Impala Intro

MingLi
xunzhang

Overview

* MPP SQL Query Engine for Hadoop Environment
* Designed for great performance
* BI Connected(ODBC/JDBC, Kerberos, LDAP, ANSI SQL)
* Hadoop Components
  * HDFS, HBase, Metastore, Yarn, Sentry…
  * Parquet, Avro, RCFile, …
  * Runs on same nodes running Hadoop processes

History

- Developed by Cloudera
- Versions
  - Released as beta in 10/2012
  - 1.0 version available in 05/2013
  - Current version 2.1
- Open Source
  - github.com/cloudera/impala
  - github.com/apache/incubator-impala

Architecture

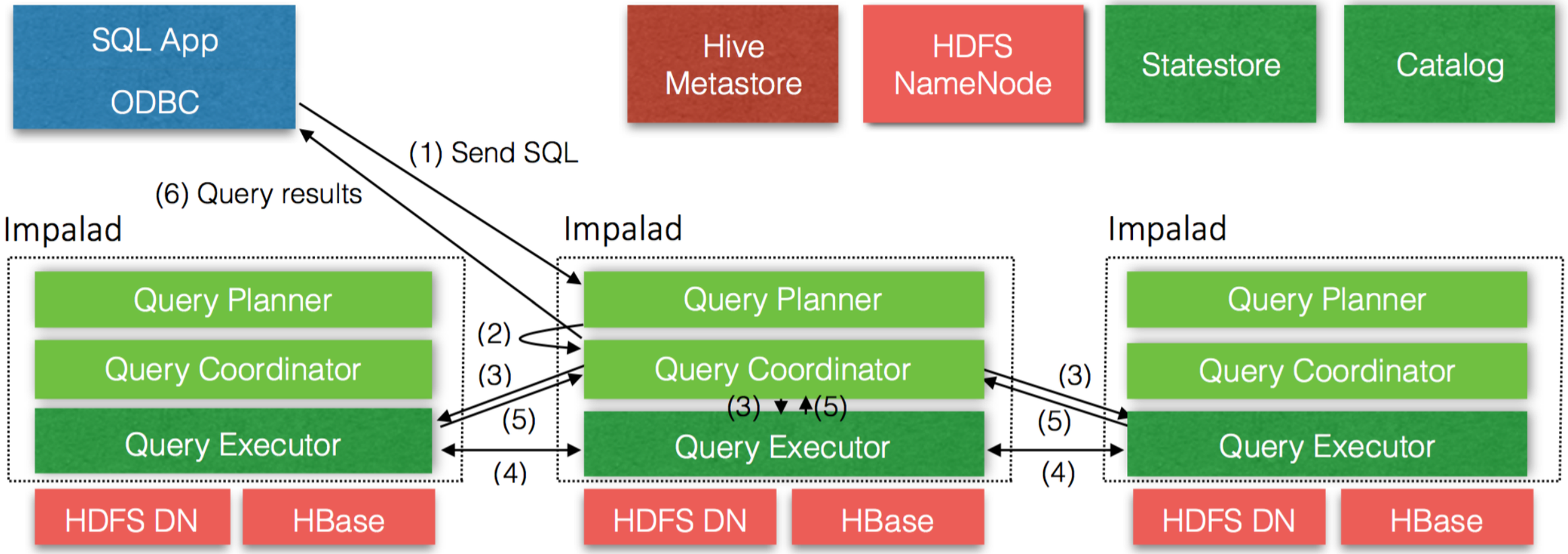
- Impalad: daemon process
- Each node handling requests simultaneously
- Metadata management
  - catalog service(DDL, third-party stores, external metadata)
  - single node
- System state repository and distribution
  - statestore(pub-sub proxy)
  - single node

Impalad

- Frontend(Java)
  - parse queries
  - plan queries
- Backend(C++)
  - coordinate
  - execute plan fragments
- Local cache of metadata
- RPC/Comm
  - thrift

Query Execution

- Query execution phases
  - request via ODBC/JDBC
  - create plan fragments
  - coordinator initiates execution
- During execution
  - Intermediate results are streamed between executors
  - query results are streamed back to client
  - blocking operators
    - top-n, aggregation, sorting

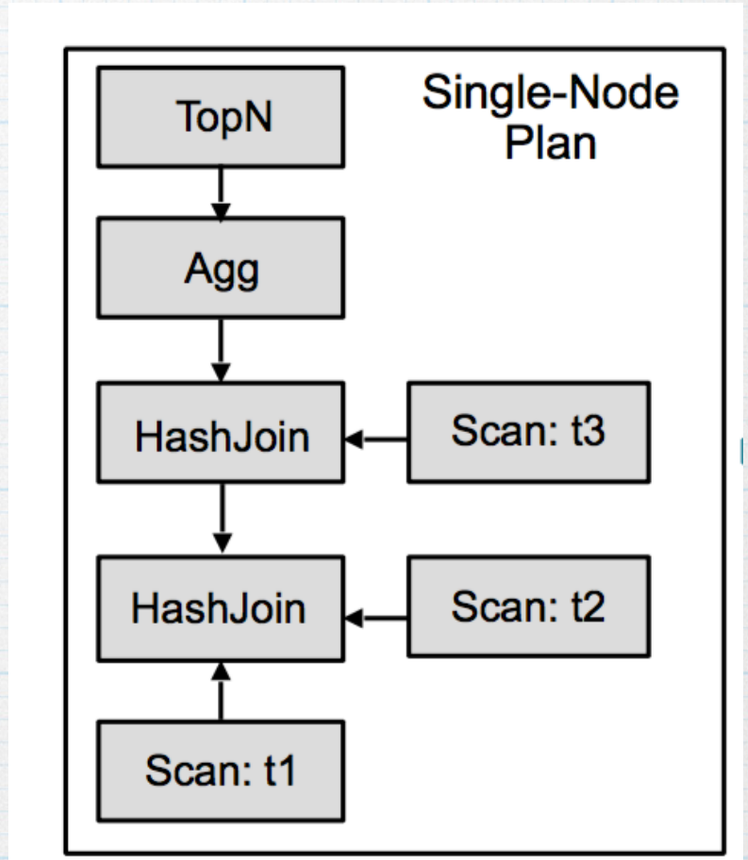

SQL App
ODBC
Hive Metastore
HDFS NameNode
Statestore
Catalog
(1) Send SQL
(6) Query results
Impalad
Query Planner
Query Coordinator
Query Executor
HDFS DN
HBase
(2)
(3)
(4)
(5)

Planning

* 2-phase process
  * I. single-node planning
  * II. plan parallelization and fragmentation

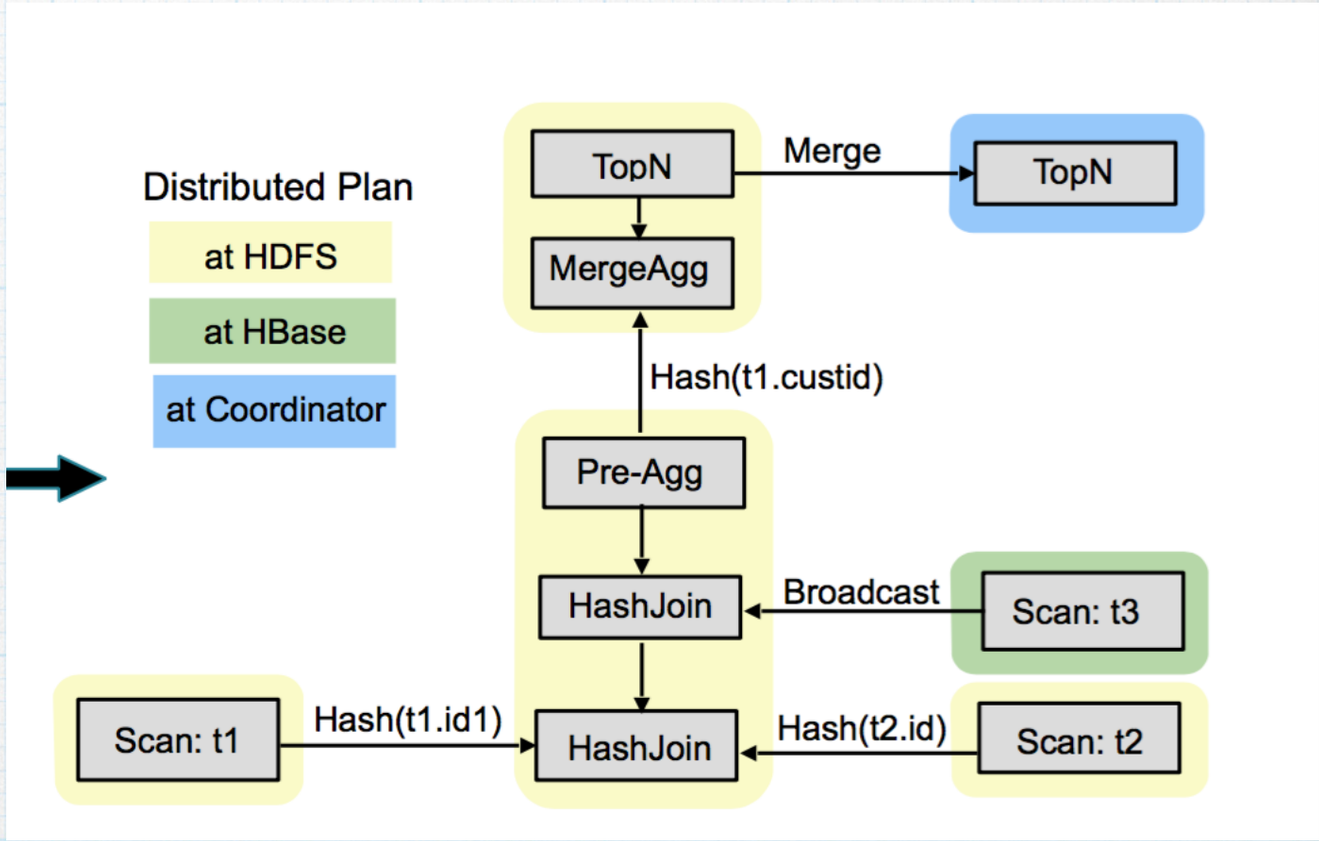
Single-Node plan

```
SELECT t1.custid,
SUM(t2.revenue) AS revenue
FROM t1
JOIN t2 ON (t1.id1 = t2.id)
JOIN t3 ON (t1.id2 = t3.id)
WHERE t3.category = ‘Online’
GROUP BY t1.custid
ORDER BY revenue DESC LIMIT 10;
```



Distributed Plan

- Goals
  - max scan locality
  - min data movement
  - add exchange plan node
  - add non-exchange plan node
- Parallel join
  - broadcast join: large x small
  - partitioned join: large x large
- Split out Fragments
- Cost based decision based on column stats/estimated cost of data transfer

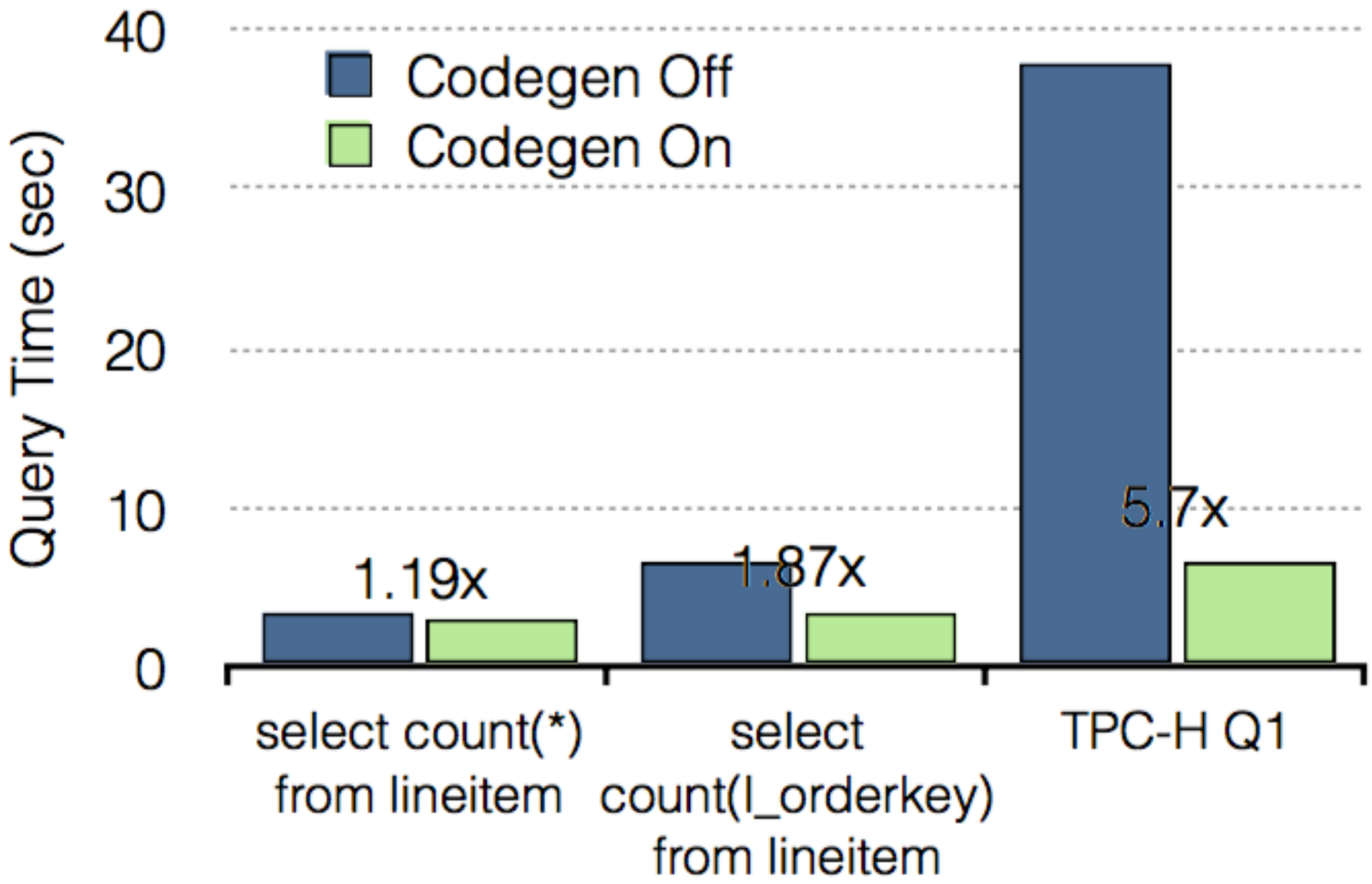


Execution Engine

* Implemented in C++
* Leverages decades DB research
    * partitioned parallelism
    * pipelined relational operators
    * batch at a time runtime
* Focussed on speed and efficiency
    * intrinsics/machine code for text parsing, hashing, etc
    * runtime code generation with LLVM

Runtime Code-gen

* JIT compile
* Effect as custom-coding query
  * remove branches, unroll loops
  * propagate constants, offsets, pointers, etc
  * inline function calls(virtual function)
* Optimized for modern CPUs(pipeline)


Query Time (sec)
40
30
20
10
0
Codegen Off
Codegen On
1.19x
1.87x
5.7x
select count(*) from lineitem
select count(l_orderkey) from lineitem
TPC-H Q1

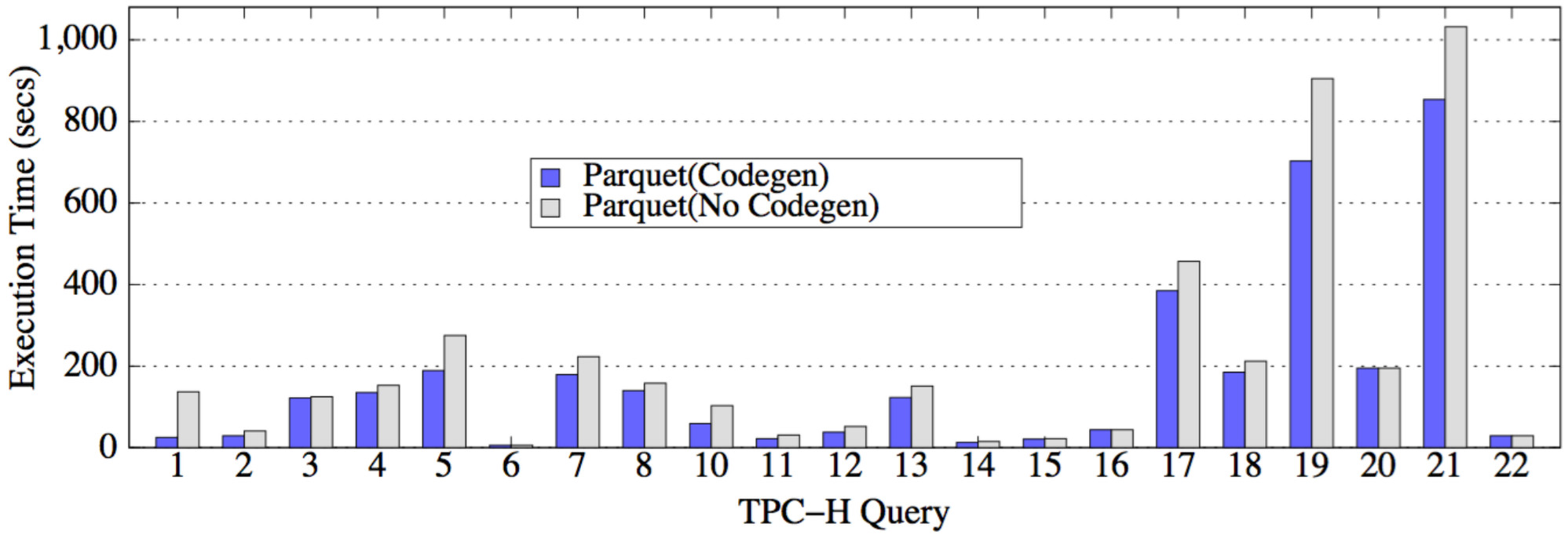


Figure 4: Effectiveness of Impala Code Generation.

1.3x for all the TPC-H queries combined!

I/O Management

* Fixed number of worker threads
    * one I/O thread per rotational disk
    * eight I/O thread per SSD
* Different operators with different method
    * join: single thread
    * aggregation: single thread
    * read threads + scan threads
* HDFS short-circuit local reads from DN
* HDFS centeralized caching for small FACT table(no copy data blocks?)

Case Study I

```
/* TPC_H Query 1 - Pricing Summary Report */
SELECT L_RETURNFLAG, L_LINESTATUS, SUM(L_QUANTITY) AS SUM_QTY,
 SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS SUM_DISC_PRICE,
 SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE, AVG(L_QUANTITY) AS AVG_QTY,
 AVG(L_EXTENDEDPRICE) AS AVG_PRICE, AVG(L_DISCOUNT) AS AVG_DISC, COUNT(*) AS COUNT_ORDER
FROM LINEITEM
WHERE L_SHIPDATE <= dateadd(dd, -90, cast('1998-12-01' as date))
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG,L_LINESTATUS
```

- Fragment 2
  - scan **LINEITEM**
  - apply predicate
  - perform partial aggregation
  - shuffle across all the nodes
- Fragment 1
  - merge partial aggregates(generate 1 row of data in 4 nodes)
- Fragment 0
  - merge all partial aggregates
  - sort results

Case Study I

Table 3: Time Breakdown for Query 1 in Hive-MR

	TXT (secs)	ORC (secs)	ORC Snappy (secs)
MR1-map phase	485	215	176
MR1-reduce phase	5	7	4
MR2	10	11	11

Table 4: Time Breakdown for Query 1 in Impala

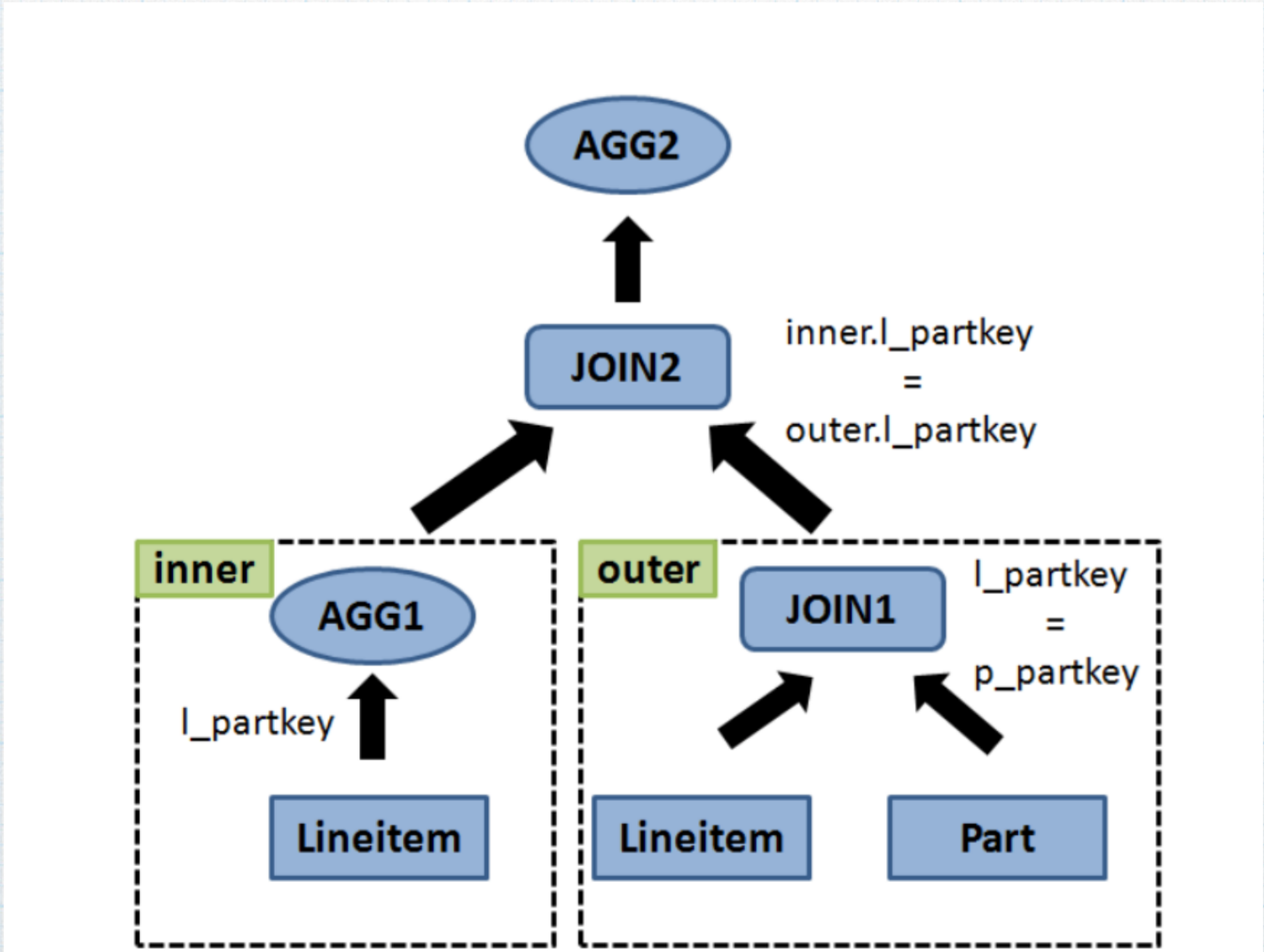
	TXT (secs)	Parquet (secs)	Parquet Snappy (secs)
F1	2	3	3
F2	71	22	21

- launch mappers and reducers
- CPU-bound(deserialization)
- disk utilisation <20%
- column format reduce deser amount

- multi-threads: read threads + scan threads
- Parquet(196GB) | ORC(223GB)
- code-gen for aggregation
- CPU-bound
- disk utilization ~ 85%

Case Study II

```
/* TPC_H Query 17 - Small-Quantity-Order Revenue */
SELECT SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY FROM LINEITEM, PART
WHERE P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#23' AND P_CONTAINER = 'MED BOX'
AND L_QUANTITY < (SELECT 0.2*AVG(L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY = P_PARTKEY)
```



Case Study II

Table 5: Time Breakdown for Query 17 in Hive-MR

	TXT (secs)	ORC (secs)	ORC Snappy (secs)
MR1-map phase	720	534	590
MR1-reduce phase	278	204	201
MR2	12	11	11

Table 6: Time Breakdown for Query 17 in Impala

	TXT (secs)	Parquet (secs)	Parquet Snappy (secs)
F0	1	1	1
F1	33	24	24
F2	109	113	120
F3	249	240	248
F4	1	1	1

* correlation optimization

* scan twice
* single-thread aggregation

More Results

Table 1: Hive vs. Impala Execution Time

TPC-H Query	Hive - MR			Hive - Tez			Impala			ORC-Snappy (Hive-Tez) over Parquet-Snappy (Impala)
	TXT (secs)	ORC (secs)	ORC Snappy (secs)	TXT (secs)	ORC (secs)	ORC Snappy (secs)	TXT (secs)	Parquet (secs)	Parquet Snappy (secs)	
Q1	535	266	228	232	154	172	73	25	24	7.2
Q2	293	299	294	130	132	111	36	29	29	3.8
Q3	761	543	462	524	286	280	149	122	115	2.4
Q4	534	277	232	272	220	214	152	135	147	1.5
Q5	1078	833	772	831	415	409	200	189	190	2.2
Q6	357	109	85	114	88	81	15	6	6	13.5
Q7	1409	1096	1038	581	597	589	195	179	182	3.2
Q8	1085	822	770	743	392	369	155	140	148	2.5
Q9	3371	2883	2848	2102	1682	1692	FAILED	FAILED	FAILED	–
Q10	778	526	502	336	236	224	117	59	50	4.5
Q11	235	234	216	158	141	134	30	22	22	6.1
Q12	555	308	241	226	186	184	37	38	40	4.6
Q13	255	266	228	176	155	156	119	123	127	1.2
Q14	417	175	155	124	123	120	17	13	14	8.6
Q15	619	272	233	166	159	156	27	21	22	7.1
Q16	247	235	239	152	143	133	39	44	43	3.1
Q17	1041	779	813	1205	945	724	401	385	395	1.8
Q18	1183	925	859	1128	723	672	259	185	189	3.6
Q19	907	677	656	919	734	569	742	703	710	0.8
Q20	663	379	332	236	179	175	196	195	191	0.9
Q21	2230	1605	1495	1688	FAILED	FAILED	879	854	857	–
Q22	363	385	452	123	132	130	50	29	30	4.3
AM	860	632	598	553	–	–	–	–	–	–
GM	665	459	422	358	–	–	–	–	–	–
AM-Q{9,21}	666	470	440	419	307	280	150	132	134	2.1
GM-Q{9,21}	577	393	360	303	238	225	90	70	70	3.2

Roadmap

- More SQL
  - ROLLUP/GROUPING SETS
  - INTERSECT/MINUS
  - MERGE
- Nested data structures
  - structs, arrays, maps in Parquet, Avro, Json
- Additional data types: DATA, TIME, DATETIME
- Multithread execution past scan
- Automated Data Conversion
- Support S3-backed tables, SAN-based Systems
- Improved query planning(more elaborate statistics, histograms)

Potential Pros

* MPP
* Parquet format data skiping
* Code generate
* I/O bound queries with great performance
* Shared-nothing database architecture
* Adjust resource consumption estimates during execution, accumulating resource allocation

Potential Cons

* MPP
* No ORC(lightweight index)
* Executors bundled
* Can not recover from mid-query failure
* Single thread execution of joins and aggregations
  * CPU bound queries
  * complex queries

HAWQ vs Impala

* Multi-threads vs Virtual Segments
* Shared-nothing vs Master-slave
    * multiply user performance
* Libhdfs3 vs Libhdfs(JNI)
* Resource Management Strategy(TBD)

Related Discussion

* [HAWQ-303] - Index support for non-heap tables
* [HAWQ-312] - Multiple active master support
* [HAWQ-319] - REST API for HAWQ
* [HAWQ-786] - Framework to support pluggable formats and file systems
* [HAWQ-864] - Support ORC as a native file format

Reference

Impala: A Modern, Open-Source SQL Engine for Hadoop. CIDR15.

SQL-on- Hadoop: Full circle back to shared-nothing database architectures. VLDB14.

The Truth

https://www.youtube.com/watch?v=tHxZsM8N0HQ