HAWQ: A Massively Parallel Processing SQL Engine in Hadoop

Lei Chang, Zhanwei Wang, Tao Ma, Lirong Jian, Lili Ma, Alon Goldshuv Luke Lonergan, Jeffrey Cohen, Caleb Welton, Gavin Sherry, Milind Bhandarkar Pivotal Inc
{lchang, zwang, tma, ljian, lma, agoldshuv, llonergan, jcohen, cwelton, gsherry, mbhandarkar}@gopivotal.com

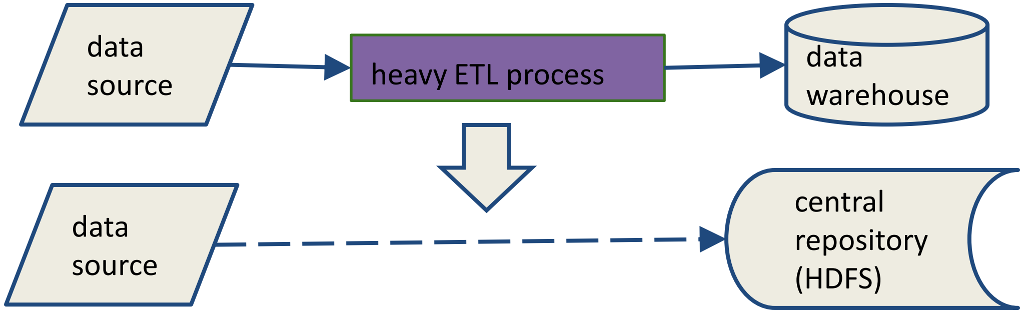
Guoyao Feng
CS 848
University of Waterloo

Agenda

- Introduction
- HAWQ Architecture
- Query Processing
- Interconnect
- Transaction Management
- Extension Framework
- Experiments
- Conclusions

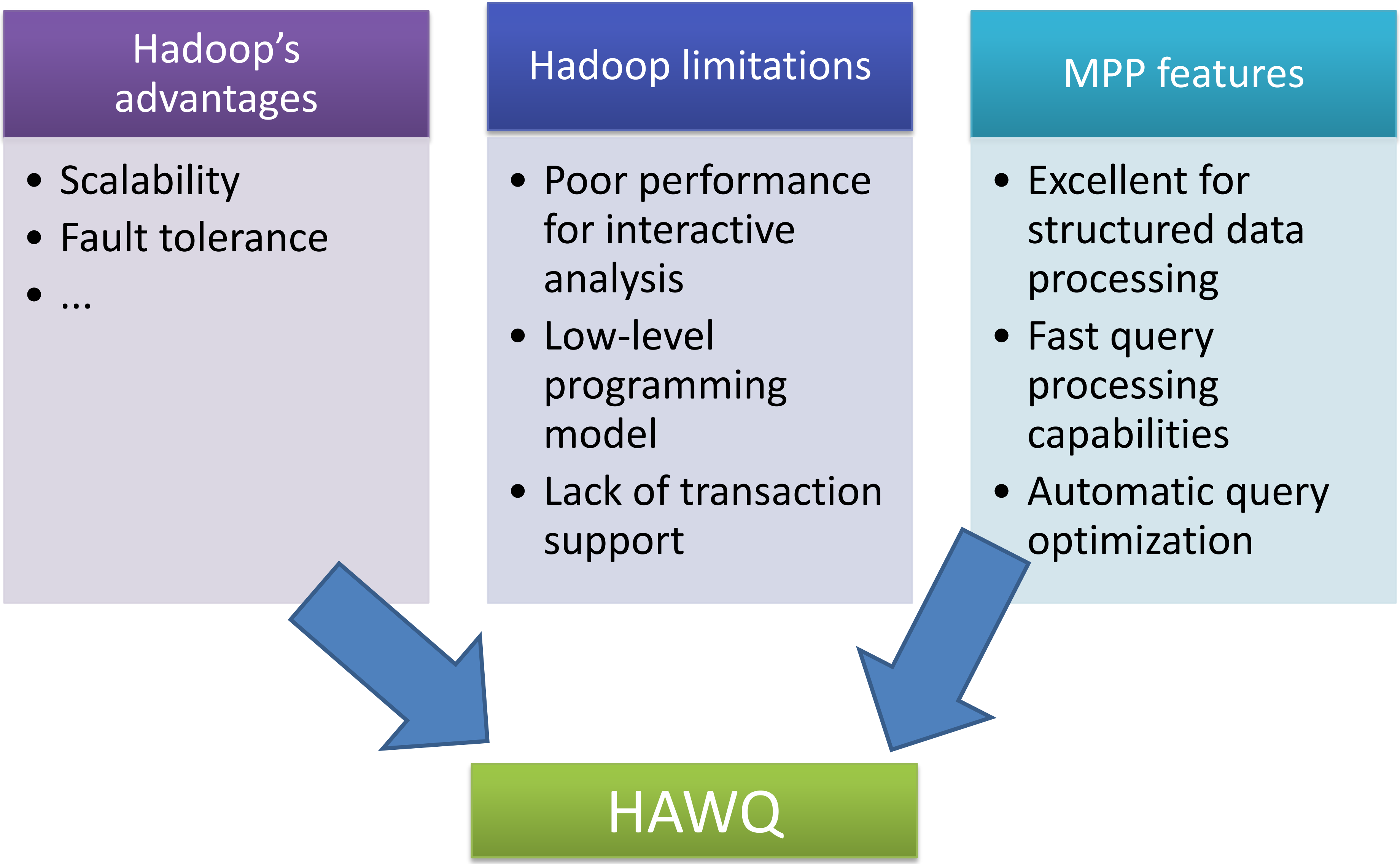
Background

- Shifting Paradigm



- Requirements for Hadoop
  - **Interactive Queries**
  - Scalability
  - **Consistency**
  - Extensibility
  - **Standard Compliance**
  - **Productivity**

Motivation



Hadoop's advantages

- Scalability
- Fault tolerance
- …

Hadoop limitations

- Poor performance for interactive analysis
- Low-level programming model
- Lack of transaction support

MPP features

- Excellent for structured data processing
- Fast query processing capabilities
- Automatic query optimization

HAWQ

Introducing HAWQ

- HAWQ – A MPP SQL query engine on top of HDFS
  - Combines the merit of Greenplum Database and Hadoop distributed storage
  - SQL compliant
  - Support large-scale analytics for big data (MADlib)
  - Interactively query various data sources in different Hadoop format

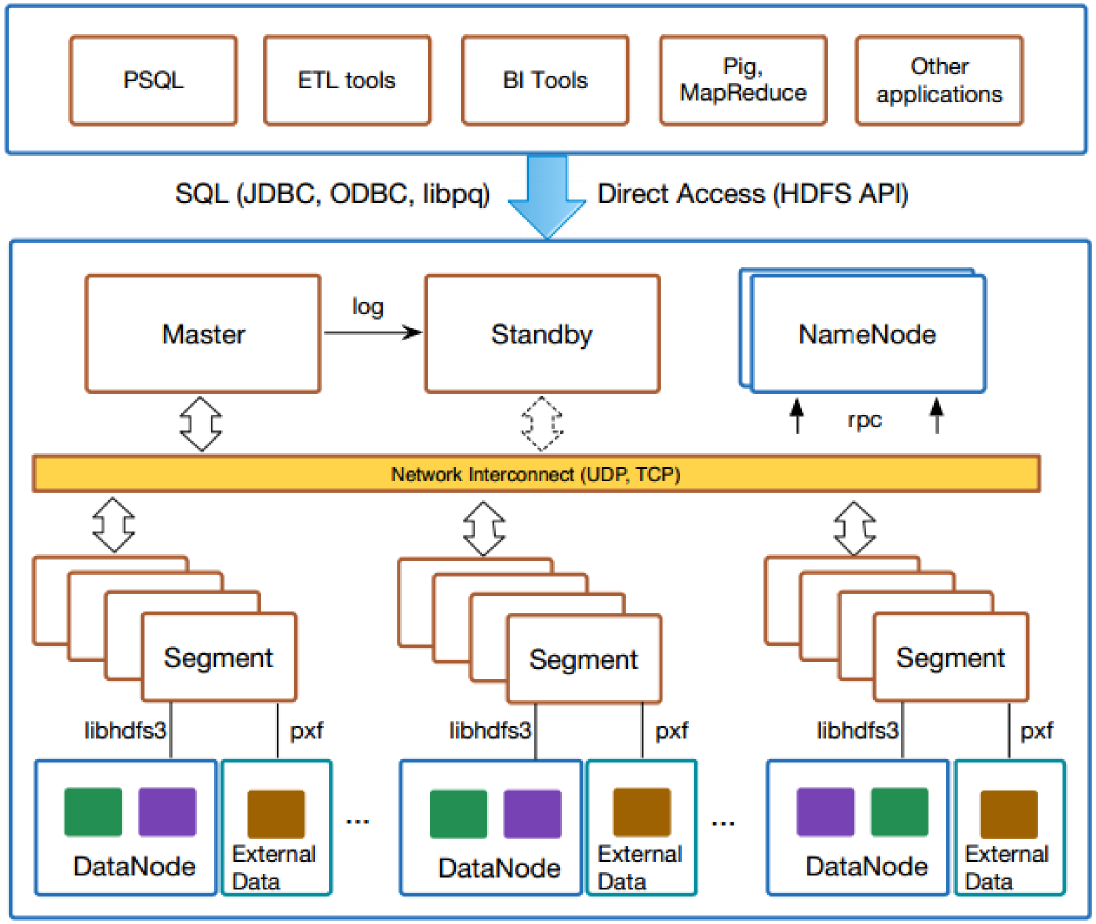
Agenda

- Introduction
- HAWQ Architecture
- Query Processing
- Interconnect
- Transaction Management
- Extension Framework
- Experiments
- Conclusions

HAWQ Architecture

MPP shared-nothing compute layer

Distributed storage layer



HAWQ Architecture: Interface

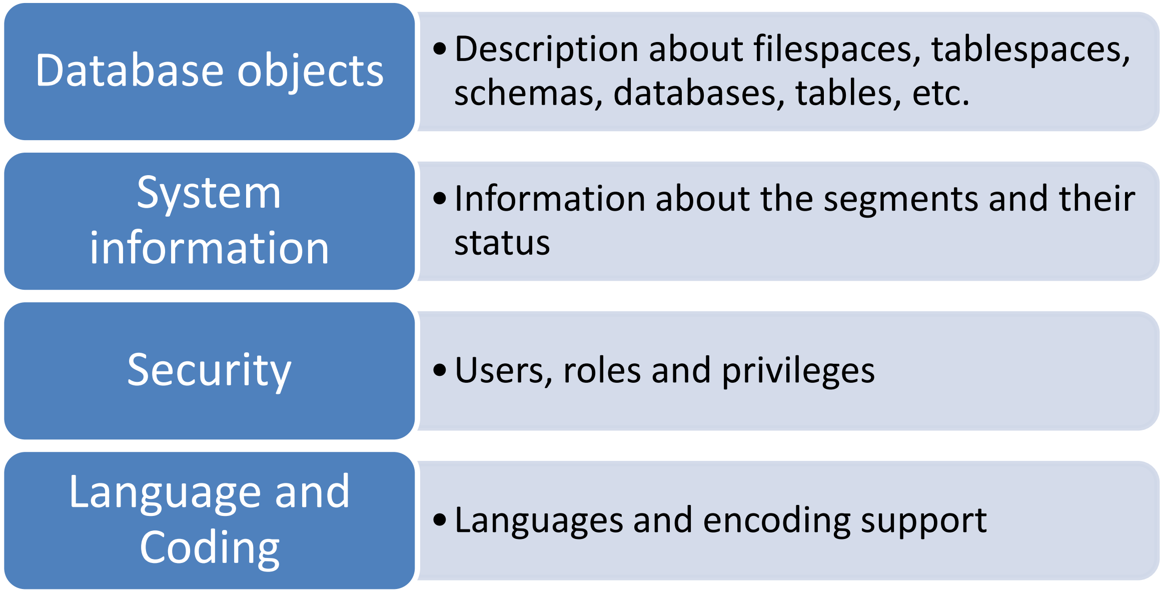
Challenge: Simplify the interaction between HAWQ and external systems

Design Choice: Enhance standard interfaces with open data format support

- External systems can bypass HAWQ to access table files on Hadoop (HDFS API)
- Open MapReduce InputFormats and OutputFormats

HAWQ Architecture: Catalog Service

- Catalog describes the system and all objects in the system



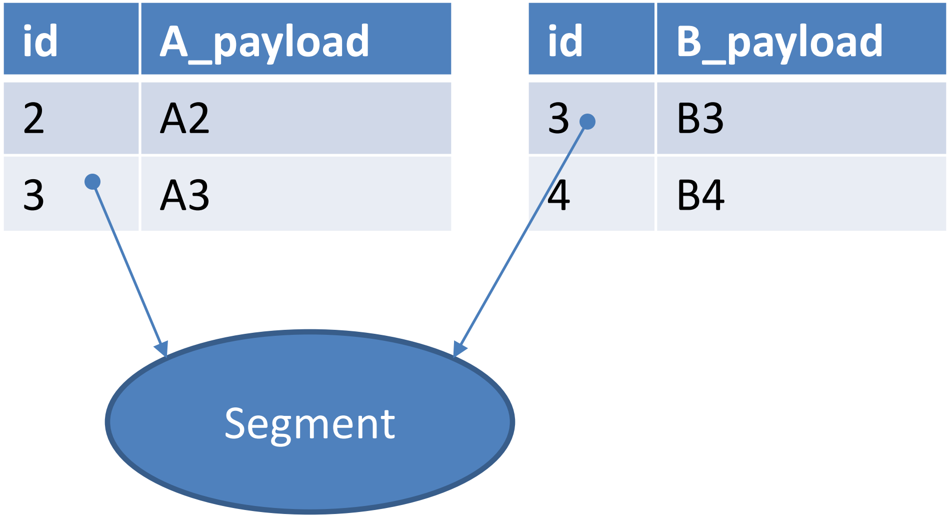
Database objects	• Description about filespaces, tablespaces, schemas, databases, tables, etc.
System information	• Information about the segments and their status
Security	• Users, roles and privileges
Language and Coding	• Languages and encoding support

- CaQL: a simplified catalog query language for internal access
  - Easy to implement
  - Scalability

HAWQ Architecture: Data Distribution

- Hash distribution
  - Most frequently used strategy
  - Align tables to improve some import query patterns

- Random distribution
  - Distribute rows in a round-robin fashion
  - Even data distribution
  - Works well for table with a small number of distinct values



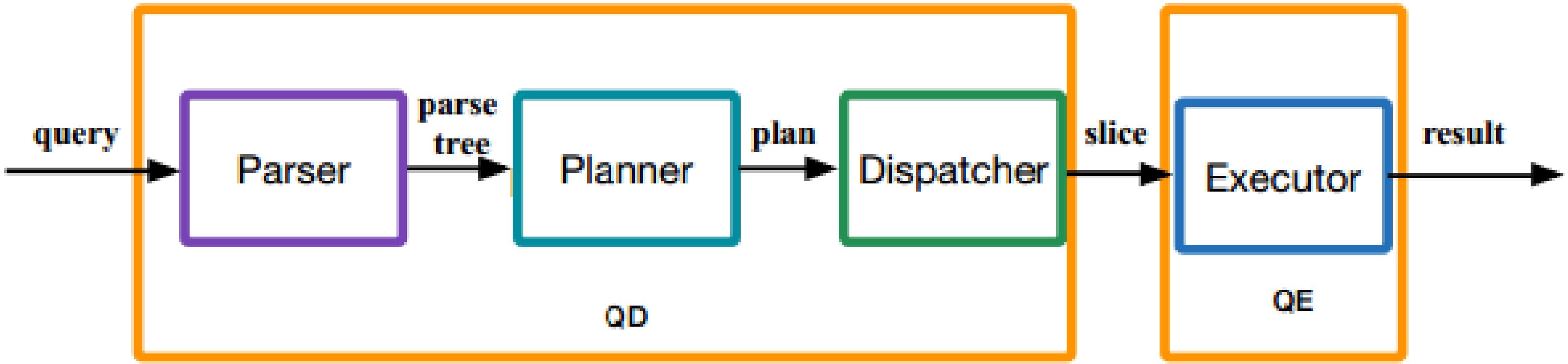
id	A_payload
2	A2
3	A3

id	B_payload
3	B3
4	B4

HAWQ Architecture: Data Distribution

- Table partitioning
  - Range partitioning + list partitioning
  - Creates an *inheritance relationship* between the top-level parent table and child tables
  - Improve query performance if only a subset of partitions are accessed

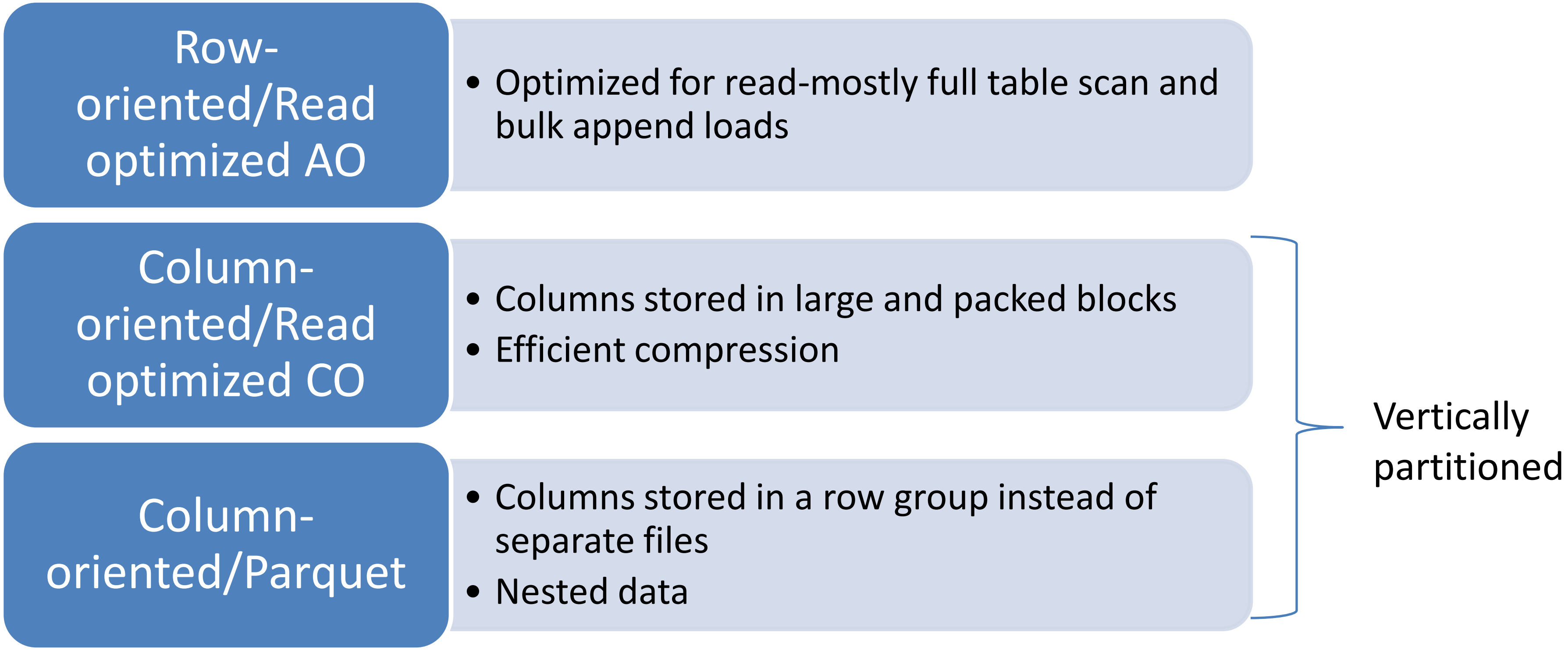
HAWQ Architecture: Query execution workflow



QD -> Query Dispatcher
QE -> Query Executer

HAWQ Architecture: Storage

- HAWQ supports a variety of storage models on HDFS
- Transformation among different storage models done at the user layer



Row-oriented/Read optimized AO

- Optimized for read-mostly full table scan and bulk append loads

Column-oriented/Read optimized CO

- Columns stored in large and packed blocks
- Efficient compression

Column-oriented/Parquet

- Columns stored in a row group instead of separate files
- Nested data

Vertically partitioned

HAWQ Architecture: Fault Tolerance

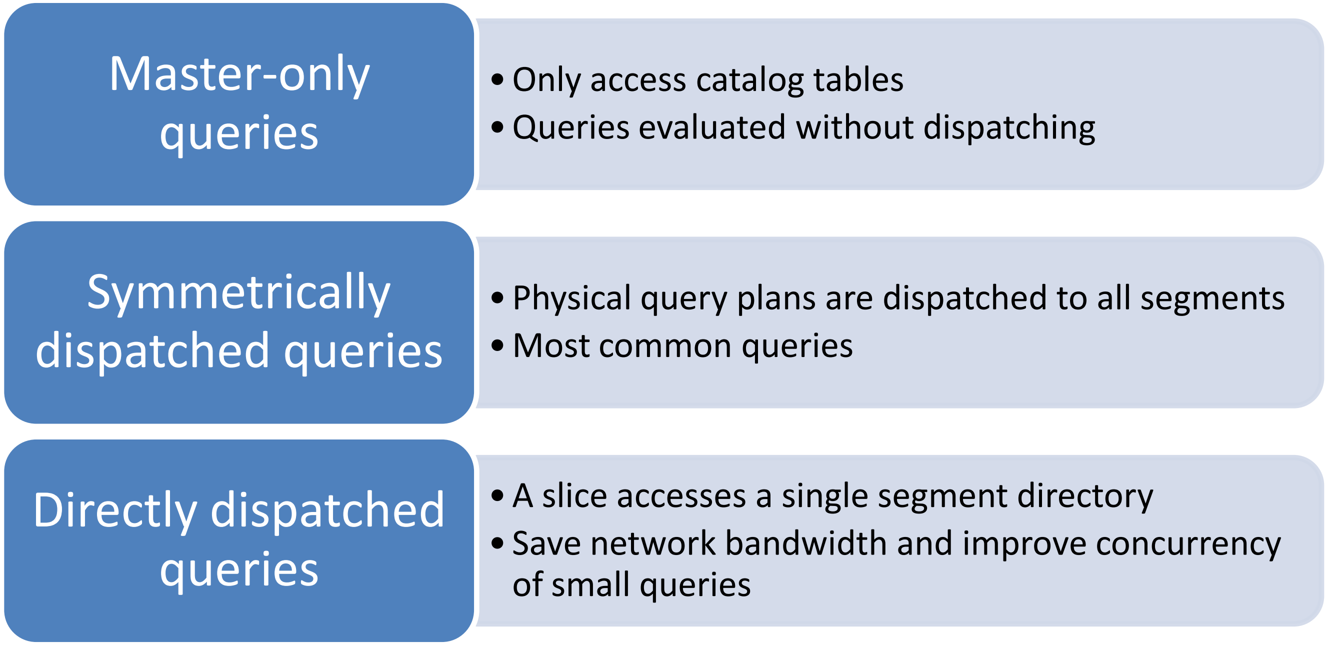
- Master
  - Warm standby kept synchronized with a *transaction log replication process*
- Segment
  - Stateless
  - Simple recovery
  - Better availability
- Disk
  - Disk failure of user data handled by HDFS
  - Disk failure of intermediate query output marked by HAWQ

Agenda

- Introduction
- HAWQ Architecture
- Query Processing
- Interconnect
- Transaction Management
- Extension Framework
- Experiments
- Conclusions

Query Processing

- Query Types



Master-only queries

- Only access catalog tables
- Queries evaluated without dispatching

Symmetrically dispatched queries

- Physical query plans are dispatched to all segments
- Most common queries

Directly dispatched queries

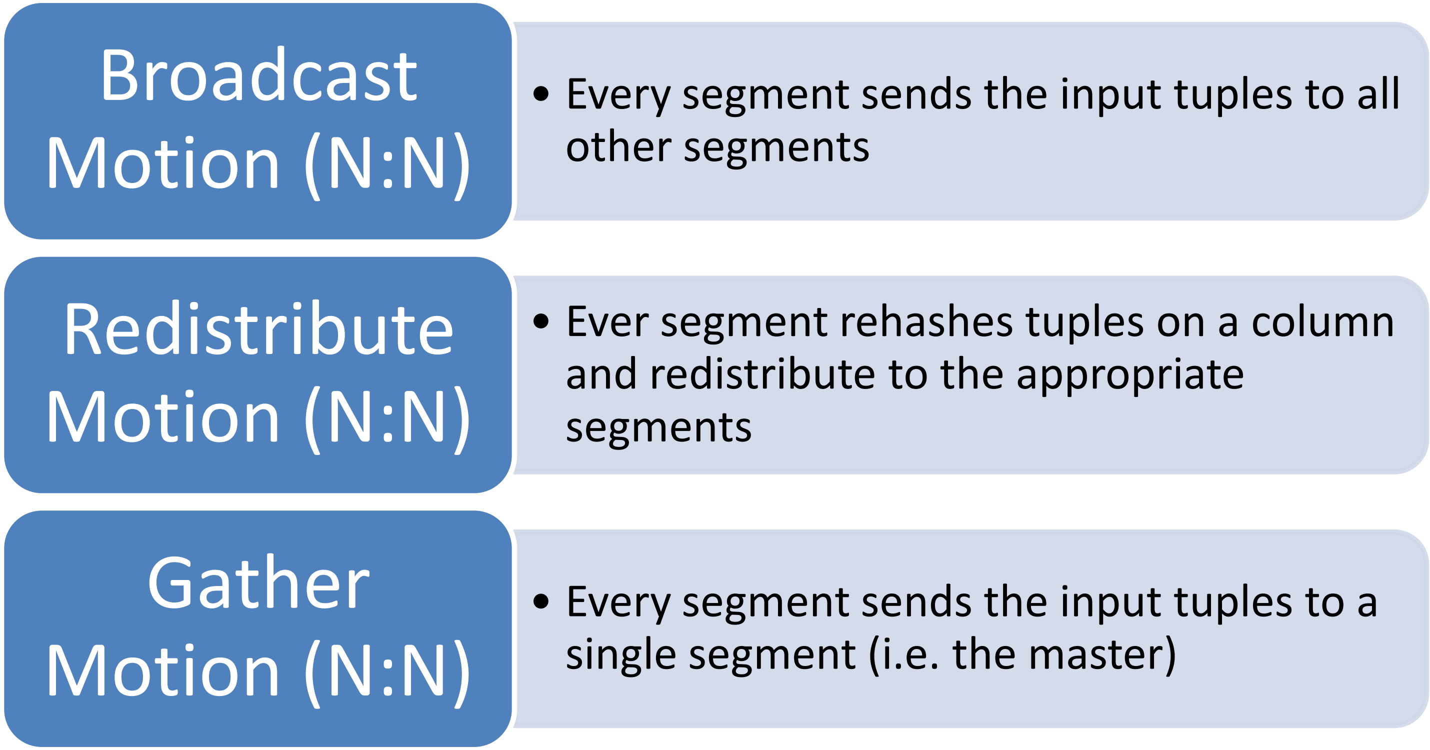
- A slice accesses a single segment directory
- Save network bandwidth and improve concurrency of small queries

- Query Plan
  - Relational operators: scans, joins, etc
  - *Parallel ‘motion’ operators*

pipelined

Query Processing

- Parallel ‘motion’ Operators



Broadcast Motion (N:N)

- Every segment sends the input tuples to all other segments

Redistribute Motion (N:N)

- Ever segment rehashes tuples on a column and redistribute to the appropriate segments

Gather Motion (N:N)

- Every segment sends the input tuples to a single segment (i.e. the master)

Query Processing

- Problem: A large number of QEs connect to the master and query meta data
- Solution:
  - Dispatch the metadata along with the execution plan (self-described plan)
  - Store read-only meta data on the segments

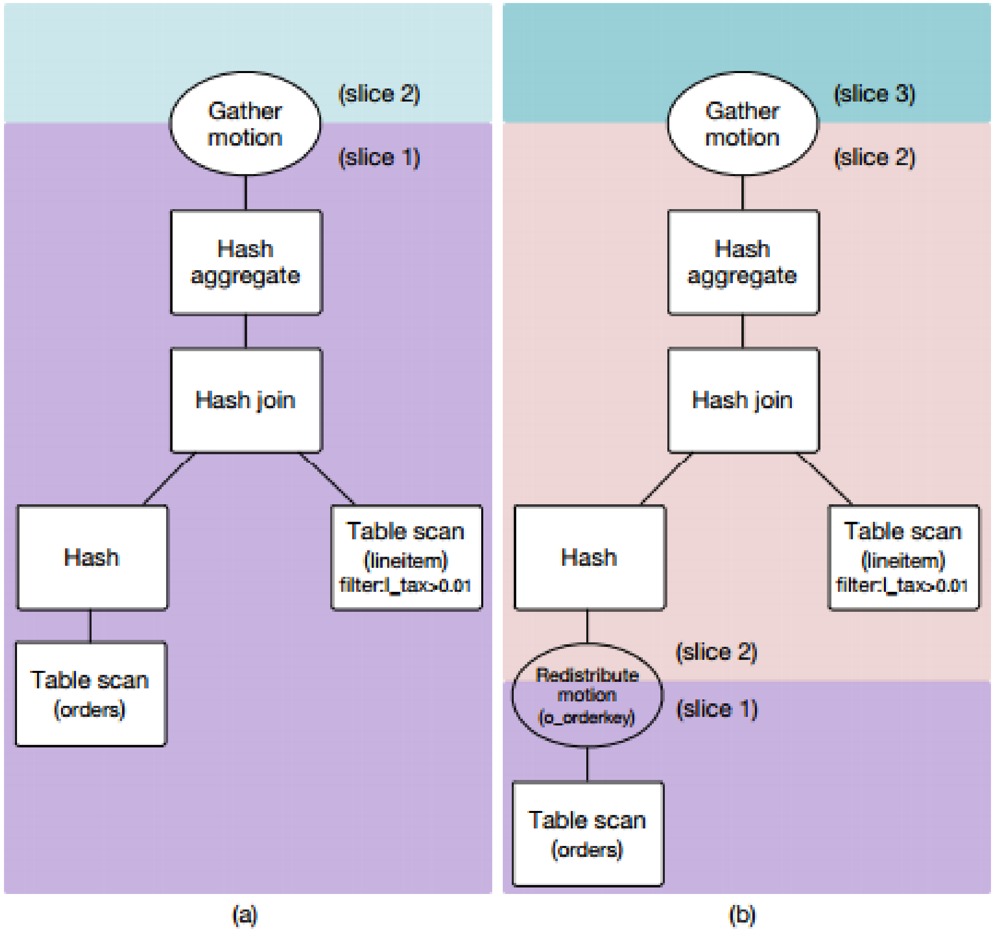
Query Processing Example

```
SELECT l_orderkey, count(l_quantity)
FROM lineitem, orders
WHERE l_orderkey=o_orderkey AND l_tax>0.01
GROUP BY l_orderkey;
```

Data motion boundary

1-gang (QD)

N-gang (QEs)

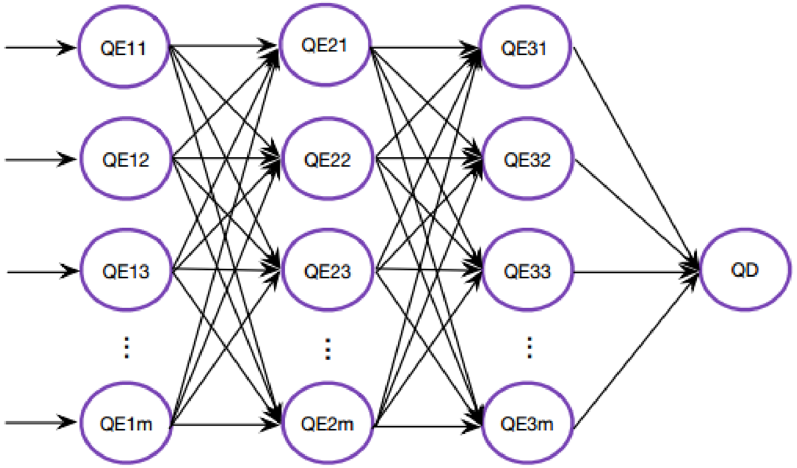


Agenda

- Introduction
- HAWQ Architecture
- Query Processing
- Interconnect
- Transaction Management
- Extension Framework
- Experiments
- Conclusions

Interconnect

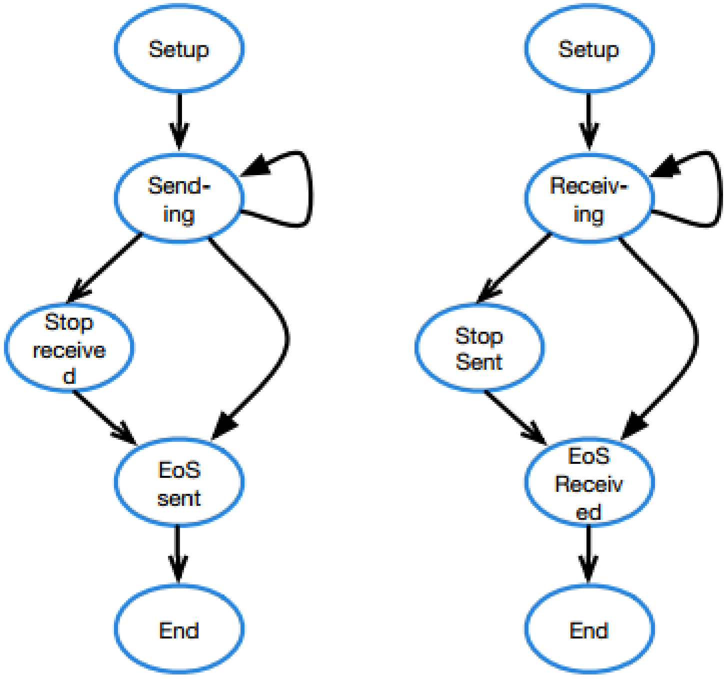
- Issues with TCP interconnect
  - Port number limitation (60k)
  - Expensive connection setup



- UDP interconnect to rescue
  - Reliability
  - Ordering
  - Flow control
  - Performance and scalability
  - Portability

UDP Interconnect

- Protocol



(a) Sender

(b) Receiver

UDP Interconnect

- Implementation details
  - Typical implementation of reliable UDP solution
  - Sender maintains a send queue and an expiration queue
  - Flow control window cut down to a minimal predefined value followed by slow start
  - OUT-OF-ORDER and DUPLICATE message
  - Send a status query message to eliminate deadlock

Agenda

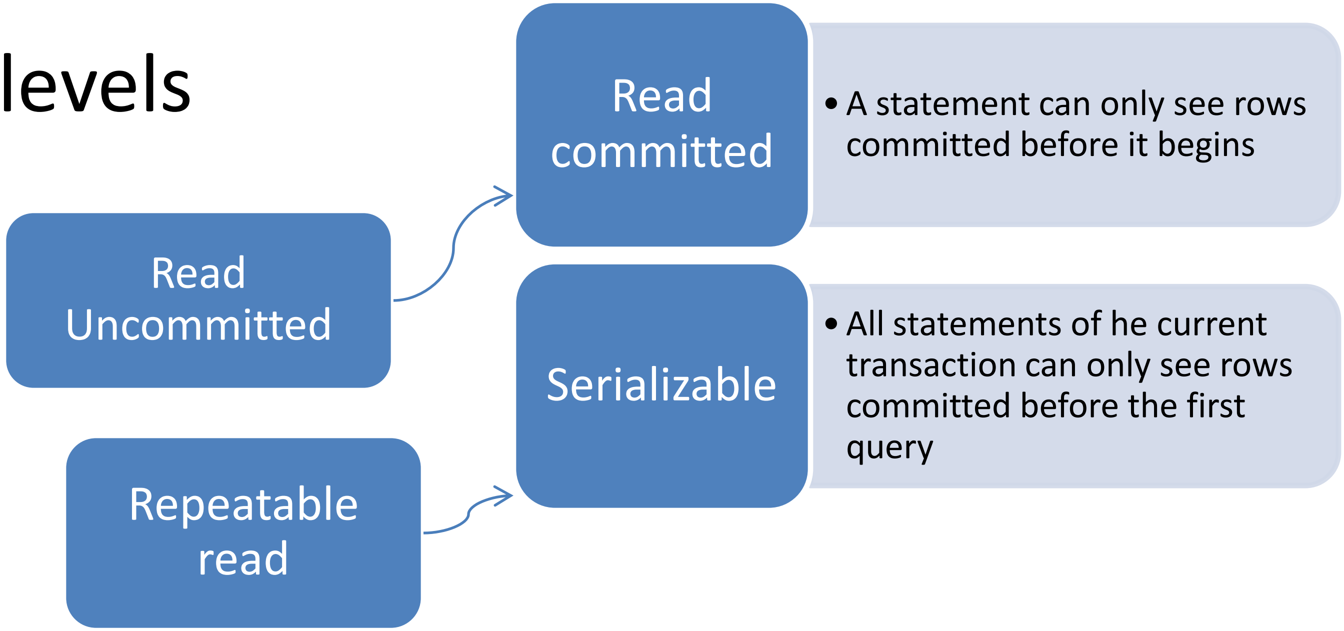
- Introduction
- HAWQ Architecture
- Query Processing
- Interconnect
- Transaction Management
- Extension Framework
- Experiments
- Conclusions

Transaction Management

- Catalog
  - Write ahead log (WAL)
  - Multi-version concurrency control (MVCC)
  - Support snapshot isolation
- User data
  - Append-only HDFS files avoids complexity
  - System catalog records logical file length to control visibility
- No distributed commit protocols
  - Transaction is only noticeable on the master node
  - Self-described plans convey visibility information to segments
  - Transaction only occurs on master node

Transaction Management

- Isolation levels



Read Uncommitted

Repeatable read

Read committed

- A statement can only see rows committed before it begins

Serializable

- All statements of he current transaction can only see rows committed before the first query

- Locking
  - Control the conflicts between concurrent DDL and DML statements

Transaction Management

- Pivotal HDFS adds truncate to support transaction
  - Truncate is necessary for undoing changes by aborted transactions
  - Hadoop HDFS currently does not support truncate
  - Atomicity is guaranteed
    - Only one update operation allowed
    - Truncate applied to closed files only
- Concurrent Updates
  - Lightweight *swimming lane* approach for concurrent inserts
  - Catalog table is used to manage user table files

Agenda

- Introduction
- HAWQ Architecture
- Query Processing
- Interconnect
- Transaction Management
- Extension Framework
- Experiments
- Conclusions

Extension Framework

PXF: a fast and extensible framework connecting HAWQ to **any** data store of choice

- Users implement a parallel connector API to create connector for data stores

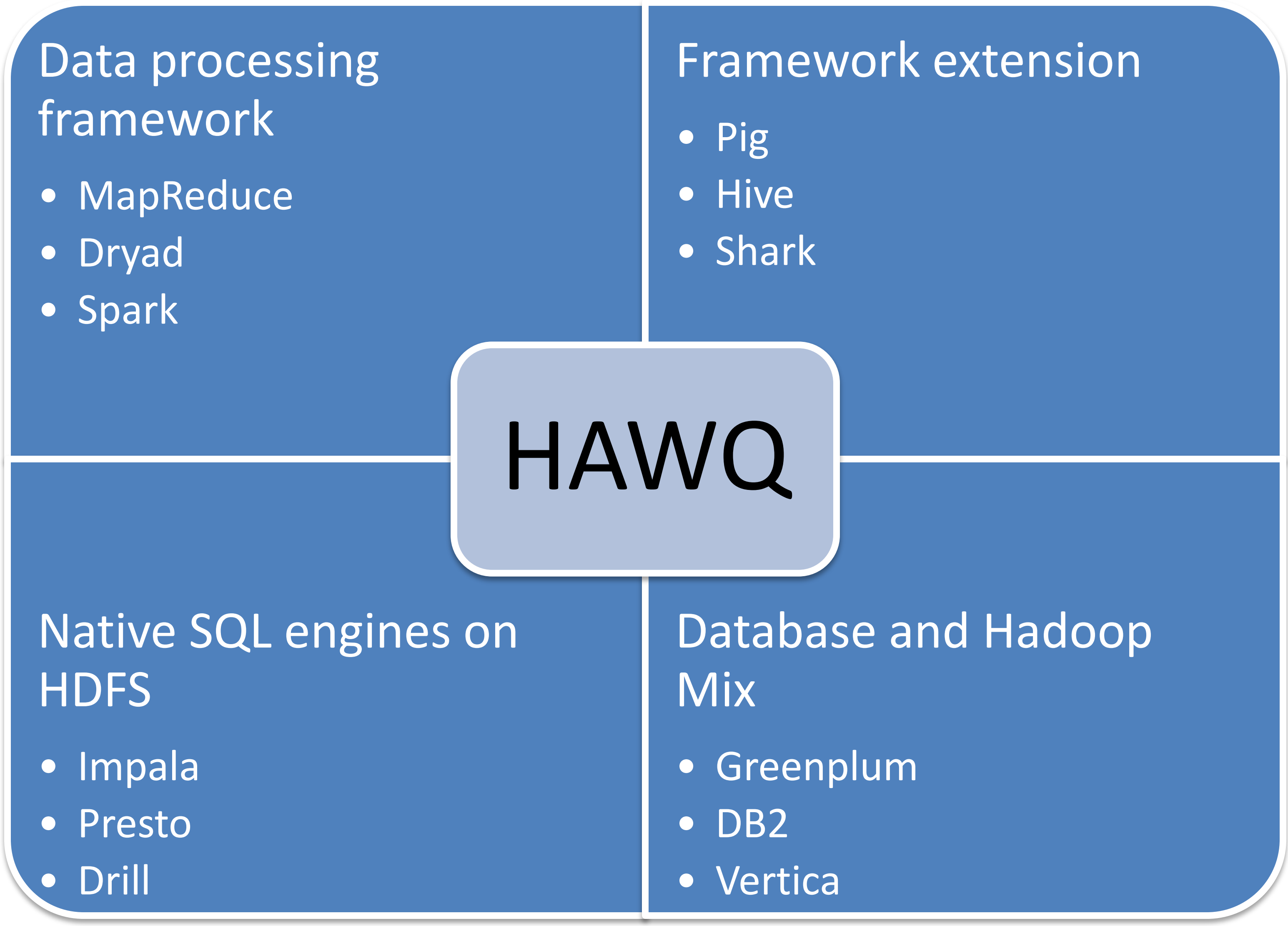
```
  CREATE EXTERNAL TABLE my_hbase_sales (
    recordkey BYTEA,
    "details:storeid" INT,
    "details:price" DOUBLE)
  LOCATION('pxf://<pxf service location>/sales?
profile=HBase')
  FORMAT 'CUSTOM' (formatter='pxfwritable_import
');
```

```
SELECT sum("details:price")
FROM my_hbase_sales
WHERE recordkey < 20130101000000;
```

Extension Framework

- Benefits
  - A real connection among various data stores to share data
  - Complex joins between internal HAWQ tables and external PXF tables
  - External jobs can run faster with HAWQ/PXF
- Advanced functionality
  - Exploits data locality to minimize network traffic
  - Exposes a filter push down API
  - Planner statics collection

HAWQ in Distributed Data Processing



Data processing framework

- MapReduce
- Dryad
- Spark

Framework extension

- Pig
- Hive
- Shark

HAWQ

Native SQL engines on HDFS

- Impala
- Presto
- Drill

Database and Hadoop Mix

- Greenplum
- DB2
- Vertica

Agenda

- Introduction
- HAWQ Architecture
- Query Processing
- Interconnect
- Transaction Management
- Extension Framework
- Experiments
- Conclusions

Experiments: Competing Systems

- Stinger
  - a community-based effort to optimize Apache Hive
  - reported 35x-45x faster than the original Hive
  - 36GB RAM on each node in YARN, 4GB minimum memory for each container
- HAWQ
  - 6 HAWQ segments are configured on each node
  - 6GB memory is allocated to each segment.

Experiments: TCP-H Results

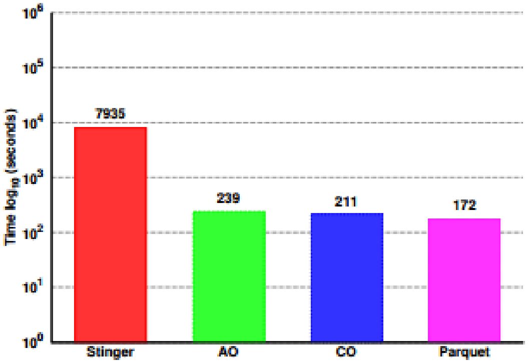


Figure 6: Overall execution time with 160GB

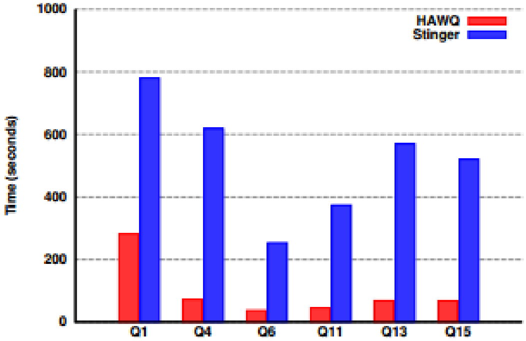


Figure 8: Simple selection queries

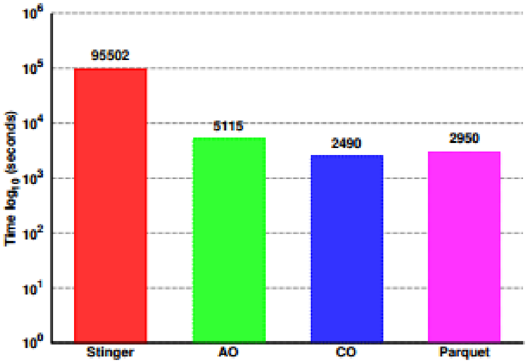


Figure 7: Overall execution time with 1.6TB

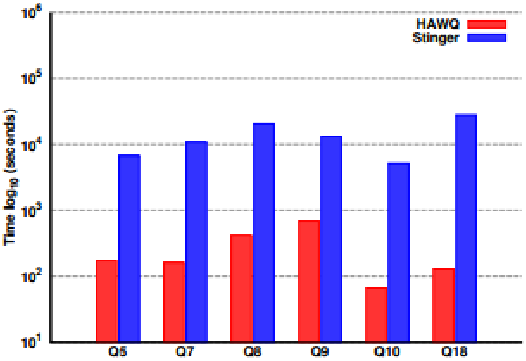


Figure 9: Complex join queries

Experiments

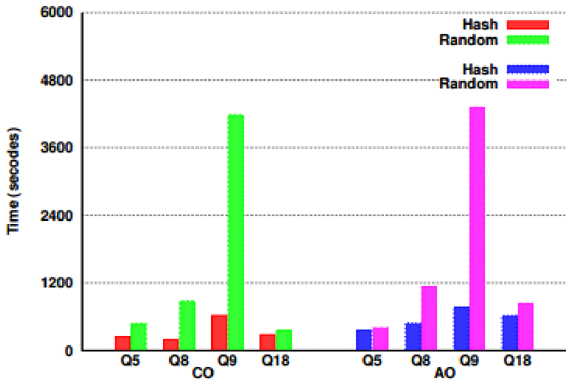


Figure 10: Data Distribution

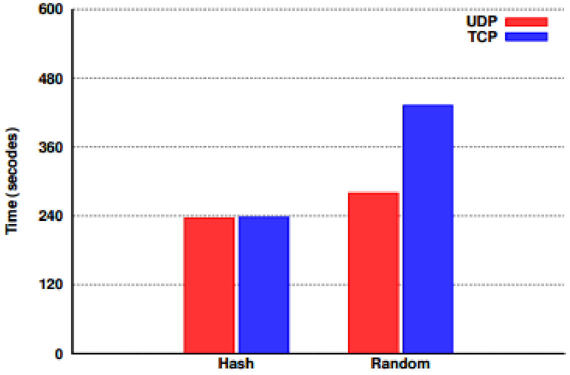
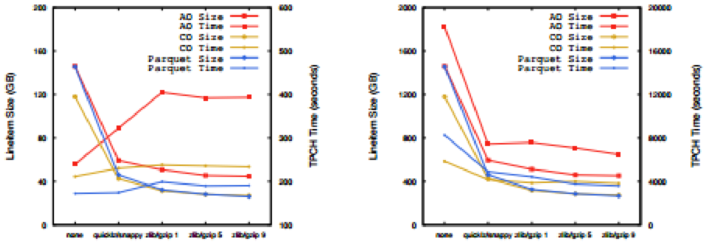


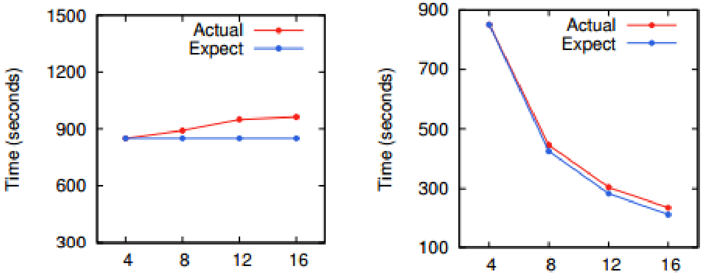
Figure 12: TCP vs. UDP



(a) 160GB dataset.

(b) 1.6TB dataset

Figure 11: Compression



(a) 40GB per node

(b) 160GB per cluster

Figure 13: Scalability

Conclusion: **HAWQ**

- A massively parallel processing SQL engine
- Inherits merits from MPP database and HDFS
- Stateless segment design supported by metadata dispatch and self-described execution plan
- UDP based interconnect to overcome TCP limitations
- Transaction management supported by a swimming lane model and truncate operation in HDFS
- Significant performance advantage over Stinger

Discussion

The master node alone handles transaction, catalog, user interaction, query processing, the final aggregation step of gather motion, etc.

1. Will it be the bottleneck in the system?
2. What are the alternative designs?

The paper claims that PXF allows HAWQ to interact with any data store of choice and run any type of SQL directly on external data sets.

1. What if the format is not compatible with Hadoop data formats or SQL query languages?
2. Is performance and efficiency a valid concern here?