# IE494 BIG DATA PROCESSING
## Study Report: T25

## Prof. P.M. Jat

Group Members:
- Shravan Kakadiya - 202201333

### A Deep Dive into "Tenzing: A SQL Implementation on the MapReduce Framework"

Introduction:

The **MapReduce framework** has conquered enough popularity for both things within Google and outside of it. It is one of the most preferred frameworks for using the distributed data systems. But **Apache Hadoop** has gained enormous popularity for its open source implementations.

As if one Framework has its Pros so it does have its cons also. So here for using **MR Architecture** one need to know distributed data working or how it is processed and some of the computer languages like **C, C#, JS** etc. So to conquer this issue some of the interfaces have been made like **PIG, HIVE**, etc. But mostly some of them have high error rates meaning efficiency and performance and are less SQL compatible.

Also MR cannot have the ability to have the **traditional database efficiency and optimization techniques**.

To conquer these changes, Tenzing was developed on MapReduce. It has:

- **Low latency**.
- **High efficiency** for distributed queries.
- Have both implementations of **SQL92** and **SQL99.**

Tenzing not only has the ability to query data using **row, column stores** but also has **Bigtable, GFS,** etc. Also supports User defined SQL features like multiple relational data handling.

This system uses traditional database techniques to increase its performance like indexing. Have the ability to handle **millions and billions of data** using MR distributed framework.

My Understanding of the Study That I Have Carried Out in each sections:

**A) IMPLEMENTATION OVERVIEW:**

1. **Four Major Components**
    - This system contains a worker **pool**, **query server**, **client interfaces**, and **metadata server.**
    - These components help ensure that efficient query and data handling is happening properly.

---

2. **Worker Pool**
    - It Uses a MR Job model to reduce the query latency.
    - Comprises of:
        - **Master nodes** and **worker nodes** to enable query execution.
        - A **master watcher** that takes care of the entire worker pool.
    - Have the ability to handling data from **ColumnIO**, **Bigtable**, **GFS**, and **MySQL.**

---

3. **Query Server**
    - It is the bridge b/w clients and the worker pool.
    - Responsibilities:
        - Does SQL query parsing.
        - Does SQL query execution using rule and cost based strategies.

---

4. **Client Interfaces**
    - Helps to have include multiple interfaces for need of different users:
        - **Command-Line Interface (CLI)**: High script writing ability for power users.

- **Web-based UI**: Have User-Friendly UI for users
- **API**: Allow direct usage of the worker pool through this.

---

5. **Metadata Server**
    - Task is to continuously manage the table, schema definitions.
    - Also used in storing metadata of tables and underlying data.
    - Takes help of Bigtable to provide reliability and persistence in storage requirements.

**B) Life Of A Query:**

1. **Query Submission**
    - Process starts when the user or any other process submits a query using Tenzing's Interfaces like CLI or API or UI.

---

2. **Parsing the Query**
    - Query server uses the **intermediate parse tree** to parse the submitted query for further usage.
    - This step ensures that for usage of the query optimization and execution.

---

3. **Metadata Retrieval**
    - Query server then fetches the data from the metadata with the help of metadata server like table schemas and their locations to enrich the parse tree into a more complete intermediate format
    - This step highlights Tenzing's reliance on an organized metadata layer for informed execution planning.

---

4. **Query Optimization**
    - The intermediate format uses rule-based and cost-based optimization techniques for optimizing the query using query server.
    - This ensures that the resulting execution plan is efficient and suitable for the system's distributed architecture.

5. **Execution Plan Partitioning**
    - The optimized execution plan is divided into one or more **MapReduce jobs**.
    - The query server, with the help of the **master watcher**, identifies an available master node to execute each job.
    - At this stage, the query is physically partitioned into smaller, manageable units of work called **shards**.

---

6. **Work Allocation and Execution**
    - Polling happens for master nodes by the idle workers for the available work.
    - After processing the tasks, reduce workers do aggregation by storing the intermediate results.

---

7. **Result Gathering and Streaming**
    - Monitoring of the intermediate storage is happening for the output results once they are produced by the query server.
    - The query server merges the results from all worker nodes also ensuring low latency for sending outputs.

**C) SQL Features:**

**1) SQL Compatibility and Enhancements**

- Tenzing supports SQL92 having some standard operations like projections and aggregation and filtering.
- For large scale data it does use **parallelizable SQL enhancements** for analysis of the data on MR framework.

---

2) **Projection and Filtering Optimizations**

- Both SQL operations and Sawzall operations and functions are supported giving the user both standard means built-in and UDF.
- The compiler optimizes query performance using strategies like:
    - Constants conversion during compilation time.
    - Usage of indexed databases like Bigtable
    - Skipping unnecessary partitions in range-partitioned data.
    - Scanning only relevant columns and leveraging file headers to bypass irrelevant data in formats like ColumnIO.

---

3) **Aggregation Mechanisms**

- Tenzing not only supports standard aggregate functions but also some complex functions like (e.g., SUM, COUNT DISTINCT, STDDEV, CORR)
- Aggregation happens using hash-based aggregation so that unnecessary sorting is discarded.

---

4) **Hash-Based Aggregation Workflow**

- Hash-based aggregation relies on hash tables in both the mapper and reducer phases:
    - Mappers build hash tables for grouping and incrementing counts without sorting.
    - Reducers aggregate hash table data directly, enhancing query performance by avoiding the overhead of sorting.
- While this method boosts efficiency, it requires explicit user selection and sufficient memory for hash table storage.

---

5) **Join Capabilities and Support**

- Tenzing supports most of the Joins like **inner, left, right, full outer, cross, equi, non-equi**, **semi-equi,** etc.
- Right joins do take help of sorting and merging techniques and cross joins are only used for small tables.

---

6) **Types of Join Implementations**

- **Broadcast Joins**:
    - Only used when the secondary table is smaller enough to get fitted in memory.
    - Supports Optimizations like filtering join data while loading, joins also supports various types like cross join, equi, semi join etc.
    - Broadcast joins support various conditions, such as cross, equi, semi-equi, and non-equi joins.
- **Remote Lookup Joins**:
    - Most efficient for indexed data sources.
    - Have good performance by using local LRU caches.
- **Distributed Sort-Merge Joins**:
    - Best suited for tables of similar size without indexed join keys.
- **Distributed Hash Joins**:
    - Used when we have table size too huge to fit in memory and also indexing is lacking.
    - Hash joins avoid sorting and employ efficient MapReduce operations for partitioning and aggregation.

---

7) **Optimizations for Joins**

- It uses **sort avoidance**, **memory chaining**, and **block shuffling** so that distributed Hash joins have more good performance.
- It also has some scheduler that chains and parallelizes operations intelligently.

---

8) **Hash Join Workflow Example**

- A typical hash join involves:
    - Firstly partitioning of tables is done by the use of the join key using MR Jobs and then lookups happen.
    - And then Aggregating results in an incremental way without sorting.

---

9) **Analytic Functions**

- For advanced data analysis Tenzing supports some of the analytic functions like **RANK, SUM, MIN, MAX,etc.**

- All these functions are taking help of MR framework based backend where mapper o/p are sorted and merged and then passed to reducer for further processing.

---

**10) OLAP Extensions**

- This Tenzing supports **ROLLUP() and CUBE()** extensions for M-arry dimensional aggregation.
- Mappers o/p have key-value pairs to perform aggregation and then reducers do sorting aggregations.
- This approach efficiently computes hierarchical totals and subtotals.

---

**11) Set Operations**

- Supports multiple SQL set operations like (**UNION, UNION ALL, MINUS, MINUS ALL**).
- Mostly set operations are executed in the reducer phase only.
- Round-robin partitioning used for UNION ALL to ensure good efficiency.

---

**12) Nested Queries and Subqueries**

- Here each nested query is translated into a separate MapReduce job.
- Performance improvement occurs by eliminating unnecessary jobs for simple queries.
- Efficiency is ensured by combining reducers and mappers in the same process.

---

**13) Views and Security**

- Logical views are one of the main features of the Tenzing SQL. It allows both row and column oriented securities for views.
- This enhances data security while maintaining usability for authorized users.

---

**14) DML and DDL Operations**

- **INSERT, UPDATE, DELETE** operations are supported in Tenzing but still ACID properties are not supported.
- Standard **DDL operations** are supported like CREATE, DROP, ALTER, etc.

---

**D) Performance:**

**1) MapReduce Enhancements**

- **Worker Pool Design:**
  - Worker pool architecture helps in reducing query latency as it avoids overhead of new processes for every query.
  - It does task assignment, query coordination and data processing and this is done using **master watcher, master pool and worker pool.**
  - This decreased the execution time latency by approx 7 seconds.
- **Block Shuffle Mechanism:**
  - Here we will take use of **block-based shuffling** instead of row-based which does compression in blocks for transmitting data.
  - When this method is used we get 3 times faster data shuffling.

---

**2) Local Execution for Small Data Sets**

- Datasets smaller than approx a threshold, Tenzing do local execution of the query instead of taking it to the worker pool.
- This helps in reducing the latency by approximately 2 seconds.

---

**3) Scalability and Throughput**

- As Tenzing uses MR framework so it gets exceptional performance and scalability.
- When tested with 200 Billion data across 2 centers so we get steady performance metrics even without using the worker nodes.
- We can extend resources to 6GB RAM and 24GB ROM per core in it.

---

**4) Experimental LLVM Query Engine**

- Using iterative analysis, Tenzing increased its execution method to max performance per-worker node:
    - We got good performance by using direct parse-tree using **Dremel's SQL engine** but was limited by the interpreter-like processing.
- **LLVM-Based Native Code Generation**
    - By using Native code for SQL, throughput was increased by 10x.
    - As intermediate results are stored by rows so we do easy handling of hash-based operations.
- **Column-Major Vector Processing**
    - For low-selectivity queries we can use Columnar representation as it has efficient in-memory processing.
    - Less used or less performance where row based entering data or querying data are needed.

---

Conclusions drawn from this Research paper:

• From this research paper we come to the conclusion that we can make a good performance SQL engine on top of the MR framework.

• Also we can implement a huge number of performance query optimizations in our databases.

• We choose this MapReduce Framework as it provides us with a combination of high performance, high reliability and high scalability that helps us to solve complex problems easily.

• As we have heterogeneous data sources, it is possible to create a smart system which takes full advantage of the data sources using the engine and the optimizer.

Ideas for future study/work based on study that I have done:

1) **Improved Query Optimization Strategies**
   We can use advanced ML-based query optimization strategies to further improve query latency and performance. These ML models help to see the query patterns and then accordingly select the best optimal Algos to process further.

2) **Distributed Query Execution Enhancements**
   We can find new partitioning and workload allocation strategies for MR Jobs so that data transferring decreases and proper cache utilization occurs. Also we can work on in-memory processing for iterative queries.

3) **Data Security and Compliance Features**
   We can support fine-grained access controls along with encrypted query execution to improve security of Tenzing's architecture. Mostly these features are used where safety and security are prime requirements such as the finance sector. By this way Tenzing's security is increased.

4) **Scalability Testing on Emerging Hardware**
   We can test Tenzing's **Scalability** on huge hardware platforms such as GPUs. This helps us to find the performance metrics on integrating emerging hardware into the worker pool.

5) **Integration with Modern Big Data Ecosystems**
   We can include big data ecosystems, along with cloud-native storage solutions and generative AI workflows. This will help Tenzing to serve it as a full performance occupied SQL architecture for processing the data that can be used in Artificial Intelligence or Machine Learning pipelines.

References:


- Chattopadhyay, Biswapesh, et al. "Tenzing a sql implementation on the MapReduce framework." Proceedings of the VLDB Endowment 4.12 (2011): 1318-1327. https://www.vldb.org/pvldb/vol4/p1318-chattopadhyay.pdf
- https://www.geeksforgeeks.org/mapreduce-architecture/
- F.N. Afrati and J.D. Ullman. Optimizing joins in a Map-Reduce environment. In Proceedings of the 13th International Conference on Extending Database Technology, pages 99–110. ACM, 2010 https://dl.acm.org/doi/pdf/10.1145/1739041.1739056
- D. Cutting et al. Apache Hadoop Project. http://hadoop.apache.org/
- A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Raisin. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. Proceedings of the VLDB Endowment, 2:922–933, August 2009. https://dl.acm.org/doi/pdf/10.14778/1687627.1687731