing, but the rest of the code, i.e., the map and reduce functions, can be written in any programming language of choice, and without worrying about parallelism.

Unfortunately, the map-reduce model has its own set of limitations. Its one-input, two-stage data flow is extremely rigid. To perform tasks having a different data flow, e.g., joins or *n* stages, inelegant workarounds have to be devised. Also, custom code has to be written for even the most common operations, e.g., projection and filtering. These factors lead to code that is difficult to reuse and maintain, and in which the semantics of the analysis task are obscured. Moreover, the opaque nature of the map and reduce functions

SITY OF
RLOO

Recent Work and Performance analysis

Major Technical Advancements in Apache Hive

Yin Huai[1] Ashutosh Chauhan[2] Alan Gates[2] Gunther Hagleitner[2] Eric N. Hanson[3]
Owen O'Malley[2] Jitendra Pandey[2] Yuan Yuan[1] Rubao Lee[1] Xiaodong Zhang[1]

[1]The Ohio State University [2]Hortonworks Inc. [3]Microsoft

[1]{huai, yuanyu, liru, zhang}@cse.ohio-state.edu
[2]{ashutosh, gates, ghagleitner, owen, jitendra}@hortonworks.com
[3]ehans@microsoft.com

ABSTRACT

Apache Hive is a widely used data warehouse system for Apache Hadoop, and has been adopted by many organizations for various big data analytics applications. Closely working with many users and organizations, we have identified several shortcomings of Hive in its file formats, query planning, and query execution, which are key factors determining the performance of Hive. In order to make Hive continuously satisfy the requests and requirements of processing increasingly high volumes data in a scalable and efficient way, we have set two goals related to storage and runtime performance in our efforts on advancing Hive. First, we aim to maximize the effective storage capacity and to accelerate data accesses to the data warehouse by updating the existing file formats. Second, we aim to significantly improve cluster resource utilization and runtime performance of Hive by developing a highly optimized query planner and a highly efficient query execution engine. In this paper, we present a community-based effort on technical advancements in Hive. Our performance evaluation shows that these advancements provide significant improvements on storage efficiency and query execution performance. This paper also shows how academic research lays a foundation for Hive to improve its daily operations.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems

Keywords

Databases; Data Warehouse; Hadoop; Hive; MapReduce

1. INTRODUCTION

Apache Hive is a data warehouse system for Apache Hadoop [1]. It has been widely used in organizations to manage and process large volumes of data, such as eBay, Facebook, LinkedIn, Spotify, Taobao, Tencent, and Yahoo!. As an open source project, Hive has a strong technical development community working with widely than 100 developers have made technical efforts to improve Hive on more than 3000 issues. With its rapid development pace, Hive has been significantly updated by new innovations and research since the original Hive paper [45] was published four years ago. We will present its major technical advancements in this paper.

Hive was originally designed as a translation layer on top of Hadoop MapReduce. It exposes its own dialect of SQL to users and translates data manipulation statements (queries) to a directed acyclic graph (DAG) of MapReduce jobs. With an SQL interface, users do not need to write tedious and sometimes difficult MapReduce programs to manipulate data stored in Hadoop Distributed Filesystem (HDFS).

This highly abstracted SQL interface significantly improves the productivity of data management in Hadoop and accelerates the adoption of Hive. The efficiency and productivity of Hive are largely affected by how its data warehouse layer is designed, implemented, and optimized to best utilize the underlying data processing engine (e.g. Hadoop MapReduce) and HDFS. In order to make Hive continuously satisfy requirements of processing increasingly high volumes of data in a scalable and efficient way, we must improve both data storage as well as query execution aspect of Hive. First, Hive should be able to store datasets managed by it in an efficient way which guarantees both storage efficiency as well as fast data access. Second, Hive should be able to generate highly optimized query plans and execute them using a query execution model that utilizes hardware resources well.

Closely working with many users and organizations, the Hive development community has identified several shortcomings in its file formats, query planning, and query execution, which largely determine performance of queries submitted to Hive. In this paper, we present a community-based effort on addressing these several shortcomings with strong support and scientific basis from several academic research results. The main contributions of this paper are:

1. A new file format, Optimized Record Columnar File (ORC File), has been added to Hive which provides high storage and data access efficiency with low overhead.

Processing Performance on Apache Pig, Apache Hive and MySQL Cluster

Ammar Fuad, Alva Erwin, Heru Purnomo Ipung

Information Technology, Swiss German University
Edutown BSD City, Tangerang 15339, Indonesia
[1]ammar.fuad[at]student.sgu.ac.id
[2]alva_erwin[at]yahoo.com
[3]heru.ipung[at]sgu.ac.id

Abstract—**MySQL Cluster is a famous clustered database that is used to store and manipulate data. The problem with MySQL Cluster is that as the data grows larger, the time required to process the data increases and additional resources may be needed. With Hadoop and Hive and Pig, processing time can be faster than MySQL Cluster. In this paper, three data testers with the same data model will run simple queries and to find out at how many rows Hive or Pig is faster than MySQL Cluster. The data model taken from GroupLens Research Project [12] showed a result that Hive is the most appropriate for this data model in a low-cost hardware environment.**

Keywords— **Hadoop; Hive; Pig; MySQL; MySQL Cluster; Processing big data;**

I. Introduction

Hadoop is a popular open-source implementation of MapReduce that is used by academics, goverments, and industrial organizations. Hadoop can be used for storing large data and for processing data such as data mining, report generation, file analysis, web indexing, and bioinformatic research [2].

MySQL Cluster is a MySQL server with one or more data storages and management servers to configure the cluster and data replication. MySQL Cluster provides 99.999% availability to the data. MySQL Cluster is designed for distributed node architecture with no single point of failure. It consists of multiple nodes that are distributed across machines to make sure the system can work, even in case a node having a problem such as network failure [11].

Apache Hive and Apache Pig are open source programs for analyzing large data sets in a high-level language. Apache Pig is a simple query algebra that lets the user declare data transformation to files or groups of files. Hive is data

This paper presents the processing time of Hive, Pig, and MySQL Cluster on a simple data model with simple queries while the data is growing. Section 3 discusses a proposed method. Section 4 shows the results and explanations. And the last section, section 5 provides a conclusion and possible future work.

II. Related Works

Hive and Pig are a high-level language for processing data. Both are used for working with petabyte scale data [5][9]. Working at low-scale data can also be done with Hive or Pig. But processing low-scale data can consume more time with Hive or Pig rather than using other data processing software such as MySQL. As the data grows larger, MySQL requires more time to process the data until it reaches a point where Hive or Pig is faster than MySQL.

But when exactly do users need to change from MySQL to Hive or Pig for a faster processing time? This research indicates to users when they can switch to Hive or Pig as their rows of data become bigger. This test is done in a low-cost hardware environment.

III. Proposed Method

There are three aspects that will determine the result: 1) the data set file size (how many rows); 2) query statements; 3) query average time. There are three data sets with the same data model. The first data set is called ml100k (movie lens 100,000 rows) containing a total of 102,580 rows. The second data set is called ml1m containing a total of 1,075,611 rows. The last data set is called ml10m containing a total of 10,069,372 rows.

A. Hadoop Environment

Conclusion

- Hive extensively used for large data processing. Example - Facebook, Yahoo
- Easy way to process large scale data
- SQL-like query support
- Flexibility to Hadoop user
- Custom support

Few thoughts

- Why does Hive provide file based data representation rather than block ?
- Can file formats provide – faster access to data (indexable), metadata per line of each file ? What is the feasibility of index based structures.
- Why is Optimizer scope restricted to Rule based ? What can be done to make it cost based ?
- Hive required metastore server to host dictionary data. Can this be a bottleneck ?
- Intermediate result set management. (result sets are flushed to disk and read again. If cacheable ? What are provisions.)
- Subquery elimination, predicate rewrite feasibility.
-

References

- Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H., and Murthy, R. 2010. Hive — A petabyte scale data warehouse using Hadoop. In Proceedings of the International Conference on Data Engineering. 996–1005.
- https://cwiki.apache.org/confluence/display/Hive/Design
- http://www.apache.org/hadoop/hive