Research Cloudera Impala 1.0

May 14th 2013

김민우 (Minwoo Kim)
michael.kim@nexr.com

kt NexR

Introduction of Impala

- Google Dremel, VLDB 2010
- 전통적인 MapReduce 배치 처리를 보완하고 Hadoop에서 Real time ad-hoc query를 가능하게 할 기술적인 비전에 대한 영감을 얻음

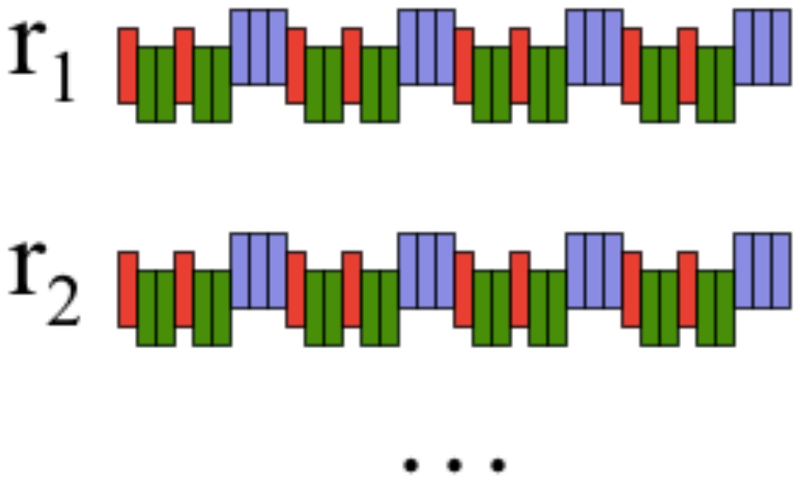
Dremel System

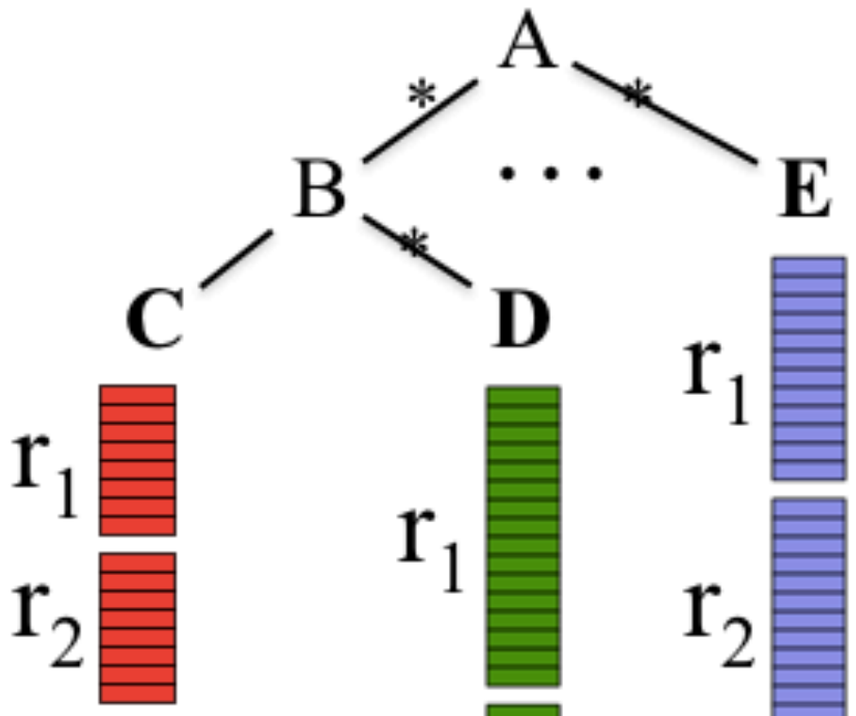
- 레코드 1조 개
- 수 테라바이트 데이터셋에서 인터렉티브 스피드
  - 수천대 노드 스케일
  - 장애 및 낙오자(straggler) 방지 실행
- 중첩 데이터 모델
  - 복합 데이터셋 - 정규화가 금지됨
  - 컬럼기반 저장 및 처리
- 트리 아키텍처
- 구글 데이터 관리 도구로 상호 운영
  - In situ (= in place) 데이터 접근 (e.g GFS, Bigtable)
  - 맵리듀스 파이프라인

Key Aspects of Dremel

- 중첩 데이터 컬럼기반 저장 포맷
  - 어떻게하면 빠르게 중첩 데이터를 컬럼 별로 직렬화/역직렬화 할 것인가?
  - Repetition / Definition level 을 정의
  - 각 컬럼의 Repetition & Definition level 구하여 값과 함께 저장
  - 역직렬화시 위에 level 값으로 FSM를 태워 역직렬화
- SQL & Multi-level serving Tree
  - 쿼리를 완전 병렬화 시켜서 트리의 리프 노드에서 루트 노드로 중간 결과를 스트리밍함
  - Aggregation만 논문에서 언급됨

Records vs. columns





더 적게 읽고
더 빨리
압축을 푼다

Nested data model

multiplicity:

```
message Document {
  required int64 DocId;                [1,1]
  optional group Links {
    repeated int64 Backward;           [0,*]
    repeated int64 Forward;
  }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;         [0,1]
    }
    optional string Url;
  }
}
```

```
DocId: 10
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

```
DocId: 20
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```


Dremel: Interactive Analysis of Web-Scale Datasets. VLDB'10

Repetition and definition levels

Name.Language.Code

value	r	d
en-us	0	2
en	2	2
NULL	1	1
en-gb	1	2
NULL	0	1

레벨 0 은 새 래코드의 시작을 나타냄

```
DocId: 10                    r1
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

r: At what repeated field in the field's path the value has repeated

d: How many fields in paths that could be undefined (opt. or rep.) are actually present

```
DocId: 20                    r2
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

Column-striped representation

DocId

value	r	d
10	0	0
20	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1
http://C	0	2

Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2
80	0	2

Links.Backward

value	r	d
NULL	0	1
10	0	2
30	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	2	2
NULL	1	1
en-gb	1	2
NULL	0	1

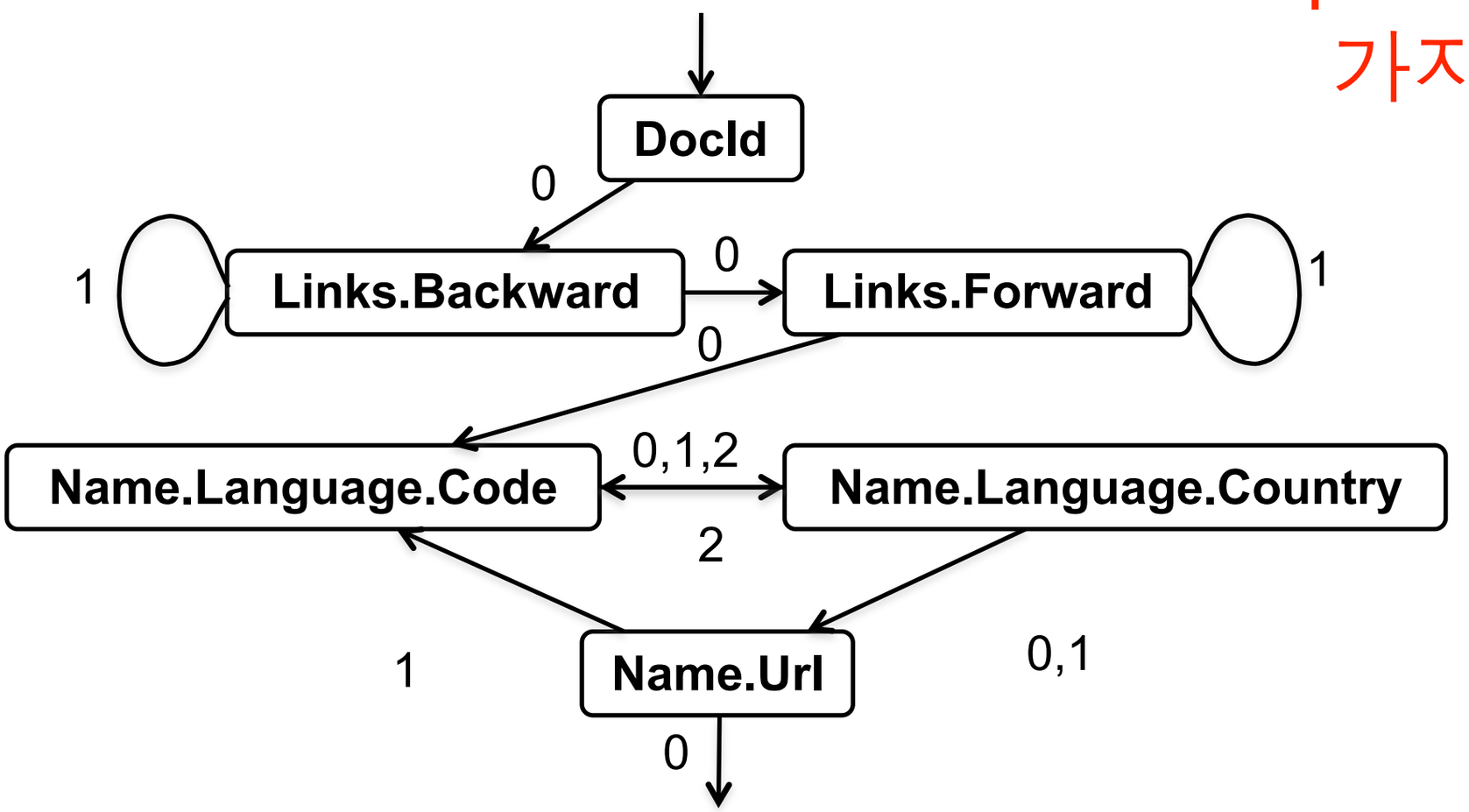
Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NULL	1	1
gb	1	3
NULL	0	1

Dremel: Interactive Analysis of Web-Scale Datasets. VLDB'10

Record assembly FSM

repetition level 을
가지고 변환

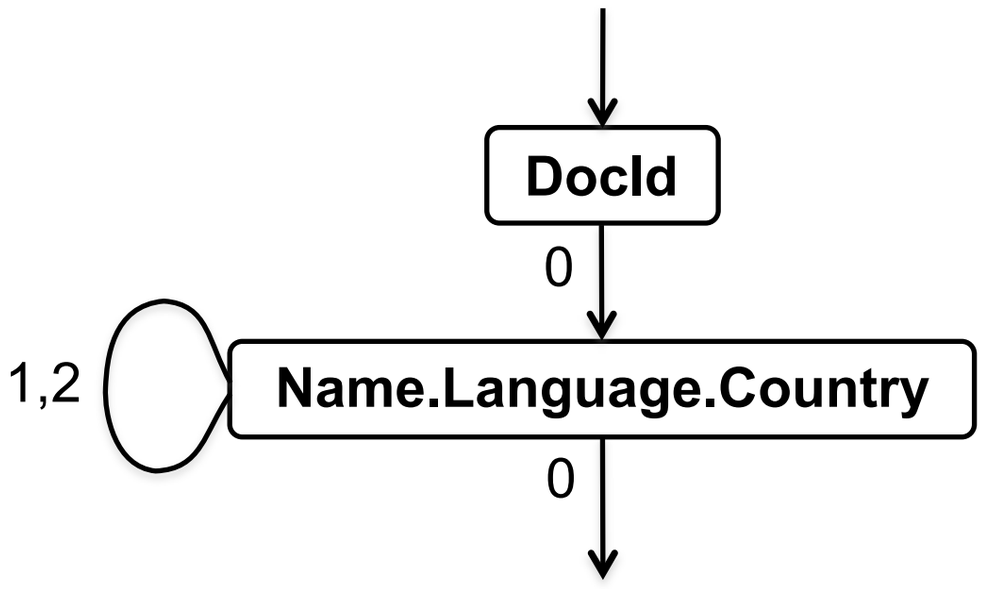


레코드 기반 데이터 처리를 위해서 (e.g. MapReduce)

kt NexR

Dremel: Interactive Analysis of Web-Scale Datasets. VLDB'10

Reading two fields




```
DocId: 10                    S1
Name
  Language
    Country: 'us'
  Language
Name
Name
  Language
    Country: 'gb'
```

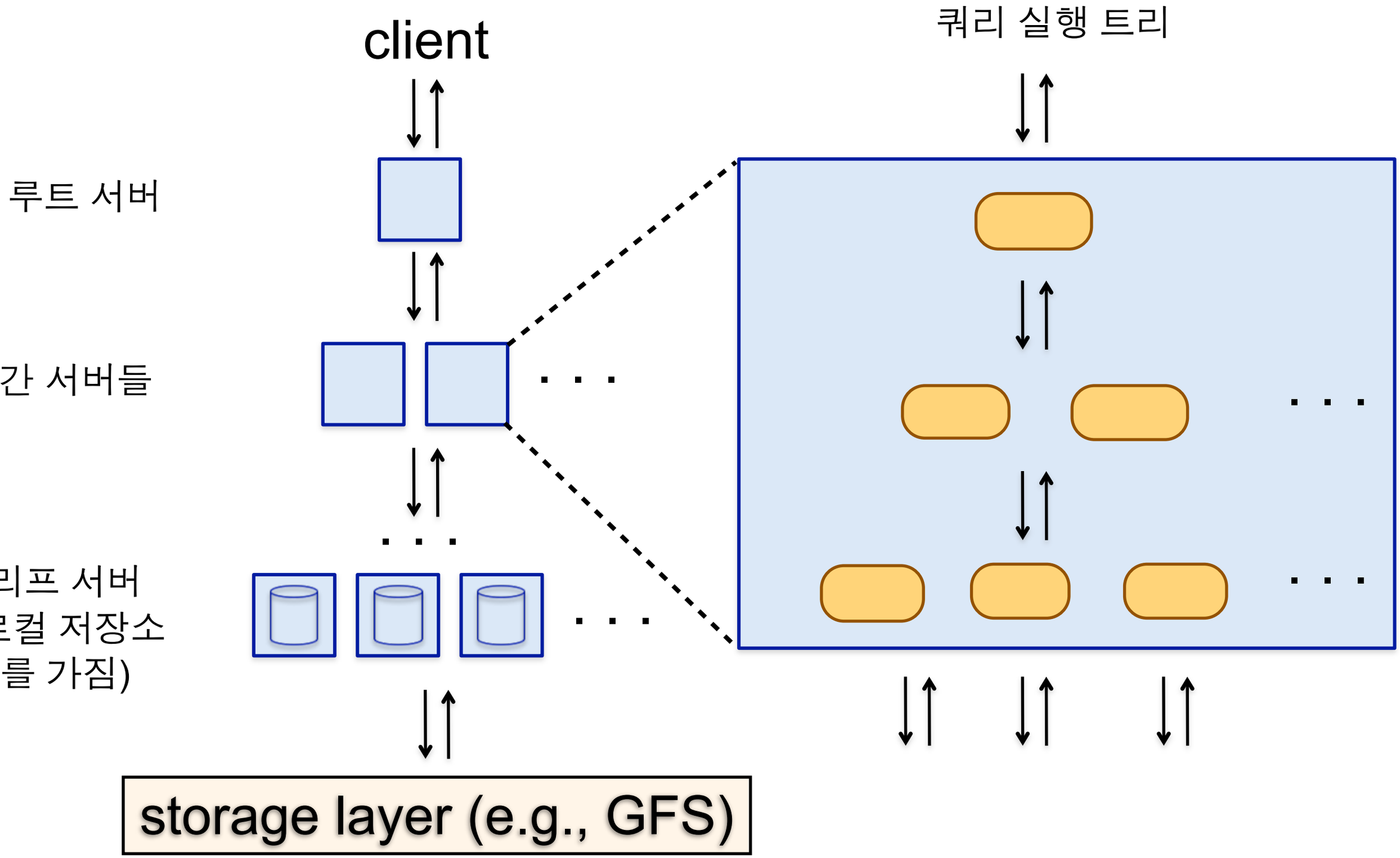
```
DocId: 20                    S2
Name
```

- 부모 필드의 구조가 보존됨
- Name[3] / Language[1] / Country 와 같은 쿼리가 유용하다

Query Processing

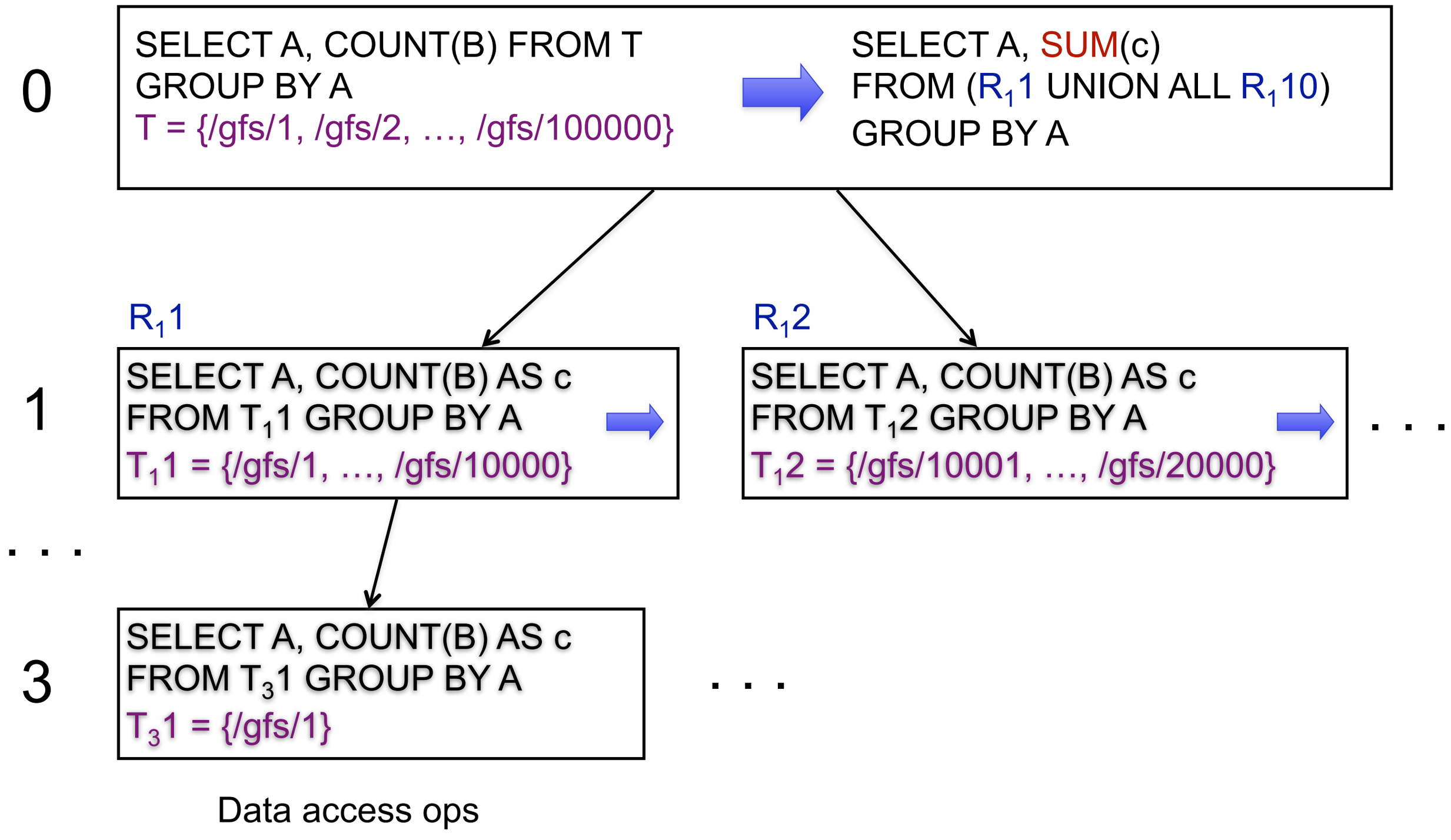
- select-project-aggregate에 최적화됨
  - 단일 스캔
  - 레코드 안에서 aggregation
  - 레코드 사이 aggregation
- 조인, 임시 테이블, UDFs/TVFs 등
  - 논문에서 언급 안됨

Multi-level Serving tree



Dremel: Interactive Analysis of Web-Scale Datasets. VLDB'10

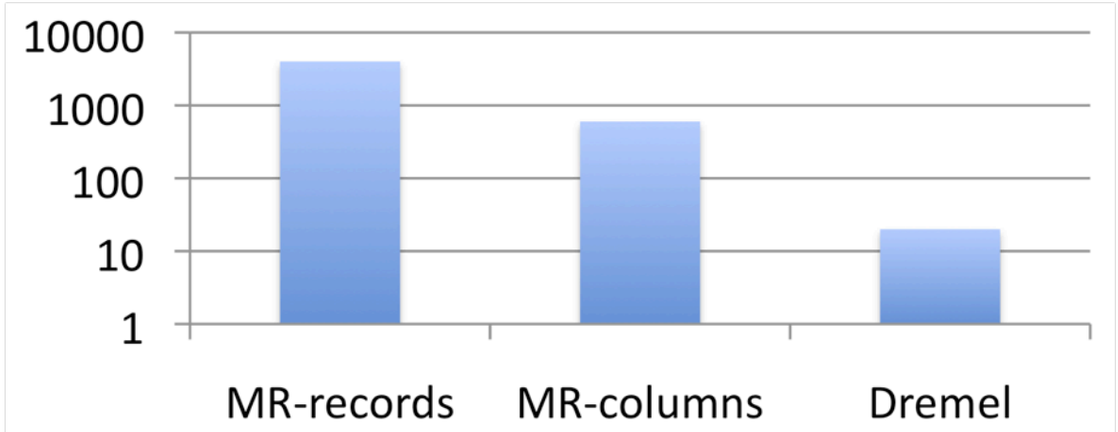
Example: count()



Experiments

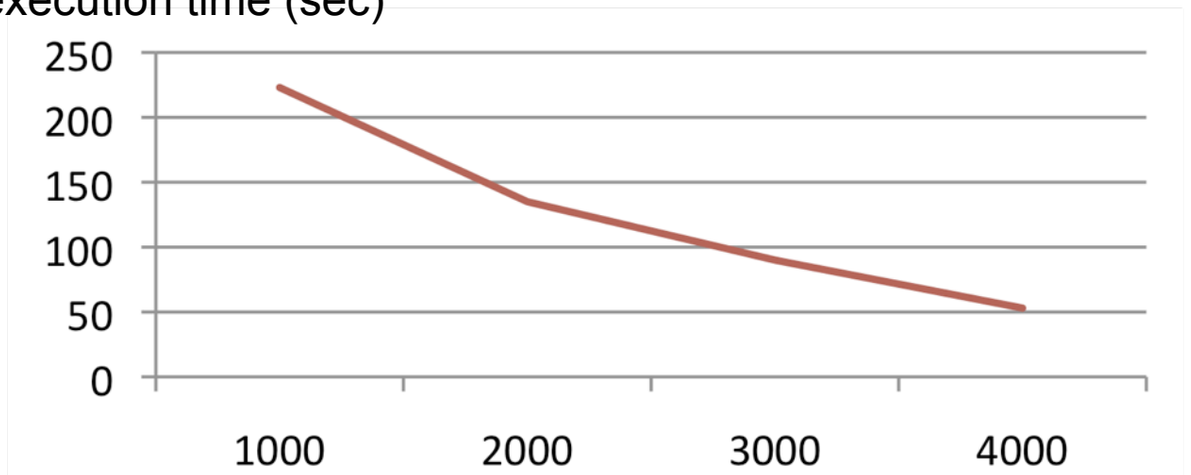
MR and Dremel execution

execution time (sec) on **3000 nodes**



Scalability

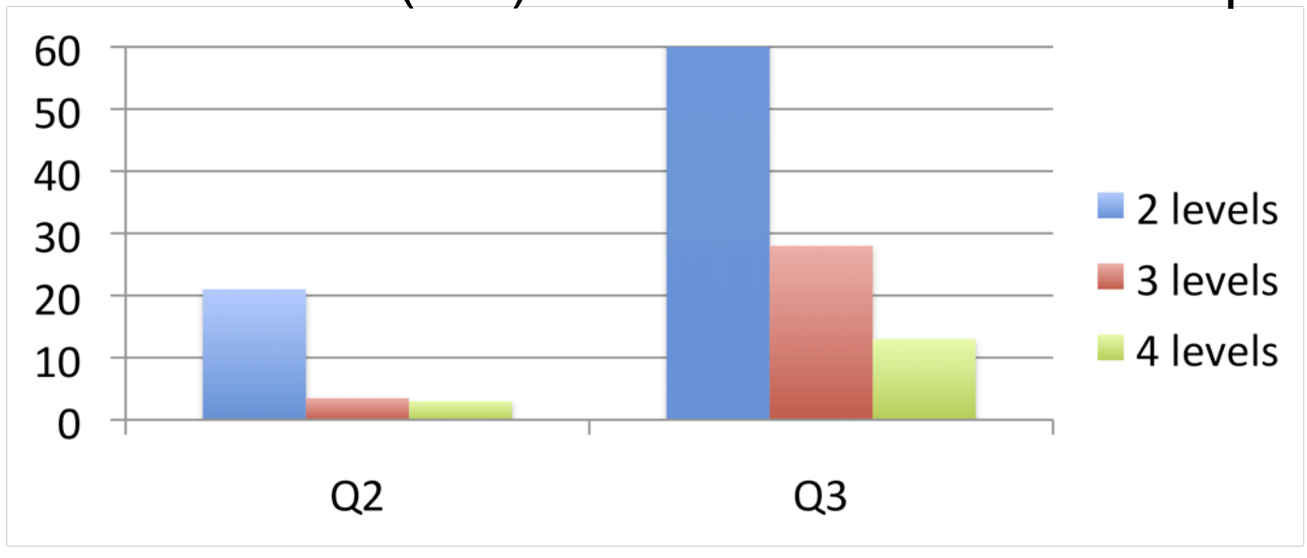
execution time (sec)



number of leaf servers

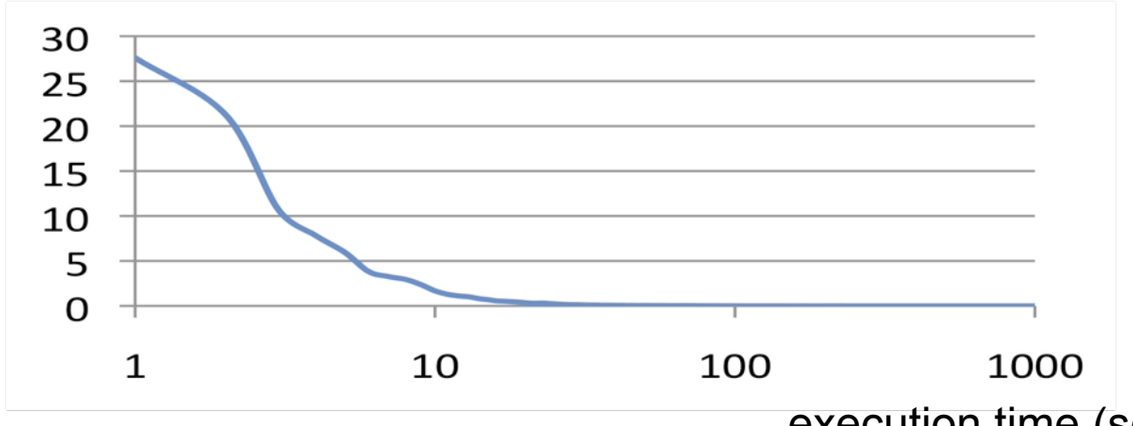
Impact of serving tree depth

execution time (sec)



Interactive speed

percentage of queries



execution time (sec)

Most queries complete under 10 sec

Impala Release

- 릴리즈
  - 1.0 - 30/Apr/13
  - 0.7 - 15/Apr/13
  - 0.6 - 26/Feb/13
  - 0.5 - 04/Feb/13
  - 0.4 - 20/Jan/13
  - 0.3 - 04/Dec/12
  - 0.2 - 13/Nov/12
  - 0.1 - 24/Oct/12
- 한달 단위로 릴리즈

kt NexR

Development

- C++ 로 개발 (기타 shell script + python)
- 임팔라 유저 그룹스
  - https://groups.google.com/a/cloudera.org/forum/?fromgroups#!forum/impala-user
- 지라
  - https://issues.cloudera.org/browse/IMPALA
- github
  - https://github.com/cloudera/impala
- 유저 메일링만 있고 개발자 메일링이 없음
- 사실상 소스만 공개되어 있는 클라우데라 내부 프로젝트라고 볼 수 있음
- 개발인원 - 약 10명
  - 리더 : Marcel Kornacker (전 google)

System Requirement

- OS
  - RHEL 5.7 / 6.2
  - Centos 5.7 / 6.2
  - SLES 11 서비스팩 1 +
  - Ubuntu 10.04 / 12.04
  - Debian 6.03
- CHD 4.1 이상
- Hive
- MySQL, PostgreSQL for 하이브 메타스토어

Impala Benefits

- 친숙한 SQL 인터페이스 제공
- 하둡에 위에 빅 데이터에 인터렉티브한 질의 가능
- 데이터 처리를 하나의 시스템에서 처리해서 모델링이나 ETL 을 피할 수 있다

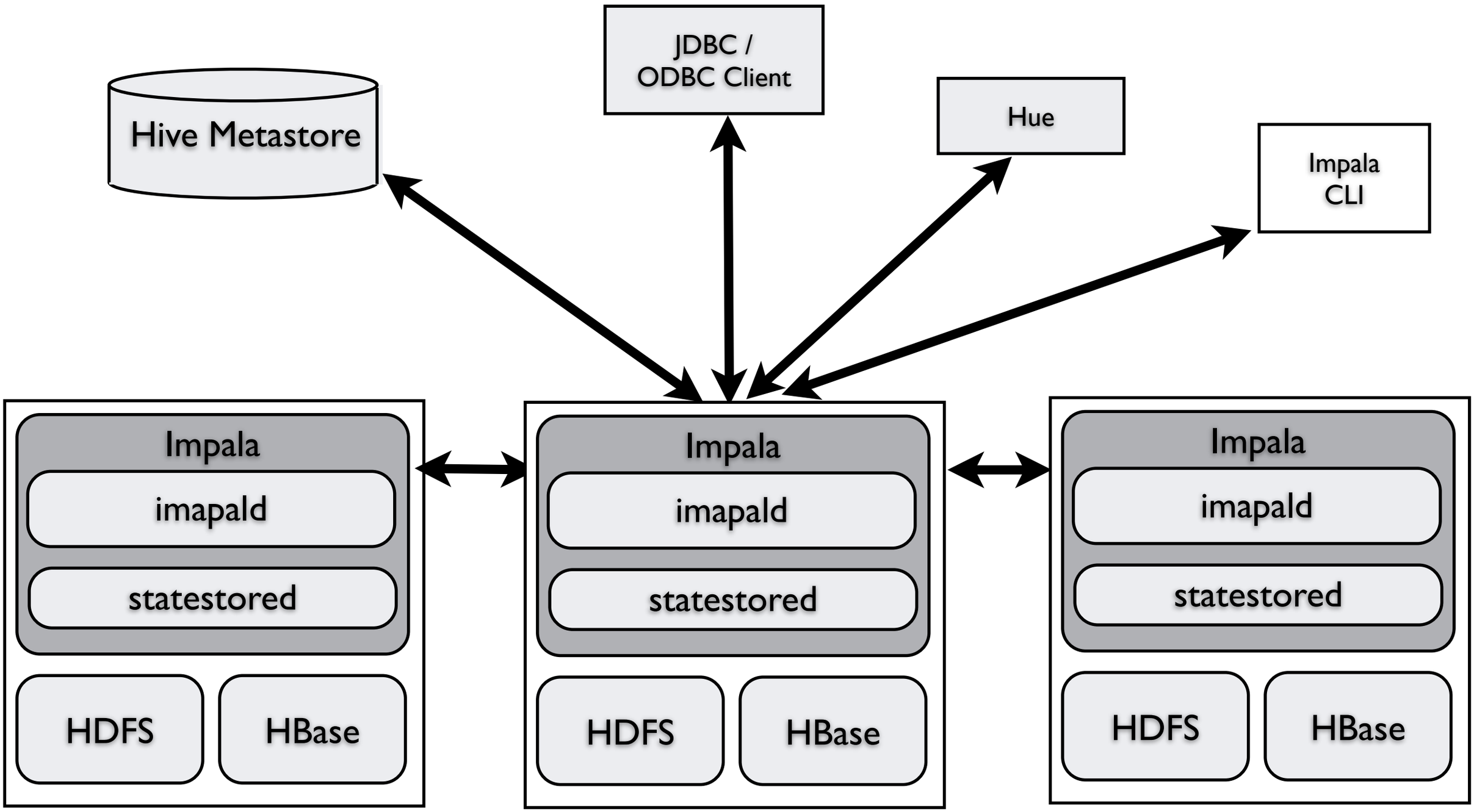
Impala Overview : Goals

- 다목적의 SQL 쿼리 엔진
  - 분석적이고 트랜잭션한 작업부하를 모두 처리해야 한다
  - 수 마이크로초에서 수 시간정도 걸리는 쿼리들을 지원
- 하둡안에서 직접 실행한다
  - 하둡 파일 포맷을 사용하여 읽음
  - 하둡 스토리지 매니저를 사용
  - 하둡 프로세스를 띄우는 같은 노드에서 실행됨
- 고성능
  - JAVA 대신 C++
  - Select, Projection, Join, Union, 서브쿼리
  - 맵리듀스 위에 올라가지 않는 완전 새로운 실행 엔진

User view of Impala : Overview

- 클러스터 안에서 분산 서비스로 실행된다 : 각각 노드에 하나의 임팔라 데몬이 뜬다
- JDBC, ODBC, Hue Beeswax, Impala CLI를 통해서 쿼리를 제출한다
- 쿼리는 적절한 데이터와 함께 모든 노드에 분산된다
- 노드가 실패하면 쿼리는 실패한다
- 임팔라는 하이브의 메타데이터 인터페이스를 사용하고 메타스토어와 연결한다.
- 지원되는 파일 포멧
  - Text File, Sequence File, RCFile, Avro file, Parquet
- 압축 코덱
  - Snappy, GZIP, Deflate, BZIP
- Kerberos 인증

Impala Architecture



Impala Architecture

- Client
  - HUE, ODBC, JDBC, Impala Shell
- Hive Metastore
  - Impala가 이용하는 정보를 저장
  - 예, 어떤 데이터 베이스가 사용가능한지 임팔라에게 알려줌
- Impala (impalad + statestored)
  - 데이터노드 위에서 실행되는 프로세서
  - 쿼리를 실행하고 조정(coordinate)한다
  - Client로 부터의 쿼리를 받고, 실행을 계획하고, 조정한다
- HBase와 HDFS
  - 데이터를 저장

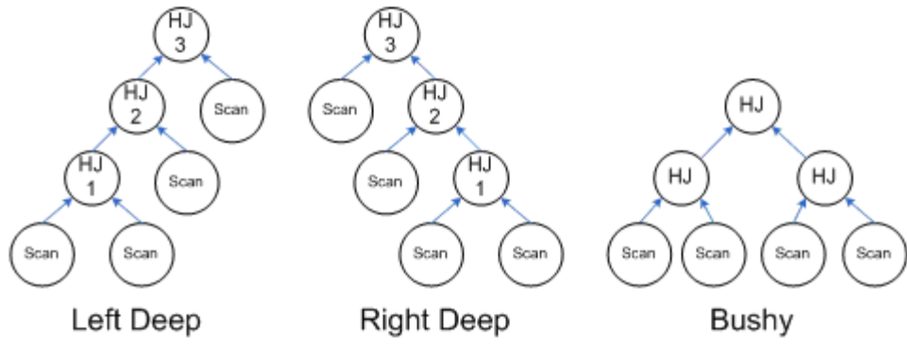
Impala Architecture

- 2 개의 바이너리 : impalad와 statestored
- 임팔라 데몬 (impalad)
  - 쿼리 실행과 관련된 클라이언트 요청과 모든 내부 요청을 다룬다
  - 이 두가지 역할을 Thrift Service로 제공
- 상태 저장 데몬 (statestored)
  - 네임 서비스 제공과 메타데이터 분배
  - Thrift Service로 제공

Impala Architecture

- 쿼리 실행 단계
  - ODBC/JDBC를 통해서 Request 도착
  - Planner가 Request를 Plan 쿼리 오퍼레이터 트리로 변환
  - Coordinator가 원격 impalad의 실행 초기화 시킴
- 실행 중
  - 중간 결과는 Executor사이에 스트리밍 된다
  - 쿼리 최종 결과는 Client로 스트리밍 된다

Query Planning



- 2 단계 플래닝 처리
  - 단일 노드 플랜 : **Left-deep tree plan**
  - 병렬 실행을 위해서 오퍼레이터 트리를 분할한다
- 플랜 오퍼레이터들의 병렬화
  - 모든 쿼리 오퍼레이터들은 완전히 분산됨
  - 조인 : broadcast와 partitioned가 있고 cost based로 결정됨
  - aggregation : fully parallel pre-aggregation과 merge aggregation
  - top-n : 최초 스테이지에서 병렬로 처리됨
- 조인 순서 = FROM 절 순서 (cost-based로 변경 예정)

kt NexR

Query Planning : Example

- 플랜 오퍼레이터들:
  - Scan, HashJoin, HashAggregation, Union, TopN, Exchange

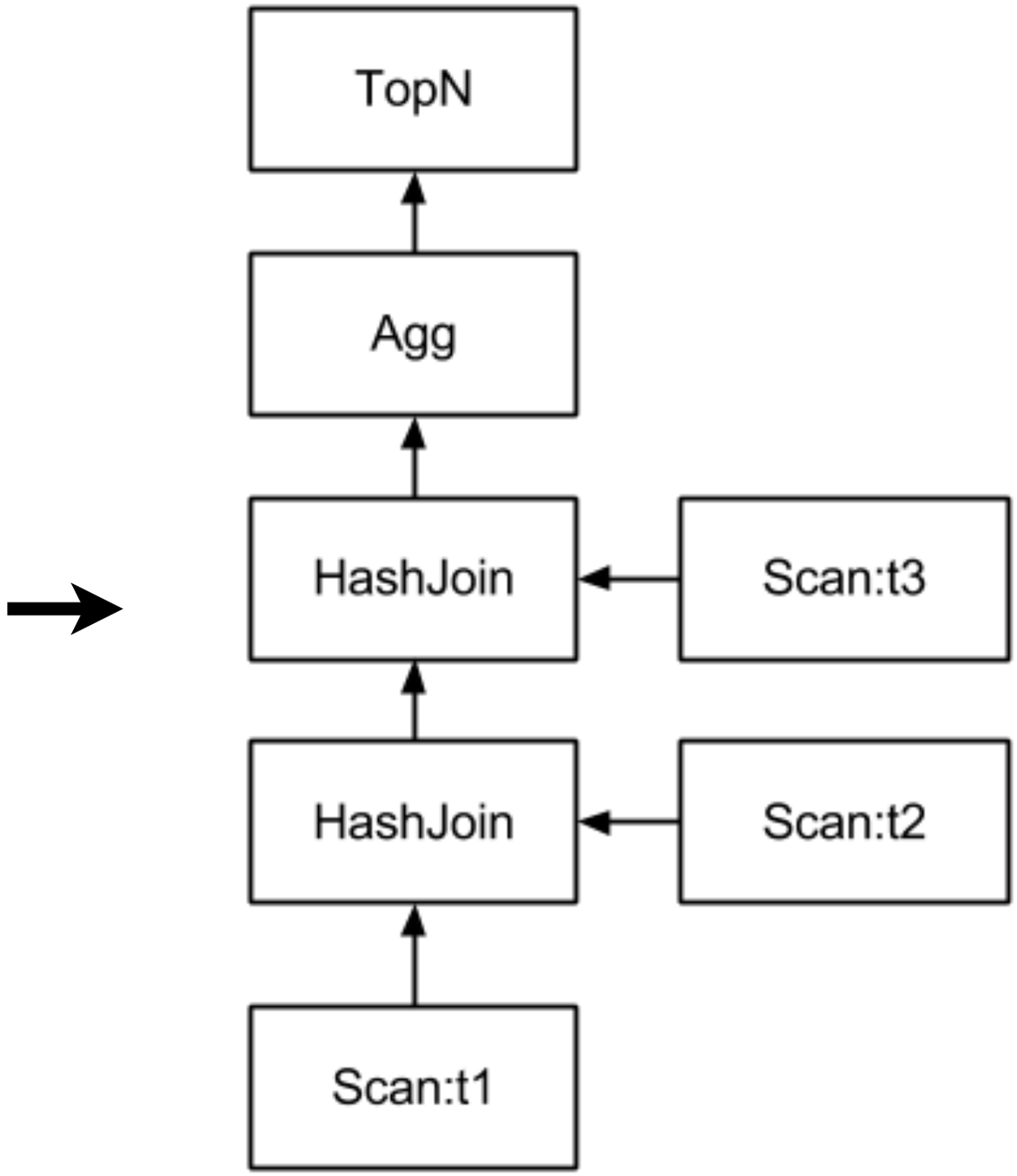
- 예제 쿼리

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

Query Planning : Example

- 단일 노드 플랜:

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



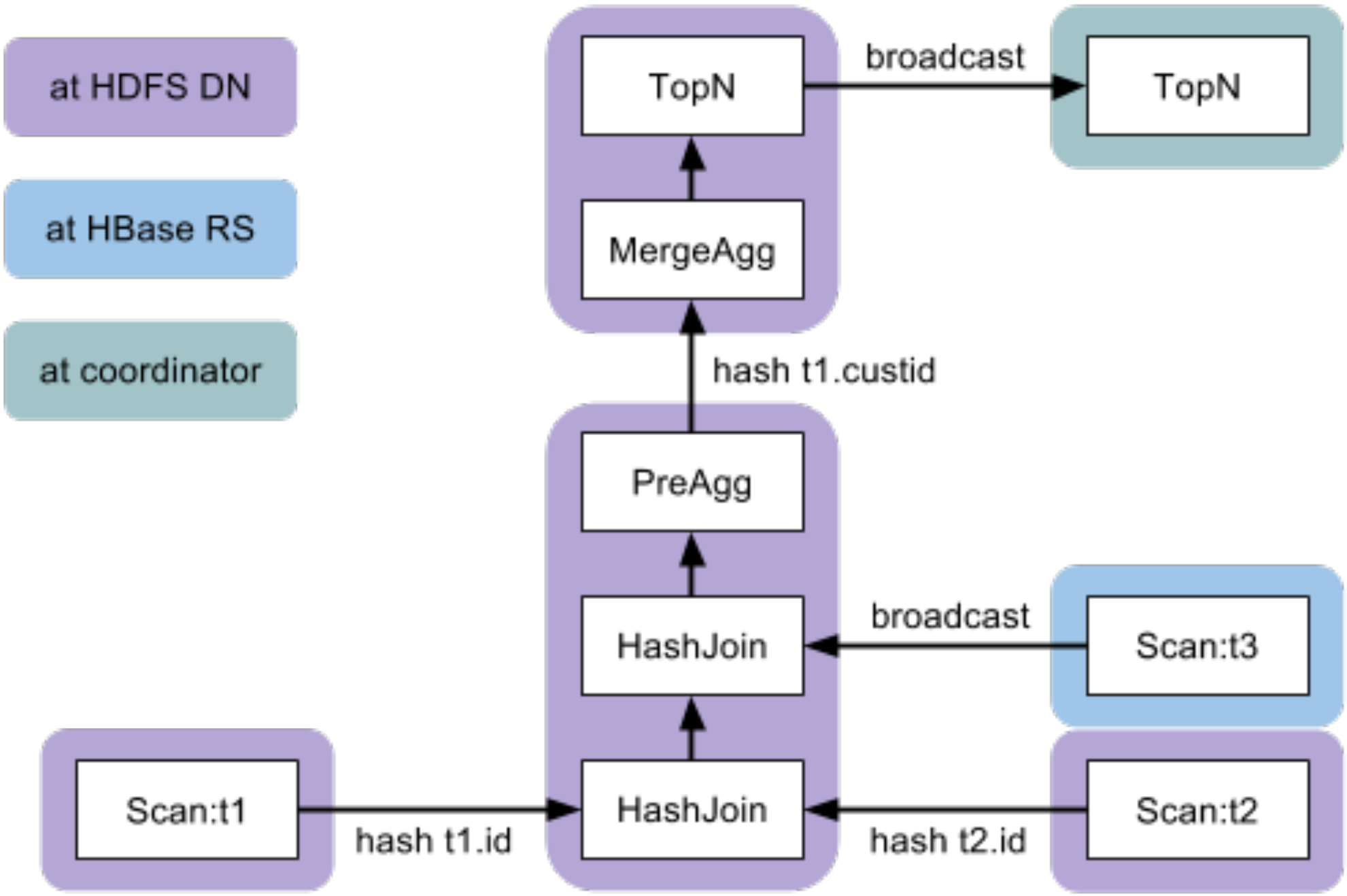
Query Planning : Distributed Plans

- 목표:
  - 지역(locality) 스캔을 극대화, 데이터 이동을 최소화
  - 쿼리 오퍼레이터들을 완전 분배
- 병렬 조인
  - broadcast join :
    - 오른쪽 테이블은 조인을 수행하는 각 노드에 broadcast된다
    - 오른쪽 테이블이 작을 경우에 선호된다.
  - partitioned join
    - 두 테이블 모두 조인 컬럼들로 hash-partitioned 된다
    - 큰 테이블 조인시 선호된다
  - 예측된 데이터 전송 비용과 컬럼 통계기반으로 cost-based 결정을 내린다.

Query Planning : Distributed Plans

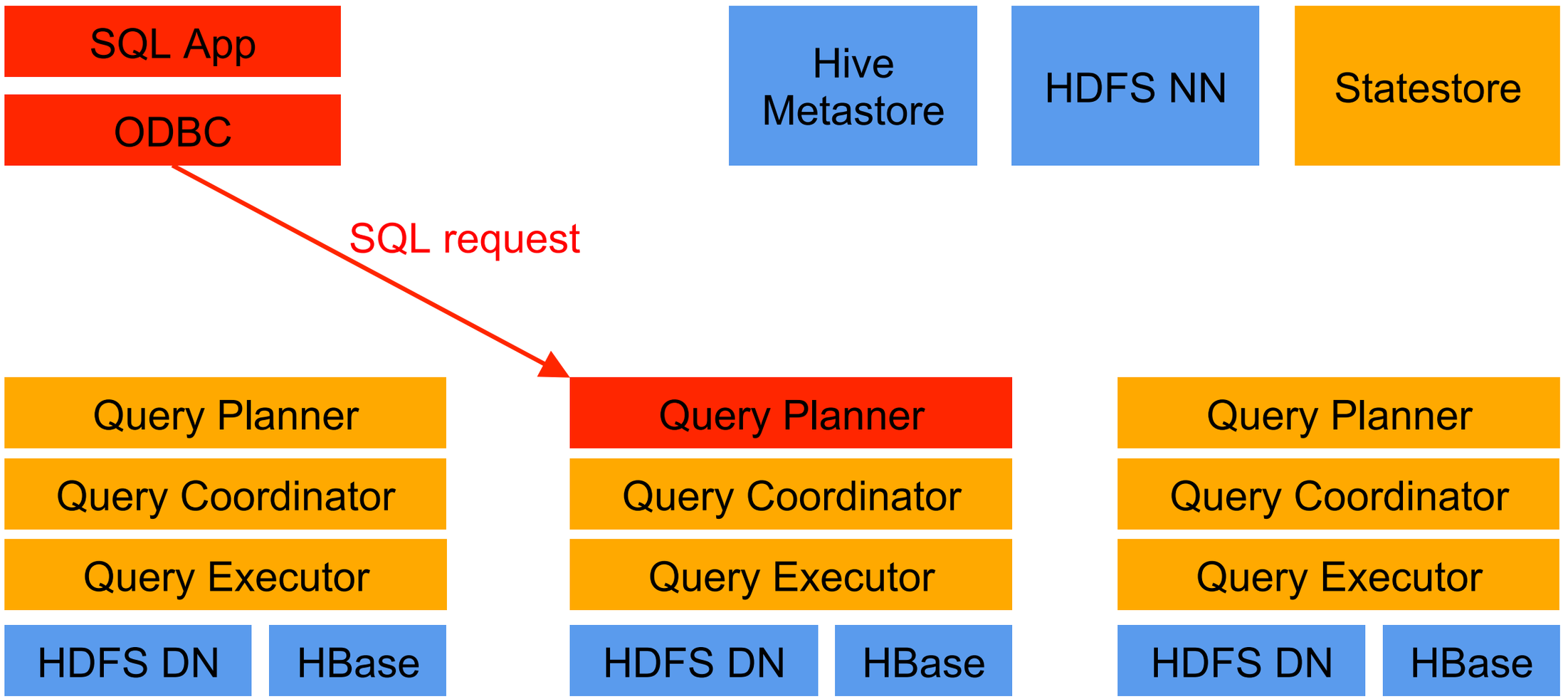
- 병렬 집계
  - 데이터가 처음 물리화된 곳에서 pre-aggregation
  - merge aggregation은 그루핑 컬럼들로 나누어진다
- 병렬 top-N
  - 데이터가 처음 물리화된 곳에서 최초 top-N
  - 단일 노드 플랜 조각에서 마지막 top-N

Query Planning: Distributed Plans



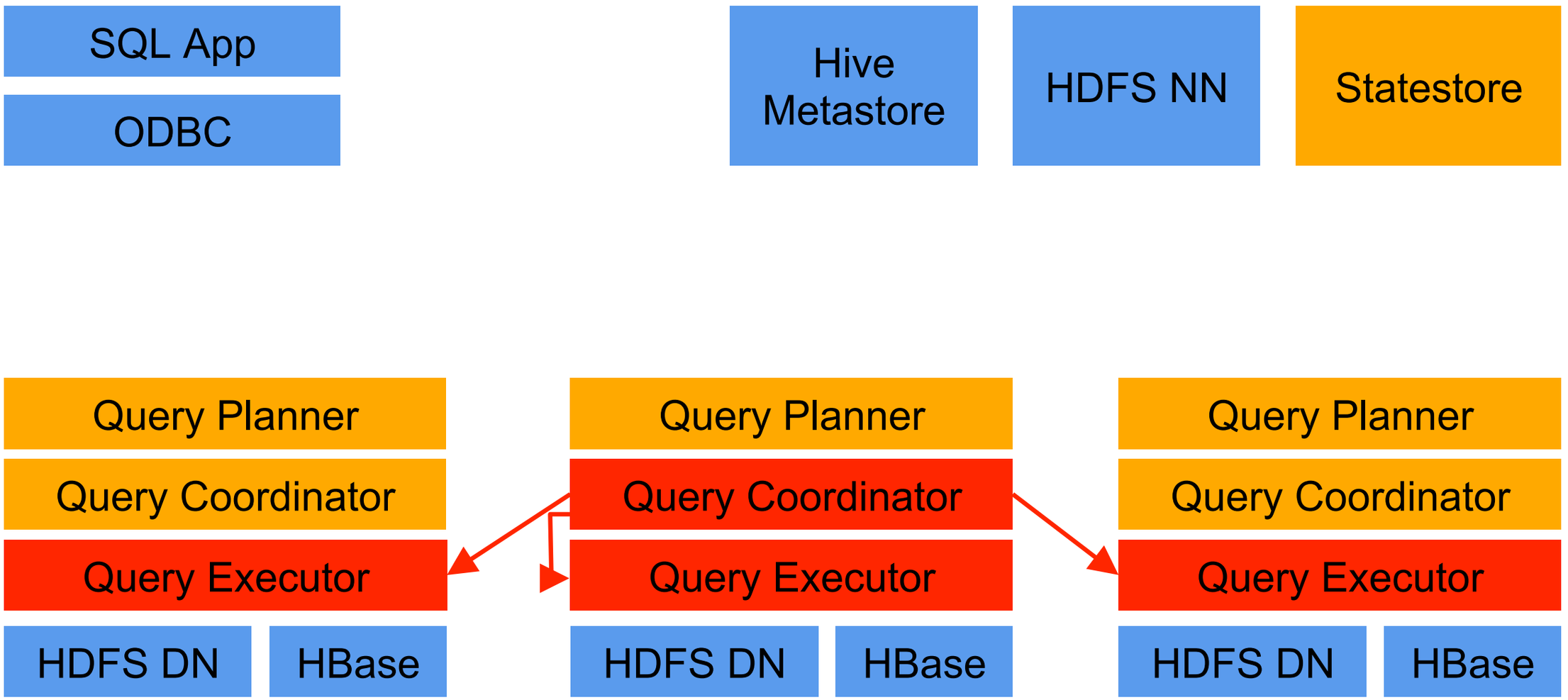
Impala Architecture: Query Execution

- ODBC/JDBC를 통해서 Request 도착



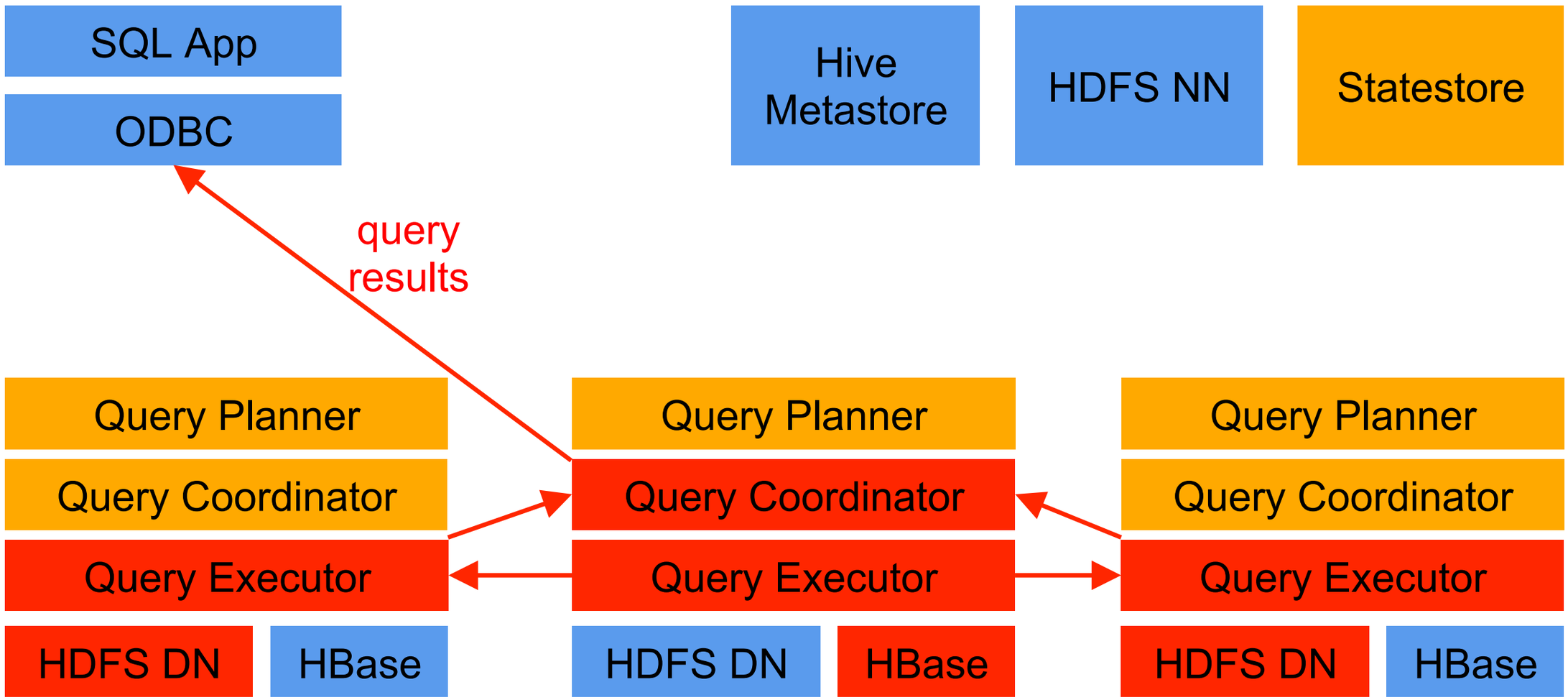
Impala Architecture: Query Execution

- Planner가 Request를 쿼리 연산자 트리 변환
- Coordinator가 원격 impalad의 실행 초기화



Impala Architecture: Query Execution

- 중간 결과는 Executor사이에 스트리밍 된다
- 쿼리 최종 결과는 Client로 스트리밍 된다.



Metadata Handling

- Hive 메타스토어를 이용함
- 메타데이터를 캐시함 : 쿼리 실행 도중에 synchronous 메타스토어 API 호출은 없다
- statestored가 메타데이터를 분배한다

Supported Data Type

- Hive 데이터 원시타입과 동일한 이름과 시멘틱으로 지원
  - tinyint, smallint, int, bigint, float, double, boolean, string, timestamp
- 복합 데이터 타입은 지원하지 않는다
  - struct, map, array

Language Reference

- Impala CREATE TABLE 커맨드, Hive DDL를 사용한 테이블 생성
- HiveQL과 비슷한 DML 지원
  - HiveQL과 동일한 파라메터와 이름의 내장 함수들을 지원
  - JOIN, AGGREGATE, DISTINCT, UNION ALL, ORDER BY, LIMIT, 서브쿼리
  - INSERT INTO, INSERT OVERWRITE

ALTER TABLE

- 테이블 이름 바꾸기
  - ALTER TABLE old_name RENAME TO new_name;
- 테이블 데이터 파일의 물리적인 위치 바꾸기
  - ALTER TABLE table_name SET LOCATION ‘hdfs_path_of_directory’;
- 테이블 컬럼 재배치
  - ALTER TABLE table_name ADD COLUMNS (column_defs);
  - ALTER TABLE table_name REPLACE COLUMNS(column_defs);
  - ALTER TABLE table_name CHANGE column_name new_name new_spec;
  - ALTER TABLE table_name DROP column_name;
- 파일 포맷 변경
  - ALTER TABLE table_name SET FILEFORMAT { PARQUETFILE | RCFILE | SEQUENCEFILE | TEXTFILE }
- 파티션 추가 삭제
  - ALTER TABLE part_table ADD PARTITION (month=concat(‘Decem’,‘ber’));
  - ALTER TABLE part_table DROP PARTITION (month=‘January’);

CREATE DATABASE / TABLE

데이터 베이스 생성

CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
[COMMENT 'database_comment'] [LOCATION hdfs_path];

- HDFS 디렉토리가 존재하지 않으면 생성한다
- 데이터베이스 생성 후 impala-shell 세션에서 기본 데이터베이스로 지정하기 위해서는 USE 구문을 사용한다

CREATE TABLE [LIKE]

테이블 생성 문법

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name data_type [COMMENT 'col_comment'], ...)]
  [COMMENT 'table_comment']
  [PARTITIONED BY (col_name data_type [COMMENT 'col_comment'], ...)]
  [  [ROW FORMAT row_format] [STORED AS file_format]  ]
  [LOCATION 'hdfs_path']

data_type  : primitive_type

primitive_type :
 TINYINT  | SMALLINT  | INT  | BIGINT  | BOOLEAN  | FLOAT  | DOUBLE  | STRING  |  TIMESTAMP

row_format   : DELIMITED [FIELDS TERMINATED BY 'char'] [LINES TERMINATED BY 'char']

file_format  :  PARQUETFILE  | SEQUENCEFILE  | TEXTFILE  | RCFILE
```

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  LIKE [db_name.]table_name  [COMMENT 'table_comment']
  [STORED AS file_format]  [LOCATION 'hdfs_path']
```

DISTINCT Clause

DISTINCT 예제

```
-- Returns the unique values from one column.
SELECT DISTINCT c_birth_country FROM customer;

-- Returns the unique combinations of values from multiple columns.
SELECT DISTINCT c_salutation, c_last_name FROM customer;
```

- 같은 쿼리에서 하나 이상의 DISTINCT 사용은 지원하지 않는다

다음은 지원하지 않음

```
SELECT COUNT(DISTINCT c_first_name) and COUNT(DISTINCT c_last_name) FROM customer;
```

DROP DATABASE / TABLE

데이터베이스 삭제

DROP (DATABASE|SCHEMA) [IF EXISTS] database_name;

테이블 삭제

DROP TABLE [IF EXISTS] table_name

- 기본적으로 Impala는 연관된 HDFS 디렉토리를 삭제한다
- 테이블이 EXTERNAL 절과 함께 생성되었다면 디렉토리는 삭제하지 않는다

GROUP BY

- 지원되는 함수는
  - COUNT(), SUM(), AVG(), MIN(), MAX() 이다.

예제

```
SELECT
  ss_item_sk as Item,
  count(ss_item_sk) as Times_Purchased,
  sum(ss_quantity) as Total_Quantity_Purchased
FROM store_sales
  GROUP BY ss_item_sk
  ORDER BY sum(ss_quantity) desc
LIMIT 5;
+--------+--------------------+-----------------------------+
| item   | times_purchased | total_quantity_purchased |
+--------+--------------------+-----------------------------+
| 9325   |  372               | 19072                       |
| 4279   | 357                | 18501                       |
| 7507   | 371                | 18475                       |
| 5953   | 369                | 18451                       |
| 16753 | 375                | 18446                       |
+--------+--------------------+-----------------------------+
```

JOIN

조인 예제

```
[localhost:21000] > SELECT c_last_name, ca_city FROM customer, customer_address WHERE
c_customer_sk = ca_address_sk LIMIT 10;
+---------------+------------------+
| c_last_name | ca_city            |
+---------------+------------------+
| Lewis         | Fairfield        |
| Moses         | Fairview         |
| Hamilton      | Pleasant Valley  |
| White         | Oak Ridge        |
| Moran         | Glendale         |
| Sharp         | Lakeview         |
| Wiles         | Farmington       |
| Shipman       | Union            |
| Gilbert       | New Hope         |
| Brunson       | Martinsville     |
+---------------+------------------+
Returned 10 row(s) in 0.63s
```

JOIN

- partitioned join
  - 두 테이블 모두 해시 알고리즘을 이용해서 로우의 서브셋을 다른 노드들에게 보낸다
  - 테이블의 인덱스 통계정보를 알 수 없다면 broadcast join으로 처리된다
- broadcast join
  - 오른쪽 테이블의 전체 내용을 조인을 처리에 관련된 모든 노드들에게 보내는 방식
  - 인덱스나 통계정보를 알 수 없을 때 기본값으로 사용됨

- 조인 순서가 Impala 성능에 영향을 미친다.
  - 가장 좋은 방법은 오른쪽에서 부터 큰 테이블 순서로 조인하는 것이다
  - cost-based로 바뀔 예정

INSERT

- 다음 두가지 INSERT를지원한다.
  - INSERT INTO
  - INSERT OVERWRITE
- 현재 Impala는 오직 Text와 Parquet 포맷을 사용하는 테이블에만 삽입을 할 수 있다
- 다른 포맷들은 하이브를 사용해서 데이터를 삽입하고 Impala로 쿼리를 실행한다

LIMIT, ORDER BY

- LIMIT
  - 결과셋의 로우 수를 제한
  - Top-N 쿼리시 사용
  - 특정 쿼리의 샘플 결과 확인 시 사용 (LIMIT절을 사용하면 성능이 대폭 향상 된다)
- ORDER BY
  - 최종 결과 로우들의 순서를 보장
  - LIMIT 절과 같이 써야 성능을 높일 수 있다

REFRESH

문법

REFRESH [table_name]

- Impala에만 있는 구문
- Impala가 쿼리에 신속하게 응답하기 위해서는 데이터베이스와 테이블에 대한 메타데이터를 가지고 있어야 한다
- Hive와 Impala가 공유하는 메타스토어에서 Impala가 사용하는 정보를 수정한다면, Impala는 캐시된 정보가 업데이트 되어야 한다.
- 모든 메타데이터를 업데이트 시킨다

Unsupported Language Elements

- ALTER DDL 요소
- 권한 (Authorization) 관련 요소들
- 복합 데이터 타입 maps, arrays, structs.
- LOAD DATA (로컬에서 데이터 로드)
- TRANSFORM, 커스텀 파일 포맷, 커스텀 SerDe
- XML, JSON 함수
- UDF, UDTF
- 샘플링
- Lateral View
- 쿼리에 다중 DISTINCT 절

kt NexR

Unsupported Language Elements

- SHOW PARTITIONS.
- SHOW TABLE EXTENDED.
- SHOW TBLPROPERTIES.
- SHOW FUNCTIONS.
- SHOW INDEXES.
- SHOW COLUMNS.
- SHOW CREATE TABLE.
- DESCRIBE DATABASE.
- DESCRIBE COLUMN
- IMPORT TABLE and EXPORT TABLE.
- ANALYZE TABLE.

kt NexR

Impala does not

- MapJoin, StreamJoin 힌트 지원하지 않음
- SORT BY, DISTRIBUTED BY, CLUSTER BY
- 가상 컬럼 지원하지 않음
- Lock 지원하지 않음
- 설정 프로퍼티 노출하지 않음
- 컬럼이 오버플로우 되었을 때 NULL을 반환하지 않고 타입의 범위의 최소 또는 최대 값을 반환한다
- local 타임존으로 Timestamp를 저장하지 않고 GMT로 저장된다

Understanding File Formats

File Type	Format	Compression Codes	Impala Can CREATE ?	Impala Can INSERT?
Parquet	Structured	Snappy, GZIP, deflate, BZIP2; Snappy이 기본값	Yes	Yes : CREAT TABLE INSERT, and query
Text	Unstructured	LZO	Yes (...)	
Avro	Structured	Snappy, GZIP, deflate, BZIP2	No, create using Hive	No, query only
RCFILE	Structured	Snappy, GZIP, deflate, BZIP2	Yes	No, query only
SequenceFile	Structured	Snappy, GZIP, deflate, BZIP2	Yes	No, query only

Parquet File Format

- 새로운 컬럼기반 파일 포맷
- 더그 커팅의 Trevni를 계승함
- 클라우데라와 트위터에서 공동 개발
- 구글 Dremel definition/repetition level 기반
- 중첩 구조와 sparse한 데이타
- 컬럼 별 인코딩을 위한 확장성을 지원
  (e.g. indexes, bloom filters, statistics)
- Trevni 보다 더 빠른 메타데이타 쓰기 성능

HDFS Improvements

- HDFS 개선점 4가지

1. 개별 블록 레플리카 디스크 위치 정보를 표시
    - 중복 읽기를 하지 않기 위해서 레플리카의 위치를 알아야 함
    - 네임노드밖에 레플리카의 위치를 알지 못함
    - 새로운 RPC콜을 통해 어떤 데이터 노드에 블록 레플리카가 있는지 정의하고 임팔라가 데이터 노드에 쿼리해 어느 디스크에 있는지 알게함
2. 특정 파일 블록 레플리카를 같은 데이터 노드에 위치시키게 만듬
    - 모든 읽기는 로컬에서 읽는 게 이상적
    - 특정 파일의 레플리카는 같은 노드에 위치 시켜 로컬 읽기를 유도함

HDFS Improvements

3. 자주 쓰는 테이블과 파일을 인메모리에 캐싱

- 디스크 OS의 버퍼 캐시를 사용하는 것보다 임팔라 인메모리 성능을 활용할 때 훨씬 빨라졌기 때문에 HDFS에 인메모리캐싱 기능을 추가해 관리자가 HDFS 파일이나 하이브 테이블을 메인 메모리에서 읽고 저장하게 함

4. HDFS 읽기 작업시 거치는 복제 과정을 줄임

- 기존 : 로컬 -> 메모리 -> 버퍼 -> 커널 -> 네트웍 -> 클라이언트
- 임팔라 데몬이 데이터 노드를 우회해 CPU가 직접 읽게 함
- HDFS에 API를 추가해서 직접 바이트버퍼도 읽을 수 있게 만듬

Performance Evaluation

- CDH 4.2.1 + Impala 1.0 GA
- 오픈소스 벤치마크 도구 "HiBench"
  - https://github.com/hibench
- 1/10 스케일
- Read-only 성능을 평가하기 위해서 "INSERT INTO TABLE ..." 쿼리 삭제
- 몇 개의 파일 포맷과 압축코덱을 조합
- 잡 쿼리 latency를 비교
- 평균 5번의 측정

http://www.slideshare.net/sudabon/performance-evaluation-of-cloudera-impala-ga

Performance Evaluation

- 마스터 * 3 (stand-by 노드 포함), 슬레이브 * 11
- 슬레이브 서버
  - CPU
    - Intel Core 2 Duo 2.13 GHz with Hyper Threading
  - OS
    - Centos 6.2
  - Memory
    - 4G
  - Disk
    - 7,200 rpm SATA HDD * 1

http://www.slideshare.net/sudabon/performance-evaluation-of-cloudera-impala-ga

Performance Evaluation

- Uservisits 테이블
  - 1억건 16,895MB as TextFile
- Rankings 테이블
  - 천만건, 744MB as TextFile

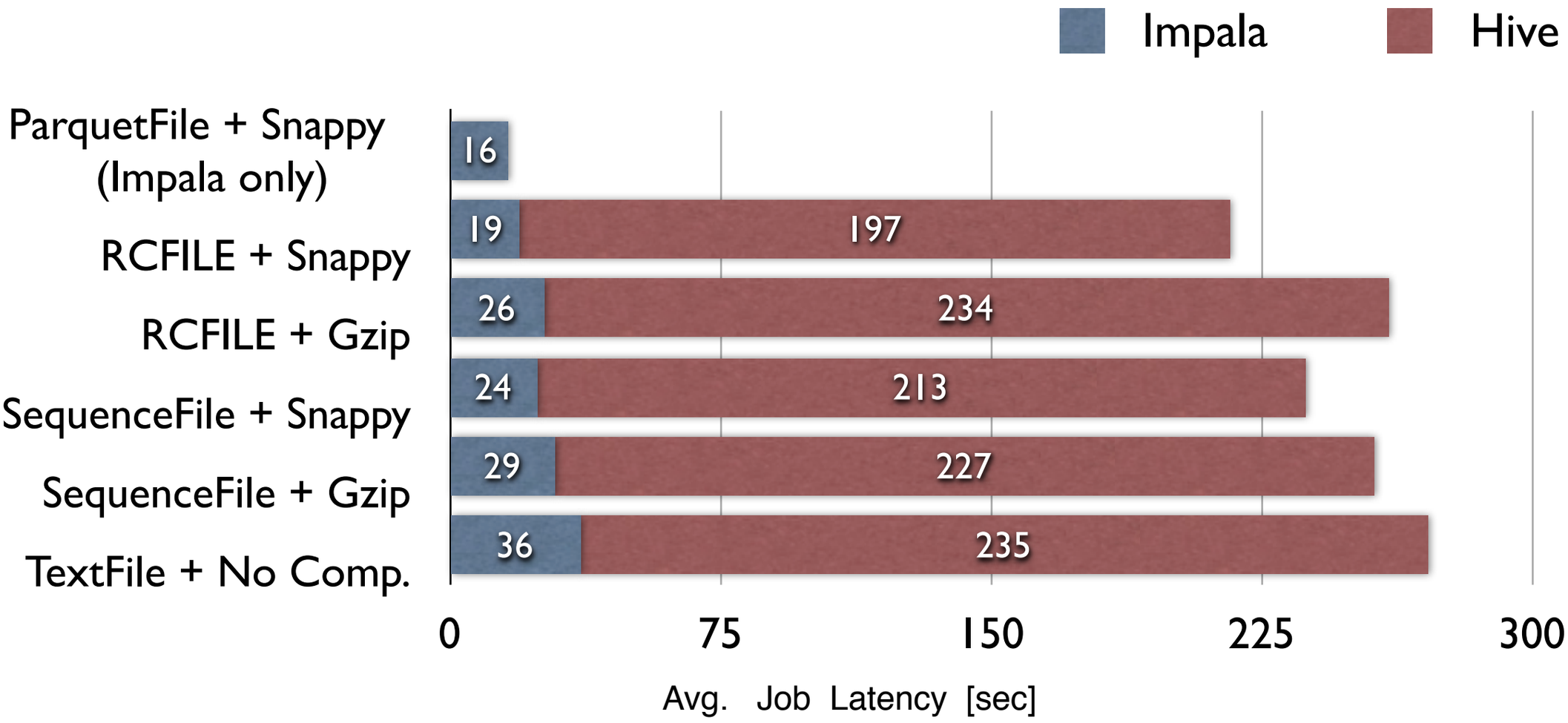
테스트 쿼리

```
SELECT sourceIP, sum(adRevenue) as totalRevenue, avg(pageRank)
FROM rankings_t R
JOIN (
      SELECT sourceIP,destURL, adRevenue
      FROM uservisits_t UV
      WHERE (datediff(UV.visitDate, '1999-0-01')>=0 AND datediff(UV.visitDate, '2000-01-01')<=0)
) NUV
ON   (R.pageURL = NUV.destURL) GROUP BY sourceIP order by totalRevenue DESC LIMIT 1;
```

kt NexR

http://www.slideshare.net/sudabon/performance-evaluation-of-cloudera-impala-ga

Benchmark Result



- Top-N 쿼리(limit 포함)라서 성능이 Hive 보다 월등하게 나온다
- 임팔라는 최종 결과셋이 작은 쿼리의 경우 성능이 높게 나온다
- 다른 쿼리에 경우에는 이 정도의 성능을 보장하지 못 함 (테스트 필요)

http://www.slideshare.net/sudabon/performance-evaluation-of-cloudera-impala-ga

Gigaom Report

- SQL-on-Hadoop platforms in 2013
- 섹터 로드맵 보고서
- 벤더
  - Cloudera, Hadapt, Teradata (SQL-H)
    EMC Greenplum (HAWQ)
    Citus Data, Splice Machine. JethroData
    Concurrent (Lingual)
- 아파치 프로젝트
  - Apache Drill, Hortonworks Stinger

kt NexR

Gigaom Report Conclusion

- 앞으로 1년~2년 예측
  - 클라우데라가 적어도 2년 정도 걸릴지라도 가장 유연하고 성공적인 솔루션 구축에 성공할 것
  - 두번째 오픈소스 대안은 Apache Drill 또는 Stinger 둘 중에 먼저 성숙한 프로젝트가 될 것 (클라우데라가 삽질하거나 모멘텀을 잃어버리면, 아파치 프로젝트 중 하나가 리더를 차지할 수 있을 것)
  - Hadapt은 틈새시장에서 이기기 위해 초점을 맞추고 연구할 것
  - EMC(Greenplum) , Teradata는 버그 없는 솔루션을 원하는 대기업의 초이스로 남을 것
  - 더 넓게 데이터베이스 시장에는, newSQL 데이터베이스가 성숙할 것이고 사용자는 하둡에서 멀어질 것
  - NewSQL은 Strong structure를 가진 빅데이터에서 사용될 것이고 SQL-on_Hadoop 솔루션들은 semistructured 데이터에 사용될 것이다

Conclusion

- SQL on Hadoop 경쟁 제품 중에서는 가장 성숙
- CDH 의존성이 생김
- Hive와 윈-윈 전략
  - 기존 하둡 / 하이브 사용자를 자연스럽게 흡수
- Hive와 서로 보완재로서 다른 제품을 제치고 시장에 안착할 가능성이 높다
- Hive는 high latency 쿼리에 적합(배치 데이터 처리)
- Impala는 작은 결과셋을 주는 쿼리에 적합 (Top-N 쿼리)
- Hbase는 select 쿼리에 적합(row key 조회)

Resources

- Impala 문서
  - http://www.cloudera.com/content/support/en/documentation/cloudera-impala/cloudera-impala-documentation-v1-latest.html
- 클라우데라 블로그
  - http://blog.cloudera.com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real/
- Dremel 논문
  - http://research.google.com/pubs/pub36632.html
- Impala 1.0 성능 평가
  - http://www.slideshare.net/sudabon/performance-evaluation-of-cloudera-impala-ga
- Gigaom 보고서
  - http://cloudera.com/content/dam/cloudera/Resources/PDF/sql-on-hadoop-roadmap-2013.pdf
- 발표자료
  - http://www.slideshare.net/SwissHUG/cloudera-impala-15376625