Hive – A Petabyte Scale Data Warehouse Using Hadoop

Thusoo et. Al.

Presented by: Manoj Sharma
Data is everywhere and increasing day by day

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 hadoop inc
**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';
```
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.
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

COPE: Easy and Efficient Parallel Processing of Massive Data Sets
Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shaktiv, Simon Weaver, Jingren Zhou
Microsoft Corporation
{rchaiken, bobjen, palarson, btrams, dweaver, jzhou}@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 servers. In this paper, we develop 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.

In this paper, we present a new streaming and scalable
language, COPE. Structured Computation (defined for
Parallel Execution) targeted for this type of massive data anal-
ysis. The language is designed for ease of use with no explicit par-
allelism and execution semantics required to achieve high perfor-
mance on large clusters. COPE borrows several features from SQL. Data is modeled as a sequence of typed columnar vectors. The select statement is rewritten with inner joins, outer joins, and aggregation allowed. Users can easily define their own functions and imple-
ment their own version of operators: extractors (gathering and con-
structing rows from a file), processors (non-interactive, processing, read-only, producing, combining, and combinations of two inputs). COPE supports nesting of expressions but also allows a computation to be specified as a series of steps, in a manner often provided by programming languages. 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
Permission to make digital or hard copies of portions of this paper for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for portions of this work are owned by others than VLDB Foundation. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

COPE: Easy and Efficient Parallel Processing of Massive Data Sets
Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shaktiv, Simon Weaver, Jingren Zhou
Microsoft Corporation
{rchaiken, bobjen, palarson, btrams, dweaver, jzhou}@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 servers. In this paper, we develop 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.

In this paper, we present a new streaming and scalable
language, COPE. Structured Computation (defined for
Parallel Execution) targeted for this type of massive data anal-
ysis. The language is designed for ease of use with no explicit par-
allelism and execution semantics required to achieve high perfor-
mance on large clusters. COPE borrows several features from SQL. Data is modeled as a sequence of typed columnar vectors. The select statement is rewritten with inner joins, outer joins, and aggregation allowed. Users can easily define their own functions and imple-
ment their own version of operators: extractors (gathering and con-
structing rows from a file), processors (non-interactive, processing, read-only, producing, combining, and combinations of two inputs). COPE supports nesting of expressions but also allows a computation to be specified as a series of steps, in a manner often provided by programming languages. 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
Permission to make digital or hard copies of portions of this paper for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for portions of this work are owned by others than VLDB Foundation. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

COPE: Easy and Efficient Parallel Processing of Massive Data Sets
Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shaktiv, Simon Weaver, Jingren Zhou
Microsoft Corporation
{rchaiken, bobjen, palarson, btrams, dweaver, jzhou}@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 servers. In this paper, we develop 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.

In this paper, we present a new streaming and scalable
language, COPE. Structured Computation (defined for
Parallel Execution) targeted for this type of massive data anal-
ysis. The language is designed for ease of use with no explicit par-
allelism and execution semantics required to achieve high perfor-
mance on large clusters. COPE borrows several features from SQL. Data is modeled as a sequence of typed columnar vectors. The select statement is rewritten with inner joins, outer joins, and aggregation allowed. Users can easily define their own functions and imple-
ment their own version of operators: extractors (gathering and con-
structing rows from a file), processors (non-interactive, processing, read-only, producing, combining, and combinations of two inputs). COPE supports nesting of expressions but also allows a computation to be specified as a series of steps, in a manner often provided by programming languages. 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
Permission to make digital or hard copies of portions of this paper for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for portions of this work are owned by others than VLDB Foundation. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

COPE: Easy and Efficient Parallel Processing of Massive Data Sets
Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shaktiv, Simon Weaver, Jingren Zhou
Microsoft Corporation
{rchaiken, bobjen, palarson, btrams, dweaver, jzhou}@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 servers. In this paper, we develop 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.

In this paper, we present a new streaming and scalable
language, COPE. Structured Computation (defined for
Parallel Execution) targeted for this type of massive data anal-
ysis. The language is designed for ease of use with no explicit par-
allelism and execution semantics required to achieve high perfor-
mance on large clusters. COPE borrows several features from SQL. Data is modeled as a sequence of typed columnar vectors. The select statement is rewritten with inner joins, outer joins, and aggregation allowed. Users can easily define their own functions and imple-
ment their own version of operators: extractors (gathering and con-
structing rows from a file), processors (non-interactive, processing, read-only, producing, combining, and combinations of two inputs). COPE supports nesting of expressions but also allows a computation to be specified as a series of steps, in a manner often provided by programming languages. 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
Permission to make digital or hard copies of portions of this paper for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for portions of this work are owned by others than VLDB Foundation. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

COPE: Easy and Efficient Parallel Processing of Massive Data Sets
Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shaktiv, Simon Weaver, Jingren Zhou
Microsoft Corporation
{rchaiken, bobjen, palarson, btrams, dweaver, jzhou}@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 servers. In this paper, we develop 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.

In this paper, we present a new streaming and scalable
language, COPE. Structured Computation (defined for
Parallel Execution) targeted for this type of massive data anal-
ysis. The language is designed for ease of use with no explicit par-
allelism and execution semantics required to achieve high perfor-
mance on large clusters. COPE borrows several features from SQL. Data is modeled as a sequence of typed columnar vectors. The select statement is rewritten with inner joins, outer joins, and aggregation allowed. Users can easily define their own functions and imple-
ment their own version of operators: extractors (gathering and con-
structing rows from a file), processors (non-interactive, processing, read-only, producing, combining, and combinations of two inputs). COPE supports nesting of expressions but also allows a computation to be specified as a series of steps, in a manner often provided by programming languages. 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
Permission to make digital or hard copies of portions of this paper for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for portions of this work are owned by others than VLDB Foundation. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Recent Work and Performance analysis

Major Technical Advancements in Apache Hive

Yin Huai1 Ashutosh Chauhan2 Alan Gates3 Gunther Hageliet4 Eric N. Hanson5
Owen O'Malley6 Jiendra Pandey7 Yuan Yuan8 Rubao Lee9 Xiaodong Zhang10
1 The Ohio State University 2 Hortonworks Inc. 3 Microsoft
(ahuai, yuanyu, liu, zhang)@cse.ohio-state.edu 2ashutosh, gates, ghageliet, owen, jiterca@hortonworks.com
3ehans@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. 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 of 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.

Recent Work and Performance analysis

Processing Performance on Apache Pig, Apache Hive and MySQL Cluster

Anamar Fauad, Alva Erwin, Hera Purnomo Impu
Information Technology, Swiss German University
Ed_ICON, BSD City, Tanggerang 15339, Indonesia
fauad@sguniversity.edu.id euleria@mysql.com

1. INTRODUCTION

Hadoop is a popular open-source implementation of MapReduce that is used by academics, governments, and industrial organizations. Hadoop can be used for storing large data and for processing data such as data management, report generation, file analysis, web indexing, and bioinformatics 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 architectures 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 failures [1].

MySQL Cluster is a simple query algebra that lets the user decide data transformation to files or arrays 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][6]. 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 mil01k (movie len 100,000 rows) containing a total of 3,075,660 rows. The second data set is called mil1m containing a total of 1,075,660 rows. The last data set is called mil10m containing a total of 10,069,372 rows.

A. Hadoop Environment

1. Introduction

Hadoop is a popular open-source implementation of MapReduce that is used by academics, governments, and industrial organizations. Hadoop can be used for storing large data and for processing data such as data management, report generation, file analysis, web indexing, and bioinformatics 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 architectures 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 failures [1].

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 decide data transformation to files or arrays 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][6]. 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 mil01k (movie len 100,000 rows) containing a total of 3,075,660 rows. The second data set is called mil1m containing a total of 1,075,660 rows. The last data set is called mil10m containing a total of 10,069,372 rows.

A. Hadoop Environment

1. Introduction

Hadoop is a popular open-source implementation of MapReduce that is used by academics, governments, and industrial organizations. Hadoop can be used for storing large data and for processing data such as data management, report generation, file analysis, web indexing, and bioinformatics 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 architectures 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 failures [1].

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 decide data transformation to files or arrays of files. Hive is data
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


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

- http://www.apache.org/hadoop/hive