Introduction to Tajo

(Tajo: A Distributed Data Warehouse System for Hadoop)

Hyunsik Choi

(hyunsik at apache dot org)

Introduction

- Tajo is a distributed warehouse system for Hadoop.
- Tajo is designed for low-latency and scalable ETL and ad-hoc queries on large-data sets by leveraging advanced database techniques.
- Support SQL standards
- Tajo has its own query engine and uses HDFS as a primary storage layer.
  - Direct control of distributed execution and data flow
  - A variety of query evaluation strategies
  - More optimization opportunities
- Row/Native columnar execution engine and optimizer
- An alternative choice to Hive/Pig on the top of MapReduce

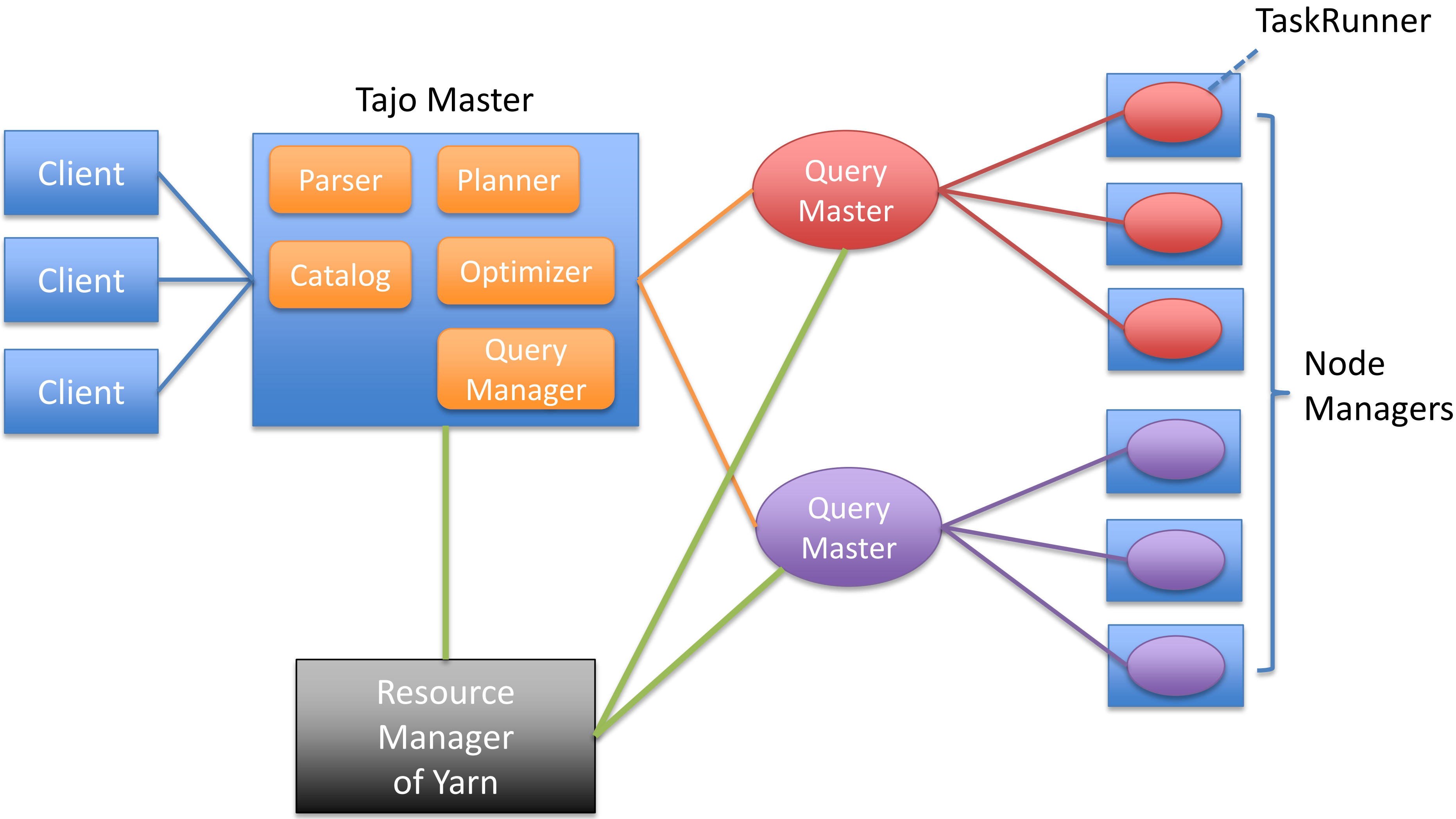
Design Principles

- Scalability
- Low Latency
- Fault Tolerance
- Application of Advanced Database Techniques

Overall Architecture

- The architecture of Tajo follows the master-worker model.
  - TajoMaster
    - A dedicated server for providing client service and coordinating QueryMasters
  - For each query , one QueryMaster and many task runners work together.
- TaskRunner includes a local query engine that executes a directed acyclic graph (DAG) of physical operators.
- Tajo employs Hadoop Yarn as a resource manager for large clusters.

Overall Architecture



Data Structure & Terminology

- Logical Plan
  – A representation of relational algebra
  – Implemented in a graph of java objects
  – (de) serialized in a JSON document via GSON
- Physical Execution Plan
  – A directed acyclic graph of *physical operators*
- Execution Block
  – A *subquery* executed across a number of cluster nodes.
- Distributed Query Execution Plan (DQEP)
  – A directed acyclic graph of *execution blocks*.

Tajo Master

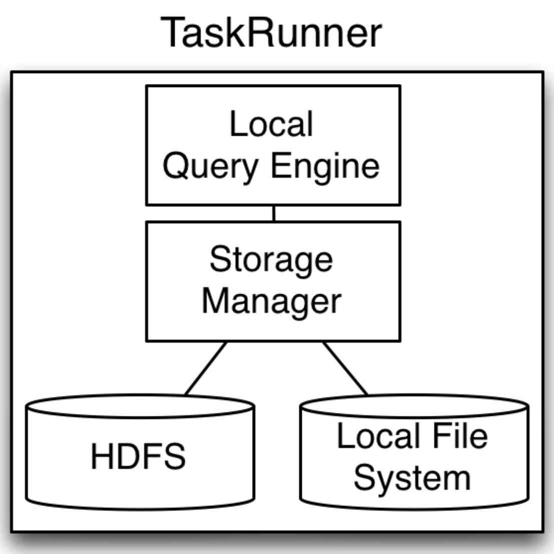
- It provides a client service that enables users to submit to queries and to ask catalog information.
- It coordinates a number of QueryMasters.
- It provides a catalog service.
  - Catalog Server is embedded in Tajo Master.
  - Catalog Server
    - Maintains the table and index descriptions
    - Supports some statistics (min, max value, num of rows, and bytes)
    - Uses Apache Derby as the persistent storage via JDBC.

Query Master

- Query Master is responsible for
  - controlling a distributed execution plan (i.e., a DAG of execution blocks)
  - launching TaskRunner containers
  - coordinating TaskRunners
  - localizing tasks
  - scheduling tasks to TaskRunners
- Also, It monitors running tasks, collects the statistics, reports them to TajoMaster.

TaskRunner

- It contains a local query engine that actually executes tasks.
- A TaskRunner is launched by NodeManager of Hadoop Yarn.
- It is reused within a execution block.
  - According to the workload of the execution block, different resources are assigned to containers.

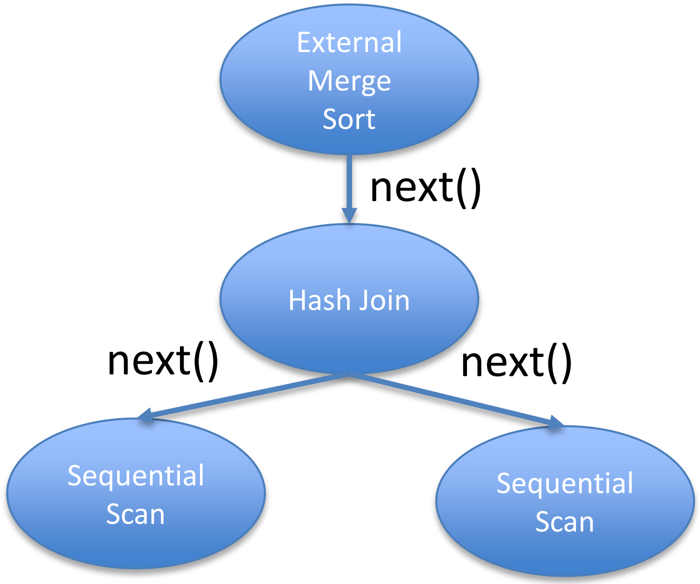


TaskRunner

- In TaskRunner, a physical execution plan is completed according to the container's resource or the size of input data.
  - It allows a TaskRunner to choose more efficient execution plans and to maximize the utilization of hardware capacity.
  - This feature is useful for heterogeneous environments.

Local Query Engine

- Now, Tajo provides a row-based execution engine.
  - It executes a physical execution plan.
  - The data flow is the pull-based iterator model like traditional row-store database.
    - For each tuple, the root node calls “next()” on its children, who call “next” on their child recursively.



Physical Operators

- We already implemented various row-based physical operators:
  - Block nested Loop Join
  - Hash join
  - Merge join
  - External sort
  - In-memory sort
  - Selection
  - Projection
  - Sort aggregation
  - Hash aggregation

Planning & Optimization

- The following optimization was implemented:
  - Simplification of Algebraic expressions
  - Selection & projection push down optimization
  - Modification of the order of some physical operators
  - Cost-based Join ordering (still working, but almost done)

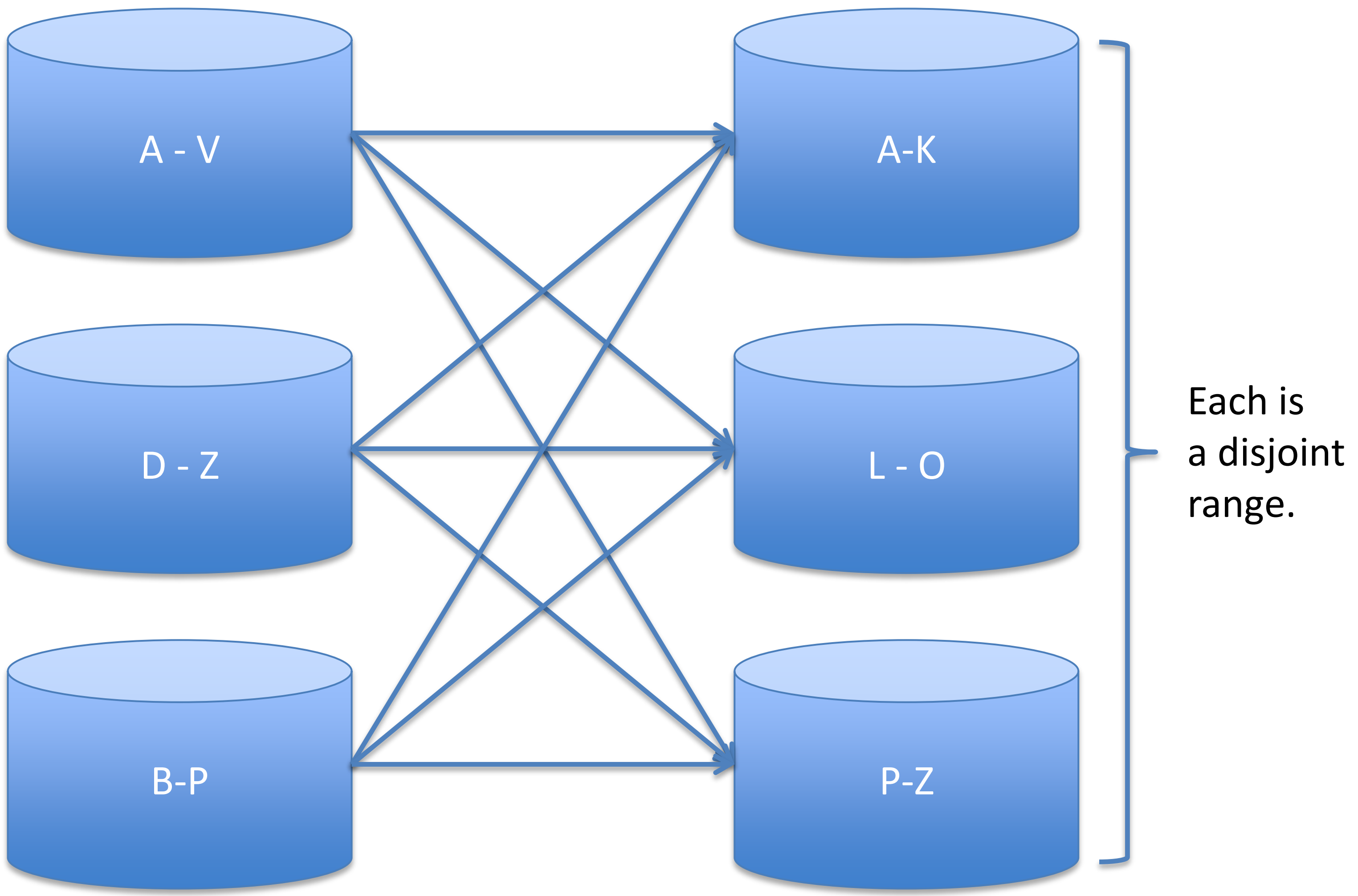
Storage System

- Provide split methods that divide an input data set into a number of fragments (similar to splits).
- Provide Scanner and Appender interfaces
  - Specialized to structured data
- Enable users to implement their own scanners or appenders.
  - If you can model unstructured data to structured data, the unstructured data is available in Tajo.
- Currently, it provides various row/columnar store file formats, such as CSVFile, RowFile, RCFile, and Trevni (still unstable).
  - They can be easily ported as follows:
    - https://github.com/tajo-project/tajo/blob/master/tajo-core/tajo-core-storage/src/main/java/tajo/storage/rcfile/RCFileWrapper.java

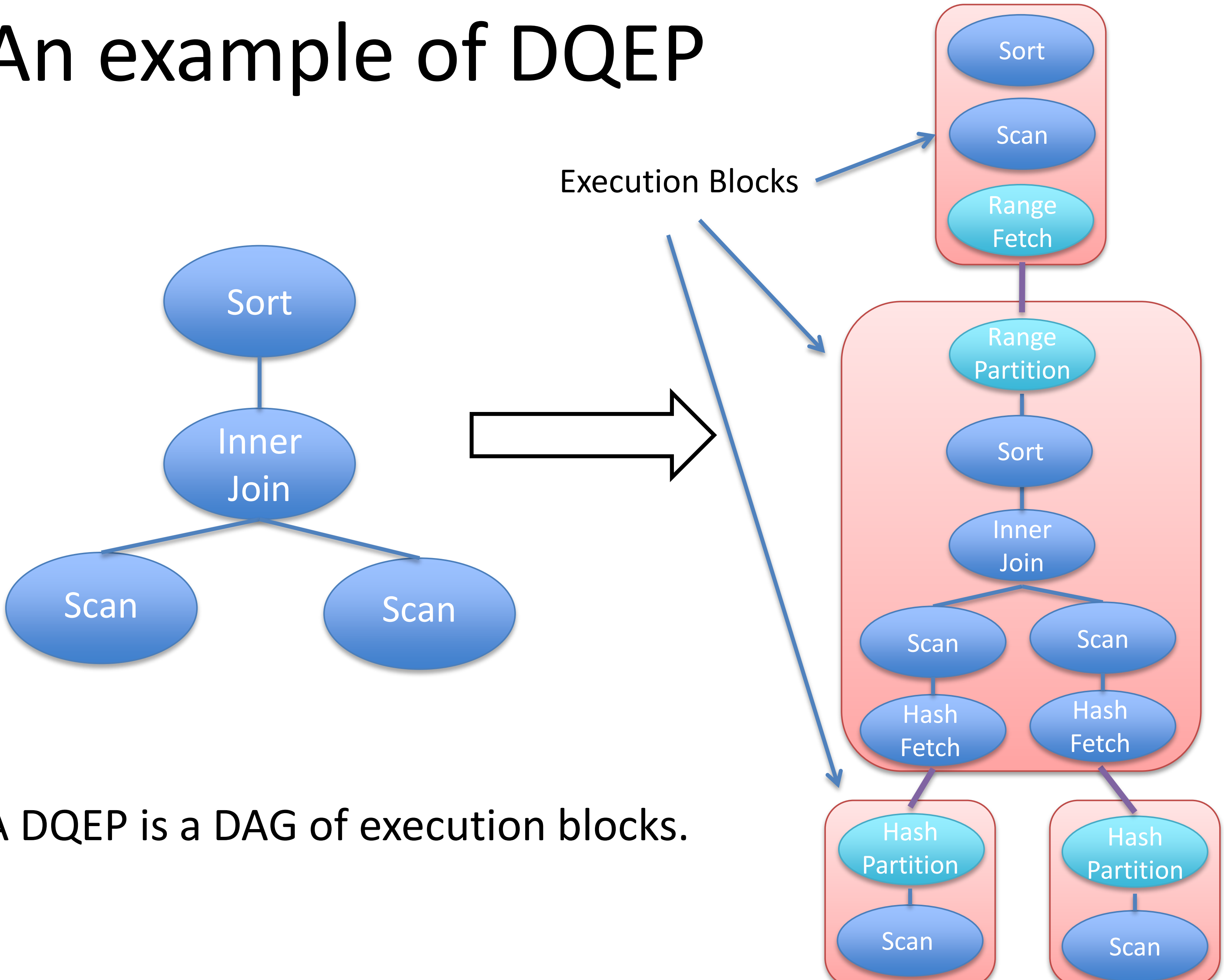
Repartitioning

- In distributed query execution, data repartition (like shuffle of MapReduce) is very important.
- Tajo has two kind of repartition methods:
  – Range repartition
    - Boosts the performance of sort operation
    - Also used to generate totally-ordered multiple files
  – Hash repartition
    - Without sort, Tajo can repartition the intermediate data to workers using the hash repartition.

Repartitioning: Range Repartition

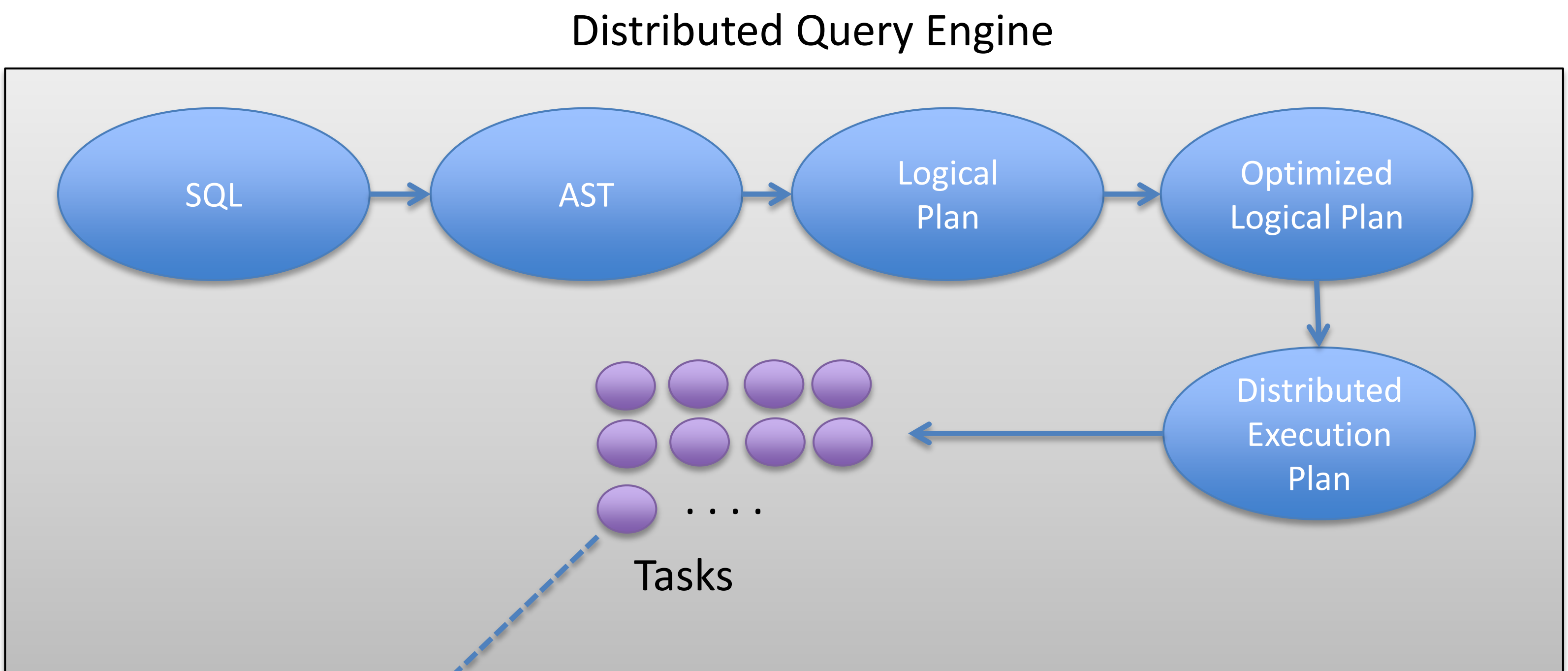


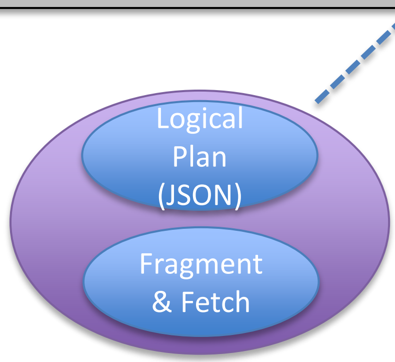
An example of DQEP



A DQEP is a DAG of execution blocks.

Query Transformation





A Task

- Fragment is similar to InputSplit of MapReduce.
  - Additionally, it includes **Schema** and **Metadata**.
- Fetch is a URI to be fetched.
- According to DQEP, each execution block is localized and is transformed to a number of tasks.

QUERY EVALUATION

Distributed Sort

- Tajo provides the range repartition.
  - So, it easy to support the distributed sort.
- Usually, a distributed sort requires two phases.
  1. First Phase: Sort on splits
  2. Range Repartition
  3. Second Phase: Sort on repartitioned data
- Finally, the multiple and ordered output files are written to HDFS.

Aggregation

- Tajo provides the range/hash repartitions.
- Aggregation can use either repartition.
  - According to the sort order of input table or consecutive operations, we can use different repartition methods.
- Usually, it requires two phases, but various aggregation algorithms can be mixed.
  1. First Phase: Sort Aggregation (or Hash Aggregation)
  2. Hash Repartition
  3. Second Phase: Hash Aggregation (if intermediate data fits in memory)

Join

- It supports existing various join strategies used in shared-nothing databases (or Hive).
  - Broadcast Join
  - Repartition Join (hash and range)
- It also requires two phases and can mix various join algorithms.
  1. First Phase: Scan (and filter by selection pushdown)
  2. Hash (or range repartition)
  3. Hash Join (or Merge Join if range repartition is performed)

Join

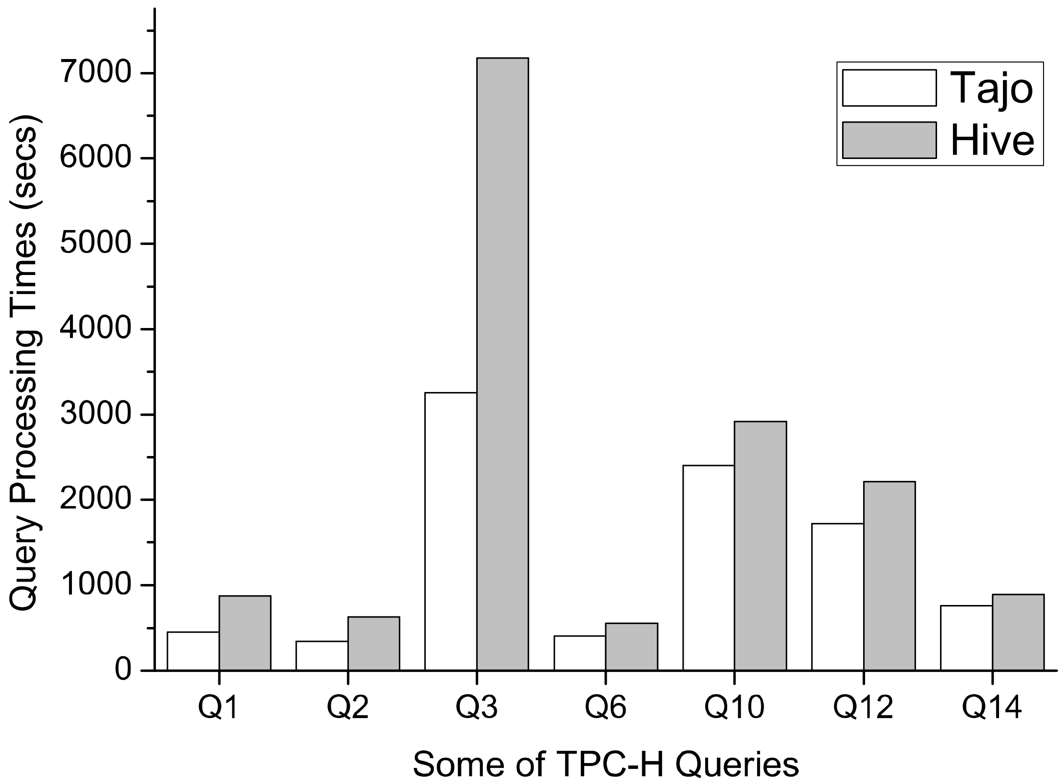
- In addition, we provide a new join strategy if the larger table is sorted on a join key.
  1. A smaller table is repartitioned via range repartition.
  2. Assign the range partitions to nodes whose large table partitions correspond to the join key range.
  3. Each node performs the merge join.

EXPERIMENTS

Experiments

- We carried out the experiments in comparison with Hive on the top of MapReduce.
- The following results are based on the experiment that we carried out in August, 2012.
- Experimental Environments
  - 1TB TPC-H Data Set
    - Some of TPC-H Queries
  - 32 Cluster Nodes
    - Intel i5
    - 16GB Memory
    - 1TB HDD x 2
    - 1G ether networks
  - Hadoop 0.20.2-cdh3u3

Tajo vs. Hive on TPC-H 1TB



Experiment results submitted in ICDE 2013
(Evaluated in August, 2012)

FUTURE WORKS

Future Works

- Cost-based Optimization
  - Cost-based Join Order Selection (almost done)
  - Progressive Optimization
    - During query processing, the optimizer will try to reoptimize the remain parts of the determined query plan.
- Column Store
  - Native Columnar Execution Engine
  - Optimizer for Columnar Execution Engine
  - Porting ORC File
- Low Latency
  - Online Aggregation
    - Enable users to get estimates of an aggregate query in an online fashion as soon as the query is issued
  - Intermediate data streaming (push-based transmission)
- More Mature Physical Operators for the Row-based Engine

Appendix - History

- August, 2012: Benchamark Test
  - Some TPC-H queries, TPC-H 1TB on 32 cluster nodes
  - Tajo outperforms about Hive 1.5-3 times
- August, 2012: Demo Paper was submitted to IEEE ICDE 2013.
- October, 2012: Tajo was accepted to IEEE ICDE 2013.
  - We will visit Brisbane in order to demonstrate Tajo in April 2013.
- January, 2013: Benchmark Test by Gruter
  - Gruter (A bigdata company in South Korea - http://www.gruter.com/)
  - TPC-H 1,3,and 6 queries, 100GB on 10 cluster nodes
  - Tajo outperforms Hive about 2-3 times, and is comparable to Cloudera Impala.