TECHNIQUES FOR ANALYTICAL QUERY PROCESSING & OPTIMIZATIONS IN MAPREDUCE

PRESENTED BY

FAIZAN AHMED
KANIKA NAGPAL
RIPUL JAIN
Our aim is to review the most challenging aspects of MapReduce framework and investigate in depth the technologies and techniques available in order to address these challenges.

The methodology that we have employed to reach this goal is by reviewing the state of the art in improving the performance of parallel query processing using MapReduce.

A set of the most significant weaknesses and limitations of MapReduce is discussed at a high level, along with solving techniques and a classification of existing research by focusing on the optimization objectives.
WHAT IS MAPREDUCE?

- Programming paradigm for large-scale processing and analysis of massive data sets on clusters of machines.
- The MapReduce framework enables the developer to focus on data wrangling by abstracting away the cumbersome tasks of distributed programming such as machine to machine communication, task scheduling, fault-tolerance, partitioning etc.
- Because of its simplicity, MapReduce has become the most popular framework for analytical query processing. Various implementations include Pig, SparkSQL etc.
THE MAPREDUCE ARCHITECTURE
BUT.. .WHAT IS WRONG WITH MR?

- **Selective access to data**: Currently, the input data to a job is consumed in a brute-force manner, in which the entire input is scanned in order to perform the map-side processing. In MapReduce, the framework will initiate map tasks on all input partitions. However, for certain types of analytical queries, it would suffice to access only a subset of the input data to produce the result.

- **High communication cost**: The size of the output of the map phase can be significant and its transmission may delay the overall execution time of the job. E.g.: Join query

- **Redundant and wasteful processing**: If two or more jobs need to perform the same processing over same data then MapReduce would perform such jobs independently.
Recomputation: MapReduce lacks the mechanism for management and future reuse of output results. So there is no efficient way for future queries to reuse the results from previous queries.

Lack of early termination: Mappers read the entire data before any of the reducers can start processing. This leads to inefficiencies when only a subset of data would suffice to execute the query. For example, typically in data analytics over massive datasets, one would execute top-k selection query which cannot be done productively with MapReduce.

Lack of iteration: For simple iterative processing or recursive queries, a sequence of MapReduce jobs need to be written and their coordination handled. This leads to a significant drop in the performance as MapReduce does not provide any technique for storing intermediate results in memory.
Quick retrieval of approximate results: MapReduce does not provide an explicit way to support quick retrieval of indicative results by processing on representative input samples of massive datasets.

Load balancing: Data distribution and partitioning in a fair manner of a complex processing task to all the available processing nodes is a challenge. Providing advanced load balancing mechanisms that aim to increase the efficiency of query processing by assigning equal shares of useful work to the available resources is a weakness of MapReduce.

Lack of real time processing: The design of MapReduce is not suitable for interactive analytical query processing as the fault-tolerance and batch processing model hinders the achievement of fast processing times. Furthermore limited exploitation of main memory, shuffling big amounts of data in the network and delays for job initiation and scheduling also add to the overall overhead of the processing task.
### Query Optimization Techniques

#### Processing Optimizations
- Using Situation-Aware Mappers (SAMs) to allow mappers to communicate in order to make globally coordinated optimization decisions.
- Moving to columnar storage model and reusing hash-tables stored in memory.

#### Configuration Parameter Tuning
- Cost-based optimizers which determine appropriate values for the configuration parameters of a MR job e.g. number of map & reduce tasks, amount of allocated memory etc.

#### Plan Refinement & Operator Reordering
- Utilizing correlation-aware SQL-to-MapReduce translators
- Collect statistics from running jobs and use these for re-optimizing future invocations of the same or similar job by inserting in a cost-based optimizer.

#### Code Analysis & Transformation
- Using frameworks like HadoopToSQL to be able to utilize SQL's features for indexing, aggregation and grouping from MR.
- Data-centric code analysis can lead to potential query optimizations by using traditional RDBMS query planning techniques.
Interactive & Real Time Processing

**Streaming & Pipelining**
- *Dremel* is an interactive data analysis engine which combines a nested data model with columnar storage to enhance retrieval efficiency. It provides real-time query processing by using a multi-level tree architecture.
- Another strategy is to keep workers running constantly in a worker pool instead of spawning new processes thus reducing query latency.

**In-memory processing**
- Spark Streaming uses a lazy dataflow model and relies on using a main memory abstraction called Resilient Distributed Datasets. Using this abstraction performance improvement is achieved via inter-query caching of data in memory.
- Hash-based aggregation and dynamic partial DAG execution are also other features which further lead to faster query processing.

**Pre-computation**
- Approximate query results in a streaming model can be achieved by pre-computing samples. The accuracy of approximate results depends on the amount of recurring query templates i.e. set of columns used in WHERE & GROUP BY clauses.
## Early Termination Techniques

<table>
<thead>
<tr>
<th>Dynamic data access</th>
<th>Pipelining</th>
<th>Statistics and Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A new type of job can be introduced that dynamically controls its data access</td>
<td>• Mappers can process random samples of data and reducers can process it immediately by using pipelining</td>
<td>• We can compute statistics during record scans and use these to compute a threshold for early termination</td>
</tr>
<tr>
<td>• An InputProvider can be added together with Mapper and Reducer that makes data access decisions by providing enumerations for different data availability states</td>
<td>• The reduce phase can be modified to support incremental processing of data</td>
<td>• Data can be placed intelligently using advanced partitioning schemes that are tailored to expected query workload</td>
</tr>
<tr>
<td></td>
<td>• Accuracy estimation can be used as terminating condition</td>
<td></td>
</tr>
</tbody>
</table>

- Pipelining allows for the processing of random samples of data and immediate processing by reducers.
- The reduce phase can be modified to support incremental processing of data.
- Accuracy estimation can be utilized as a terminating condition.
- Statistics and partitioning schemes can be used to compute thresholds for early termination and place data intelligently.
## Iterative Processing Techniques

<table>
<thead>
<tr>
<th>Loop-aware Processing, Caching, Pipelining</th>
<th>Recursive Queries &amp; Incremental Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A programming interface that provides a new loop control for expressing iterations</td>
<td>• Recursive tasks usually need to produce some output before completion, so that it can be used as feedback to the input. Hence, the blocking property of MapReduce violates the requirements of recursion</td>
</tr>
<tr>
<td>• The scheduler can be modified to ensure that same tasks are assigned to same nodes for multiple iterations</td>
<td>• It maybe possible to compute and propagate just the deltas describing the changes between iterations</td>
</tr>
<tr>
<td>• Results from the reducers can be cached for subsequent iterations</td>
<td></td>
</tr>
</tbody>
</table>
## Data Access

### INDEXING
- Can be done by means of User Defined Functions (UDFs) i.e without modifying the Hadoop framework at all. The indexing information is injected into logical input splits and will serve as a cover index for the data inside the split. Also, the index is created at load time, thus no overhead in query processing.
- Exploiting the n replicas (typically n = 3) maintained in Hadoop by default for fault-tolerance and by building a different clustered index for each replica. This improves the performance of MR processing.

### DATA LAYOUTS
- The use of a columnar file for data storage. The data is partitioned in vertical groups, each group is sorted based on a selected column and stored in column-wise format in HDFS. This gives selective access to only the columns used in the query.
- Combine horizontal with vertical partitioning to avoid wasted processing in terms of decompression of unnecessary data. First, data is partitioned horizontally, and then, each horizontal partition is partitioned vertically.
Avoiding Redundant Processing

1. Transform sets of submitted jobs into groups and treat each group as a single job, by solving an optimization problem with objective to maximize the total savings. Problem: does not support jobs that use multiple inputs.

2. ReStore: Stores and reuses the intermediate results produced by workflows of MapReduce jobs.
## Fairwork Allocation

<table>
<thead>
<tr>
<th>Pre processing &amp; Sampling</th>
<th>Repartitioning</th>
<th>Batching at reduce side</th>
</tr>
</thead>
</table>
| • Existence of a pre-processing phase (a separate MR job) that builds information about the number of entities present in each block.  
• Splitting large reduce keys into several medium-load reduce keys (identified by sampling), and assigning medium-load keys to reduce tasks using a bin packing algorithm. | • Handling data skew by means of an adaptive load balancing strategy. Can be done by computing local statistics in the map phase and aggregating them to produce global statistics. | • Can be done by reducing the number of disk accesses. Instead of writing to intermediate files at map side, data is shuffled directly and then written to a file at reduce side, one file per reduce task ( batching). |
FUTURE WORK

- We aim to compile a comprehensive list of technologies and tools in the context of analytical query processing that implement our surveyed techniques and suggested improvements that are either already successfully being used in the industry or are being studied in the academia.

- Also we plan to investigate the possible hybrid approaches that might arise by combining various different surveyed techniques and analyze the associated flaws and/or trade-offs that might occur from their implementation.