CSC443 Project

Apache Impala

Parham Taher	Sunil M. Khambaita	Huiyang Lu
taherpar	khambai1	luhui1
University of Toronto	University of Toronto	University of Toronto
Toronto, Ontario	Toronto, Ontario	Toronto, Ontario
parham.taher@mail.utoronto.ca	sunil.khambaita@mail.utoronto.ca	hy.lu@mail.utoronto.ca

INTRODUCTION

Impala is an open source SQL engine with an Apache License 2.0, written in C++ and Java. It's a product known for offering the highest performance and low latency in comparison to other SQL engines for Hadoop clusters [5].

Impala offers users a RDBMS (Relational Database Management System) like experience and provides users with the fastest way to access data stored in HDFS (Hadoop Distributed File System).

The main goal of Impala is to combine the familiar SQL support and multi-user performance of a traditional analytic database accommodating the scalability and flexibility of Apache Hadoop together with the production grade security and management extensions of the Cloudera Enterprise [5].

This project was announced in October 2012 with a public beta test distribution and became generally available the following year on May 2013. The most up to date version of the software is Apache Impala 2.11.0, which is the first release after it becoming Apache's Top Level Project. The popularity of Apache Impala has been very high, it's ecosystem momentum has been accelerating ever since its general availability with almost one million downloads.

One of the main key differences with Apache Impala compared to other databases in the market is speed. Impala is up to 13x faster than other alternatives, and 6.7x faster on average [5].

Using multi-user queries however, we have the following findings: Impala runs up to 27.4x faster than alternatives, and 18x faster on average –

or nearly three times faster on average for multi-user queries than for single-user ones [5].

The speed in Impala allows users to be able to work with HDFS or HBase using SQL queries in a much faster way compared to the other SQL engines around such as Hive.

Other similar databases to Apache Impala are Apache Hive and Apache HBase, and here are the main differences [4]:

1. Software

- HBase: HBase is wide-column store database based on Apache Hadoop. It uses the concepts of BigTable.
- Hive: Hive is a data warehouse software. Using it, we can access and manage large distributed datasets, built on Hadoop.
- Impala: Impala is a tool to manage, analyze data that is stored on Hadoop.

2. Model

- HBase: The data model of HBase is wide column store.
- Hive: Hive follows a relational model.
- Impala: Impala follows a relational model.

3. Language

- HBase: HBase is developed using Java language.
- Hive: Hive is developed using Java language.
- Impala: Impala is written in C++ and Java.

4. Schema

- HBase: The data model of HBase is schema-free.
- Hive: The data model of Hive is Schema-based.
- Impala: The data model of Impala is Schema-based.

5. API

- HBase: HBase provides Java, RESTful, and Thrift API's.
- Hive: Hive provides JDBC, ODBC, Thrift API's.
- Impala: Impala provides JDBC and ODBC API's.

6. Support

- HBase: Supports programming languages like C, C#, C++, Groovy, Java PHP, Python, and Scala.
- Hive: Supports programming languages like C++, Java, PHP, and Python.
- Impala: Impala supports all languages supporting JDBC/ODBC.

7. Triggers

- HBase: HBase provides support for triggers.
- Hive: Hive does not provide any support for triggers.
- Impala: Impala does not provide any support for triggers.

The software is mainly promoted for analysts and data scientists to perform analytics on data stored in Hadoop via SQL or business intelligence tools. This has the outcomes that large-scale data processing (via MapReduce) and interactive queries can be done on the same system using the same data and metadata – without having the need to migrate data sets into specialized systems and/or proprietary formats simply to perform analysis [14].

The main features of Apache Impala are [4]:

- Having the ability to access data with SQL like queries.
- You have much faster access for the data in HDFS compared to other SQL engines.
- Having the ability to store the data in storage system like HDFS, Apache HBase and Amazon S3.
- Apache Impala accommodates in-memory data processing which basically means that it allows users to access or analyze the data stored on Hadoop data nodes without the need of moving it.
- Apache Impala is available for free under the Apache License 2.0.
- Supports different file formats such as Avro, RCFile, Parquet, Sequence File and LZO.
- Uses few things from Apache Hive such as metadata, ODBC driver and SQL syntax.

These are some advantages of using Impala [4]:

- Impala gives use the ability to process data stored in HDFS at very fast speed with just traditional SQL knowledge.
- Whilst using Impala we have the ability to access data stored in Amazon S3, HBase, HDFS without the need of knowing how MapReduce jobs work. Basic idea of SQL queries should suffice.
- When using Impala, data processing occurs in the Hadoop cluster which means data transformation and data movement is not a requirement for data stored on Hadoop.
- Impala pioneers the usage of the Parquet file format, which is a columnar storage layout that is optimized for large-scale queries, mostly typical in data warehouse scenarios.
- In order to write queries for business tools, the data has to go through a complicated extract-transform-load (ETL) cycle, but with Impala, this procedure is shortened. The time-consuming stages of loading and reorganizing is overcome with the new techniques such as exploratory data analysis and data discovery making the process faster.

Disadvantages:

- For serialization and deserialization, Impala does not provide any support for this.
- Impala cannot read any binary files, only text files.

- When new records added to data directory in HDFS, need to refresh table.

POTENTIAL APPLICATIONS

When data scientists wish to perform analysis on data stored in Hadoop-SQL or other BI tools, Impala is a potential choice. The result is that large-scale data processing and interactive queries can be done on the same system using the same data and metadata, as mentioned previously. This removes the need to migrate data sets into specialized systems and/or proprietary formats simply to perform analysis. Therefore, Impala's potential applications include all businesses that have an interest in analyzing a huge amount of data which is stored in a computer cluster running Apache Hadoop that they collected or acquired. Aside from being used as a stand alone engine, Impala is sometimes used in conjunction with other engines like Hive to leverage the benefits of both. Many companies using Impala consider it as a premium service where Impala's powerful response rate brings them value, but tend choose other query engines such as Hive for complex queries. As [11] states, "Impala is well-suited to executing SQL queries for interactive exploratory analytics on large datasets. Hive and MapReduce are appropriate for very long running, batch-oriented tasks such as ETL". Newer companies have adopted Impala due to its performance, and often times prefer it when they don't already have an existing engine implemented [12].

According to iDatalabs [13], the top companies using Impala include:

- JPMorgan Chase
- Cloudera Inc.
- Ubisoft
- ViaSat Inc.
- MicroStrategy Incorporated

Although Impala is used primarily in the industries of Computer Software and Information Technology and Services to no surprise, it is also used in a wide range of other industries, including but not limited to:

- Staffing and Recruiting
- Financial Services
- Insurance
- Hospital & Health Care
- Management Consulting
- Banking
- Retail
- Telecommunications

SYSTEM ARCHITECTURE

Impala is a Massively-Parallel Processing (MPP) SQL query execution engine, which runs on hundreds of machines in existing Hadoop clusters. It is decoupled from the underlying storage engine,

unlike traditional relational database management systems where the query processing and the underlying storage engine are components of a single tightly-coupled system. Impala's high-level architecture is shown in Figure 1.

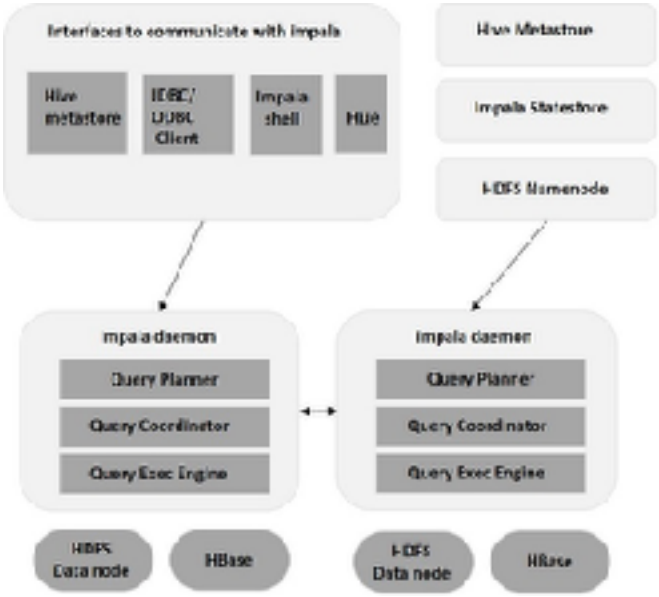

Figure 1: Architecture Overview [4]

An Impala deployment has a few main components, namely, the Impala daemon (Impalad), the Statestore daemon (statestored), and the Impala metadata and meta store.

- Impala daemon, also known as Impalad, runs on each node where Impala is installed. It accepts queries from interfaces such as Impala shell, hue browser, etc. and processes them. When a query is submitted to an impalad on a particular node, that node becomes a "coordinator node" for that query. Multiple queries are served by Impalad running on other nodes as well. Impalad reads and writes to data files and parallelizes the queries by distributing the work to the other Impala nodes in the Impala cluster after accepting queries. All Impalad instances return processed results to the central coordinating node when queries are being processed on multiple Impalad instances. Queries can be submitted to a dedicated Impalad or in a load balanced manner to another Impalad in the cluster depending on the requirement.
- The Statestore daemon (statestored) distributes cluster-wide metadata to all Impala processes. The statestore maintains a set of topics, which are arrays of (key, value, version) called entries. 'Key' and 'value' are byte arrays, and 'version' is a 64-bit integer. Topics persist through the lifetime of the statestore, but are lost when service restarts. Subscribers are processes that wish to receive updates to topics. Subscribers register with the statestore at start-up and provide it with a list of interested topics. The statestore send the subscriber an initial topic update for each registered topic, consisting of all current entries for the topic, in response to registration. The statestore periodically sends two types of messages to each of the subscribers. The first type is a topic update, consisting of all new & modified entries and deletions since the last update. Each subscriber maintains a most-recent-version for each topic, which allows the statestore to send updates between versions. After receiving a topic update, each subscriber sends a list of changes to its

subscribed topics that it wishes to make. These changes will be made before the next update. The second type of statestore message is a keepalive. Keepalive messages are used to maintain connections between the statestore and each subscriber. Without a keepalive message, a subscriber will time-out its subscription and attempt to re-register with the statestore. When the statestore detects a failed subscriber (e.g. by repeated failed keepalive deliveries), it will stop sending updates. Some topics may be marked 'transient', which means that if their 'owning' subscriber failed, the topics will be removed.

- Impala uses traditional MySQL or PostgreSQL databases to store table definitions. Important details such as column information and table definitions are stored in a centralized database called a meta store. Each Impala node caches locally all the metadata. Getting table specific metadata could take a large amount of time when dealing with an extreme amount of data and partitions. Therefore, it is helpful to have a locally stored metadata cache in order to provide information instantly. When table definition or table data is updated, all Impalads must update their metadata cache by retrieving latest metadata, before they can issue new queries on the table.

QUERY LANGUAGE / INTERFACE

According to [5]*:*

- "Impala supports most of the SQL-92 SELECT statement syntax, with additional SQL-2003 analytic functions, and most of the standard scalar data types:
  - integer and floating point types
  - STRING
  - CHAR
  - VARCHAR
  - TIMESTAMP,
  - DECIMAL with up to 38 digits of precision.
- Custom application logic can be incorporated through user-defined functions (UDFs) in Java and C++, and user-defined aggregate functions (UDAs), which is available in C++.
- Due to the limitations of HDFS as a storage manager, Impala does not support UPDATE or DELETE, and basically only supports bulk insertions such as INSERT INTO and SELECT.
- Unlike in a traditional RDBMS, using HDFS's API the user can add data to a table simply by copying/moving data files into the directory location of that table. Alternatively, the same can be accomplished with the LOAD DATA statement. Like bulk insert, Impala supports bulk data deletion by dropping a table partition (ALTER TABLE DROP PARTITION). Because it is not possible to update HDFS files in-place, Impala does not support an UPDATE statement. Instead, the user typically recomputes parts of the data set to incorporate updates, and then replaces

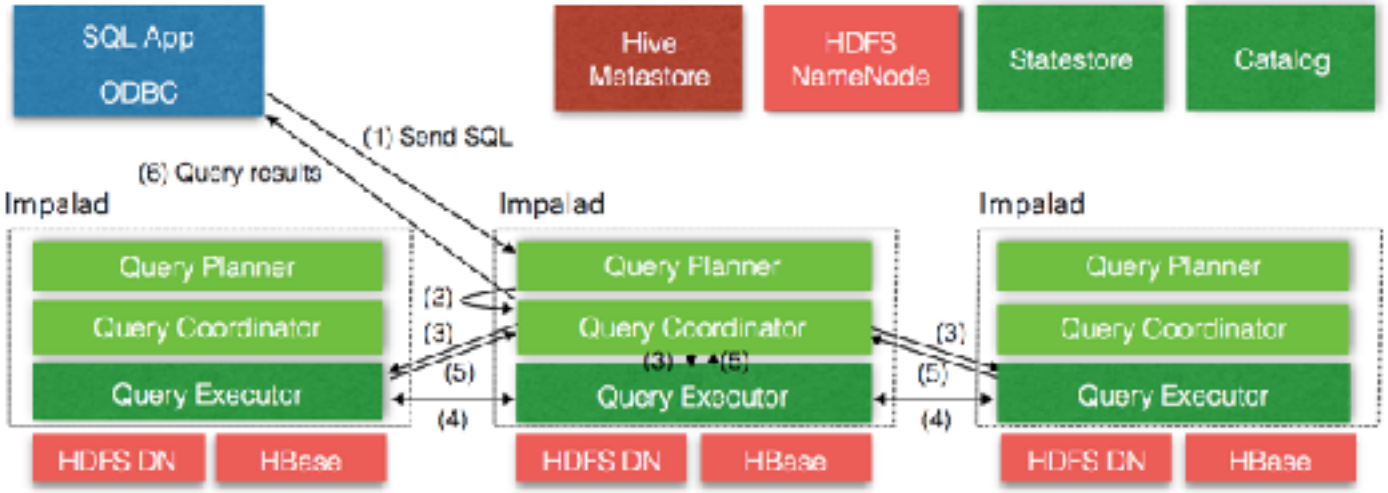

Figure 2: Flow during query processing [5]

the corresponding data files, often by dropping and re-adding the partition.

- After the initial data load, or whenever a significant fraction of the table's data changes, the user should run the COMPUTE STATS statement, which instructs Impala to gather statistics on the table. Those statistics will subsequently be used during query optimization."

As also described in the Cloudera Documentation [1], Impala uses a Query language that is similar to SQL and HiveQL. Following are some of the key characteristics of the Impala Query language:

- You cannot update or delete individual records in Impala.
- Impala does not support transactions.
- Impala does not support indexing.
- Impala stores and manages large amounts of data (petabytes).

As for existing interfaces, Impala provides these three [4]:

- Impala-shell - A command prompt based shell (terminal).
- Hue interface - Impala queries can be processed using the Hue browser. In the Hue browser, there is a Impala query editor where the user can type and execute impala queries.
- ODBC/JDBC drivers - Just like other databases, Impala provides ODBC/JDBC drivers. Using these drivers, impala can be connected to through programming languages that supports these drivers and build applications that process queries in impala using those programming languages.

Whenever a query is passed using any of the above interfaces, one of the Impalads in the cluster will accept the query. This Impalad is treated as a coordinator for that particular query.

After receiving the query, the query coordinator verifies whether the query is valid, using the Table Schema from the Hive meta store. Then, it collects the information about the data's location from the HDFS name node and relays this

information to other impalads in order to execute the query.

All the other Impalads read the specified data block and processes the query. As soon as all the daemons complete their tasks, the query coordinator collects the results and delivers them to the user [4]. See figure 2 for a diagram of Impala's query processing.

STORAGE DETAILS

Impala's backend, written in C++ is designed to take advantage of modern hardware, and in addition to having code generation at runtime, it is able to run in an optimal manner without much memory overhead when compared to java-based engines. Although Impala is memory bound for execution, its engine is designed so that when extra memory is required, operators which include hash join, hash-based aggregation, sorting, and analytic function evaluation are able to swap to the disk. It is a MPP SQL query engine, which is an interface that sits on top of HDFS (Hadoop Distributed File System), designed to handle big data. Impala uses an HDFS feature known as short-circuit local reads, bypassing the DataNode protocol when reading from local disk to efficiently retrieve data with increased throughput from HDFS, which has been a huge challenge for many SQL systems designed for Hadoop [5].

Storage Formats:

Impala has a wide variety of support for different file storage formats such as plain text, Avro, Sequence, etc, which can all be used with compression algorithms for greater efficiency. The most recommended file format which takes full advantage of Impalas features is Apache Parquet. Parquet is an open source columnar file format developed mainly by Twitter and Cloudera. It has a high compression potential, high scan efficiency, and can be processed by most Hadoop based processing frameworks such as Hive and MapReduce. In essence, Parquet is designed to deal with large data blocks ranging from ten to thousands of megabytes. "Parquet stores nested fields column-wise and augments them with minimal information to enable re-assembly of the nesting structure from column data at scan time" [5]. Whether the database is stored in plain text, Sequence, RC, and Parquet formats, "Parquet consistently outperforms all of them by up to 5 times" [5].

HDFS:

To discuss the central details of Impala's file structure, it is best to describe HDFS, since it is the underlying and most optimal storage solution for Impala. It is designed from start to handle huge amounts of data [8]. HDFS is Highly fault tolerant and "Impala relies on the redundancy provided by it to guard against hardware or network outages on individual nodes" [7]. HDFS Runs on commodity hardware, is designed to work on inexpensive hardware, provides high throughput of data when dealing with large datasets, and is very portable from

platform to platform. HDFS has a master-slave architecture. It is organized into clusters, and each cluster consists of a single NameNode and multiple DataNodes. GNU/Linux operating systems are generally used to implement this architecture [8]. The NameNode is the coordinator which determines the mapping of blocks to DataNodes. It is a master server used to manage the file system namespace and client access to files, and contains multiple DataNodes as shown in figure 3. A file is split into one or more blocks and these blocks are stored in a set of these DataNodes. The number of replicas of a file can be specified (known as the replication factor), which is essential for fault tolerance. Optimizing the replication factor is important and is done by the NameNode being rack aware, and replicating to the correct machines [8].

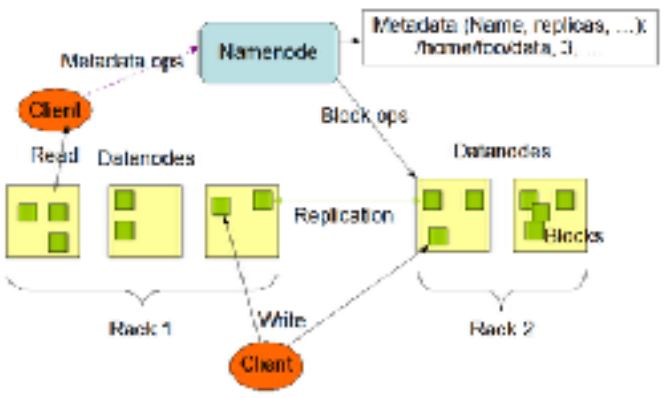

Figure 3: HDFS Architecture [8]

Files are stored as a sequence of blocks of the same size, except the last block. The block size is configurable which usually ranges from 64MB to 256 MB per block. “HDFS supports write-once-read-many semantics on files. A typical block size used by HDFS is 64 MB. Thus, an HDFS file is chopped up into 64 MB chunks, and if possible, each chunk will reside on a different DataNode“ [8]. “The NameNode contains an image of the entire file system namespace and file Blockmap in its memory. This is piece of metadata is compact so that a NameNode with a minimum of 4GB RAM can handle large amounts of files [8]. As mentioned, HDFS is designed to support very large files and applications which require storage of large files. In this case, typically writes of data are done once, but reads are done multiple times. HDFS caching can also be used to get increased performance, by making effective use of RAM. “The HDFS caching feature lets you designate a subset of frequently accessed data to be pinned permanently in memory, remaining in the cache across multiple queries and never being evicted“ [9].

In terms of file size, *Cloudera Product Documentation: Impala Performance Guidelines and Best Practices*[10] mentions that, “You want to find a sweet spot between ‘many tiny files’ and a ‘single giant file’ that balances bulk I/O and parallel processing. Also, you can “set the PARQUET_FILE_SIZE query option before doing an INSERT ... SELECT statement to reduce the size of each generated Parquet file” [10]. Impala maintains information regarding table definitions and the physical locations of blocks in the metastore within the HDFS. Each Impala node caches all of this metadata to reuse for future queries against the same table.

Impala also has support for HBase as a storage medium. When integrating with HBase, the Cloudera documentation mentions that: “HBase is an alternative to HDFS as a storage medium for Impala data. It is a database storage system built on top of HDFS, without built-in SQL support. Many Hadoop users already have it configured and store large (often sparse) data sets in it. By defining tables in Impala and mapping them to equivalent tables in HBase, you can query the contents of the HBase tables through Impala, and even perform join queries including both Impala and HBase tables” [7].

EVALUATION

We created a test database using the Cloudera QuickStart VM which has the Hadoop Stack preinstalled. We ran some test commands and noted the results.

For example, some commands we tested include:

```
CREATE DATABASE IF NOT EXISTS my_db;

CREATE TABLE IF NOT EXISTS my_db.employees (ID
INT, name STRING, age INT, address STRING,
salary BIGINT );

INSERT INTO employees (ID, NAME, AGE, ADDRESS,
SALARY) VALUES (1, ‘Parham’, 22, ‘123 DB
Street’, 5000);

SELECT * FROM employees
```

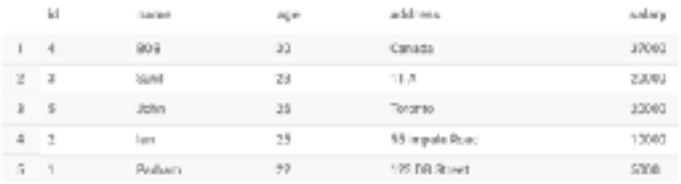

Figure 4: Sample Results

Query Runtime Profile:

```
Query (id=164c190e2f4abf64:bf736c1f00000000):
  Summary:
    Session ID:
174b118067e36747:ac921a5efd0ceba4
    Session Type: BEESWAX
    Start Time: 2018-03-25 15:03:55.820665000
    End Time: 2018-03-25 15:03:55.987249000
    Query Type: QUERY
    Query State: FINISHED
    Query Status: OK
    Impala Version: impalad version 2.10.0-
cdh5.13.0 RELEASE (build
2511805f1eaa991df1460276c7e9f19d819cd4e4)
    User: cloudera
    Connected User: cloudera
    Delegated User:
    Network Address: 10.0.2.15:57213
    Default Db: my_database
    Sql Statement: select * FROM employees
    Coordinator: quickstart.cloudera:22000
    Query Options (set by configuration):
    Query Options (set by configuration and
planner): MT_DOP=0
```

According to [3], benchmarking Impala queries using a single node cluster with a small dataset (tens of gigabytes) will not return a realistic view of the engine, since there will not be enough data to take advantage of Impala's parallel distributed queries. As such, it is infeasible to evaluate Impala locally.

After further investigation, we found that a great number of benchmarkings on Impala are done

on the cloud, through services such as Amazon Web Services (AWS). However, we ran into several difficulties attempting to set up a multi-node cluster environment with a sufficiently large dataset on Amazon Elastic Compute Cloud (Amazon EC2).

Therefore, we will instead present a few of the methods and findings from an excellent benchmark *Big Data Benchmark [6],* and analyze them.

In the database, there are three datasets with the following schemas:

```
Documents
Unstructured HTML documents

Rankings
Lists websites and their page rank
pageURL VARCHAR(300)
pageRank INT
avgDuration INT

UserVisits
Stores server logs for each web page
sourceIP VARCHAR(116)
destURL VARCHAR(100)
visitDate DATE
adRevenue FLOAT
userAgent VARCHAR(256)
countryCode CHAR(3)
languageCode CHAR(6)
searchWord VARCHAR(32)
duration INT
```

Then, they launched EC2 clusters and ran queries in a few different frameworks including Impala. Below are the queries ran and results for Impala with tables on disk, Impala with tables on OS buffer cache, and Hive. The analysis is done without using parquet and measures response time.

1. Scan Query

```
SELECT pageURL, pageRank FROM rankings WHERE pageRank > X
```

Results:

	Median Response Time		
	Small Result	Medium Result	Large Result
Impala (Disk)	12.015	12.015	37.085
Impala (Mem)	2.17	3.01	36.04
Hive (YARN)	50.49	59.93	43.34

Analysis:

- Tests throughput of read and write table data.
- Impala(mem) performs well by avoiding disk.
- Impala outperforms Hive partly due to more efficient task launching and scheduling.
- As the result gets larger, Impala bottlenecked due to poor ability to persist the result to disk.

2. Aggregation Query

```
SELECT SUBSTR(sourceIP, 1, X), SUM(adRevenue) FROM universities GROUP BY SUBSTR(sourceIP, 1, X)
```

Results:

	Median Response Time		
	Small Result	Medium Result	Large Result
Impala (Disk)	113.72	155.31	277.53

Impala (Mem)	84.35	134.82	261.015
Hive (YARN)	730.62	764.95	833.3

Analysis:

- Impala reads from OS buffer cache, therefore it has to read and decompress entire rows.
- For large results, Impala performs quite slow due to the speed of materializing output tables.

3. Join Query

```
SELECT sourceIP, totalRevenue, avgPageRank
FROM
  (SELECT sourceIP,
          AVG(pageRank) as avgPageRank,
          SUM(adRevenue) as totalRevenue
    FROM Rankings AS R, UserVisits AS UV
    WHERE R.pageURL = UV.destURL
       AND UV.visitDate BETWEEN
Date(`1980-01-01') AND Date(`X')
    GROUP BY UV.sourceIP)
  ORDER BY totalRevenue DESC LIMIT 1
```

	Median Response Time		
	Small Result	Medium Result	Large Result
Impala (Disk)	108.68	129.815	431.26
Impala (Mem)	41.21	76.005	386.6
Hive (YARN)	561.14	717.56	2374.17

Analysis:

- Small join most time spent scanning and doing date comparisons.
- Impala on Disk and Impala on Mem very similar on large queries.
- Hive requires all the data to join, which makes it slow on large queries.
- Hashing join keys and network I/O is a bottle neck for all three.

As for query optimizations, Impala does not support indexing, we will discuss its ways of query optimization below:

Query planning/optimization in Impala happens in two phases: single node planning and parallelization and fragmentation planning. The query planner is given a parse tree with query-global information.

- In the first phase, the parse tree is translated into a non-executable single-node plan tree, consisting of HDFS/HBase scan, hash join, cross join, union, hash aggregation, sort, top-n and analytic evaluation.
- The second phase takes the single-node plan as input and produces a distributed execution plan. Since in HDFS, remote reads are slower than local reads, the general goal of this step is to minimize data movement and maximize scan locality [5].

See figure 5 for a diagram of Impala's two phase query optimization.

Impala can also optimize better for complex/ multi-table queries when statistics are available, to understand the volume of data and distribution of values better. This information will help parallelize and distribute the work for a query.

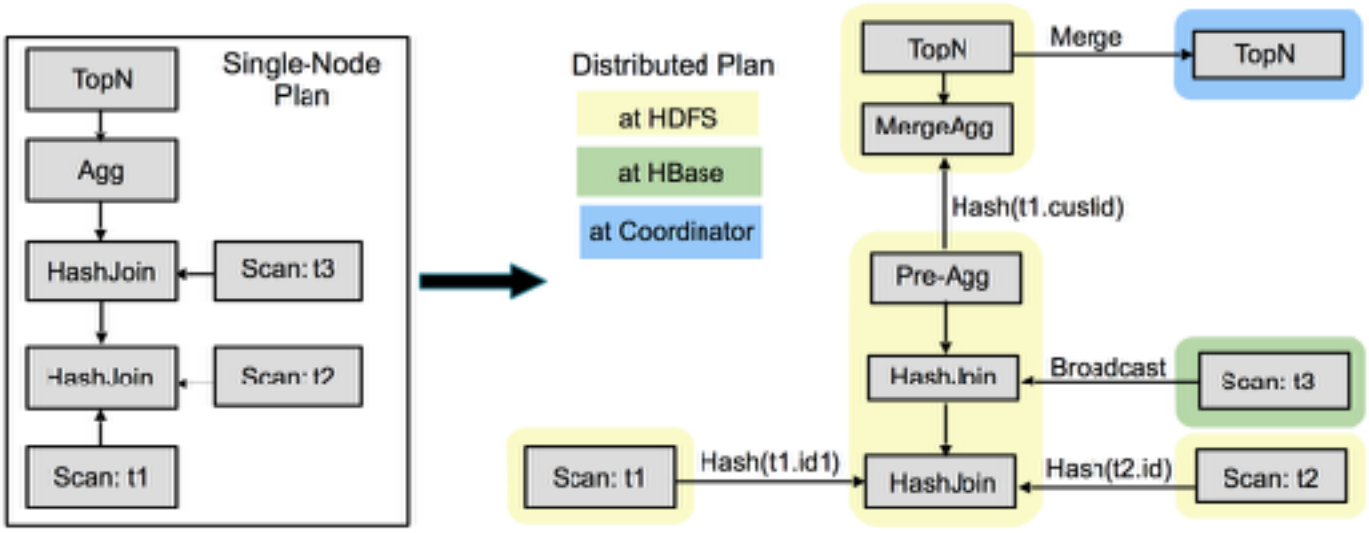

Figure 5: Two phase query optimization.[5]

The categories of statistics that can be used by Impala include:

- Table Statistics: The Impala query planner can use statistics about entire tables and partitions.
- Column Statistics: The Impala query planner can use statistics about individual columns. This metadata is valuable for columns compared across tables in JOIN queries, to help estimate how many rows will be retrieved from each table [2].

REFERENCES

[1] *Cloudera Product Documentation: Cloudera Impala User Guide*. http://www.cloudera.com/documentation/cdh/5-1-x/Impala/impala.html

[2] *Cloudera Product Documentation: How Impala Uses Statistics for Query Optimization.* http://www.cloudera.com/documentation/cdh/5-1-x/Impala/Installing-and-Using-Impala/ciiu_perf_stats.html

[3] *Cloudera Product Documentation: Benchmarking Impala Queries.* http://www.cloudera.com/documentation/cdh/5-1-x/Impala/Installing-and-Using-Impala/ciiu_perf_benchmarking.html

[4] *Learning Impala.* https://www.tutorialspoint.com/impala/index.htm

[5] *Impala: A Modern, Open-Source SQL Engine for Hadoop.* http://cidrdb.org/cidr2015/Papers/CIDR15_Paper28.pdf

[6] *Big Data Benchmark.* https://amplab.cs.berkeley.edu/benchmark/.

[7] *Cloudera Product Documentation: Concepts and Architecture*. https://www.cloudera.com/documentation/enterprise/5-2-x/topics/impala_hadoop.html.

[8] *HDFS Architecture Guide*. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[9] *Cloudera Product Documentation: Using HDFS Caching with Impala*. https://www.cloudera.com/documentation/enterprise/5-2-x/topics/impala_perf_hdfs_caching.html

[10] *Cloudera Product Documentation: Impala Performance Guidelines and Best Practices*. https://www.cloudera.com/documentation/enterprise/5-9-x/topics/impala_perf_cookbook.html

[11] I*mpala or Hive - When to Use What?* http://www.thecloudavenue.com/2013/11/HiveVsImpala.html

[12] *Impala vs Hive: Difference between Sql on Hadoop components*. https://www.dezyre.com/article/impala-vs-hive-difference-between-sql-on-hadoop-components/180

[13] *Companies using Cloudera Impala*. https://idatalabs.com/tech/products/cloudera-impala

[14] *Apache Cloudera* https://en.wikipedia.org/wiki/Apache_Impala