GRUTER
Enjoy Connecting

Tajo와 SQL on Hadoop

2013. 08. 29

최현식 책임 연구원

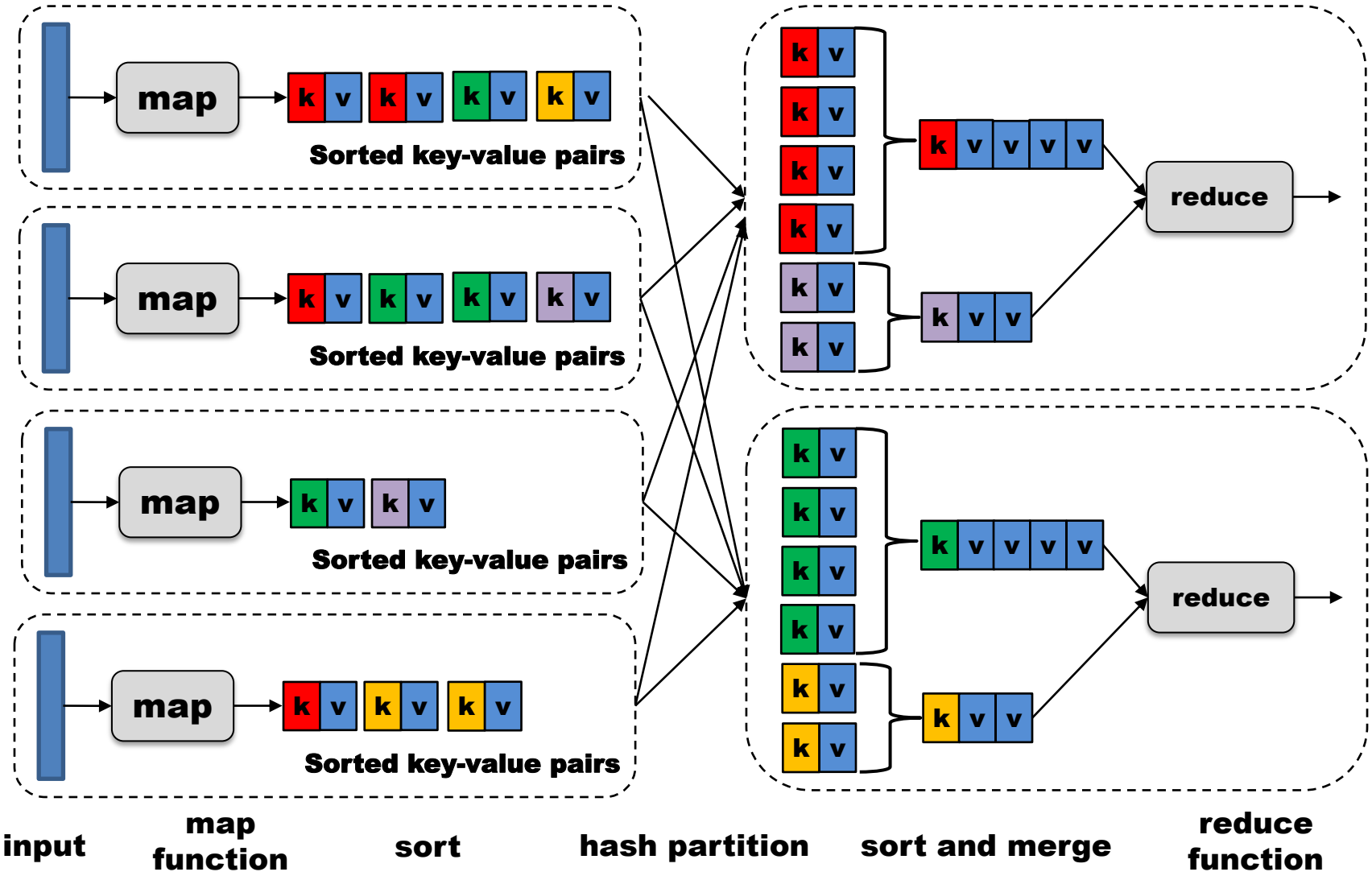
Index

- Hadoop & MapReduce
- SQL-on-Hadoop Systems
- Tajo 소개
- Tajo 설계 동기 및 목표
- Tajo 아키텍쳐
- Tajo JIT & Vectorized Query Engine
- Tajo 로드맵
- 벤치마크

Hadoop과 MapReduce

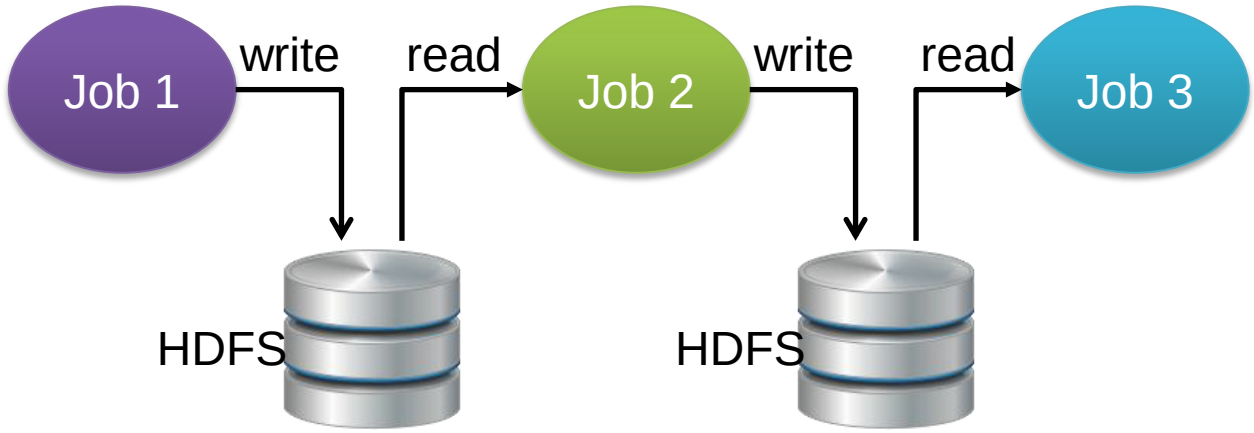
- **유연성**
  - 다양한 데이터 포맷 지원 (구조적, 비구조적 데이터, 텍스트)
  - 범용 프로그래밍 언어를 이용하여 다양한 알고리즘 적용 가능
- **확장성**
  - 노드 증가에 따른 선형적인 (성능, 용량) 확장 가능
- **비용**
  - 다수의 범용 서버 클러스터에서 동작하도록 설계
  - 노드 증가를 통한 용량 증설 및 처리 성능 향상 비용이 저렴

MapReduce의 동작 방식



MapReduce의 한계

- Map과 Reduce 간 셔플의 한계
  - merge sort->hashing->merge sort
- Job 간의 데이터 교환 오버헤드
- 관계형 데이터에 부적합
- 고정된 data flow



기존 MapReduce 기반 SQL 처리 시스템

- Hive (<=0.11)
  - HiveQL 을 다수의 MapReduce 작업으로 변환하여 동작
  - SQL 표준 미지원 (비슷하나 많은 부분 상이 및 미지원)
  - 높은 지연 시간
    - 각 MapReduce Job 별 5~15초에 달하는 시동 시간
  - 낮은 노드 당 처리 성능
    - Tuple 단위 처리 모델 (Tuple-at-a-time 방식)
  - Shuffle로 야기되는 큰 오버헤드
    - 중간 데이터 materialization
    - Pull 방식이 야기하는 Random Access

=> 대용량 데이터 배치처리만 가능, 처리 시 비효율적인 부분 많음

SQL-on-Hadoop

- **Hadoop 기반의 차세대 분석 엔진들을 지칭**

- **공통적 특징**
  - SQL 표준 지원
    - 기존 시스템과 통합 또는 대체 용이
  - 높은 처리 성능 (high throughput)
    - MapReduce의 한계를 극복하는 분산 처리 프레임워크
    - CPU와 메모리를 보다 잘 활용하는 처리 방식
  - 낮은 반응 시간 (low latency)
    - 100 msec ~

Why SQL-on-Hadoop?

- **Needs의 변화**
  - '투자대비 저렴한 가격으로 대용량 데이터 처리에 만족'
    -> '보다 높은 처리 성능 및 빠른 반응' 요구
  - 많은 사용자가 Ad-hoc 질의를 위해 DB 병행 사용에 불만

- **대화형 질의 (Interactive Query)**
  - 발견은 '질의 -> 결과 분석과 사고 -> 질의'의 순환
    - 시스템의 빠른 반응 속도가 데이터 분석의 생산성
  - 빠른 의사 결정 가능

- **성능 보장 및 사람에 의한 오류 방지**
  - MapReduce 프로그래밍
    - 개발자 역량에 의존적
    - 버그 가능성 높음
  - 질의 언어
    - 적절한 성능은 시스템이 보장
    - 버그 가능성 낮음

다양한 SQL-on-Hadoop 시스템

- **Cloudera Impala**
  - Low-latency 질의 처리에 특화되어 설계
  - Block-at-a-time 방식 엔진
  - 고성능을 위해 C++
  - SIMD과 LLVM 이용하여 String 처리, virtual 함수 호출 빈도수 격감
  - 인메모리 처리에 특화되어 대용량 데이터, 결과 값이 큰 데이터 처리에 한계
  - 소스는 Open, 참여는 Closed

- **Hortonworks Stringer**
  - Hive 기반 시스템
  - Vectorized 엔진 도입으로 기존 튜플 단위 처리 엔진 대체 작업 중
  - Tez (incubating)를 개발하여 MapReduce 프레임워크 대체 계획
  - Tez가 아직 초기 개발 상태(early stage)에 있음

Apache Tajo

- **Tajo**
  - Hadoop 기반 Data Warehouse 시스템
  - HDFS 및 다양한 소스의 대용량 데이터에 대한 ETL, 집계 연산, 조인, 정렬 제공
- **호환성**
  - 거의 완벽한 SQL 호환
  - JDBC 지원, ODBC 지원 (추후 계획)
  - UDF 지원

- **고성능 (high throughput) 및 낮은 반응 시간 (low latency)**
  - 유연하고 효율적인 분산 처리 엔진
  - 비용 기반 최적화 엔진 (cost-based optimization)
  - JIT Query Compilation 및 Vectorized 질의 엔진
- **오픈소스**
  - Apache Software Foundation의 인큐베이팅 프로젝트
  - 완전한 커뮤니티 기반 프로젝트 (열린 참여)

Tajo 설계 동기 (1/2)

- **기존 시스템에게 얻은 교훈 및 중요 우선 순위**

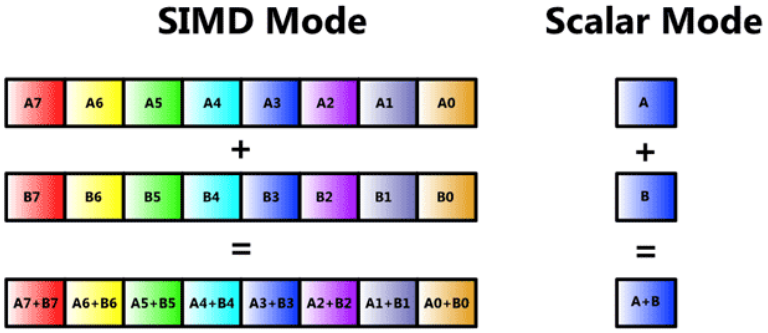
1. 잘못된 질의 계획 및 최적화
   - 잘못된 질의 계획은 수 분 짜리 질의를 수 시간이 걸리게 할 수 있음

2. 분산 처리 프레임워크
   - 테스크 시작 오버헤드 (노드 당 낮은 처리량)
     - MR은 각 테스크 당 수십 msec ~ 1초 이상
     - 반면 64MB 데이터 기준 task 처리 속도는 약 1초
   - 중간 데이터 전달 부하가 분산 처리의 주 병목 지점
     - Pull 방식이 유발하는 Random I/O로 인해 네트워크 대역폭 활용도 낮음
   - 고정된 Map -> Reduce 단계
   - Hash Shuffle로 인해 적은 최적화 기회

3. 워커 레벨의 데이터 처리 엔진
   - 낮은 처리 성능
   - CPU-friendly 하지 않은 구현
   - Scale-up 대한 고려 거의 없음

Tajo 설계 동기 (2/2)

- **하드웨어의 발전**
  - Many Cores
    - 단일 범용 서버가 8-24 코어 보유
  - 더욱 빨라지는 스토리지
    - 향후 3-5년내 SSD 서버 스토리지 대중화 예상
    - SSD의 순차 읽기 속도 500MB/s ~ 1500MB/s
    - CPU Core 당 처리 속도 2GB/s
    - Disk I/O -> CPU 병목 예상
  - SIMD의 발전
    - 최신 CPU들은 String 비교나 Hashing을 위한 명령어 까지 제공
    - 단일 명령어 처리 데이터 증가 (향후 256bit -> 1024bit)



(SIMD vs. scalar operations)

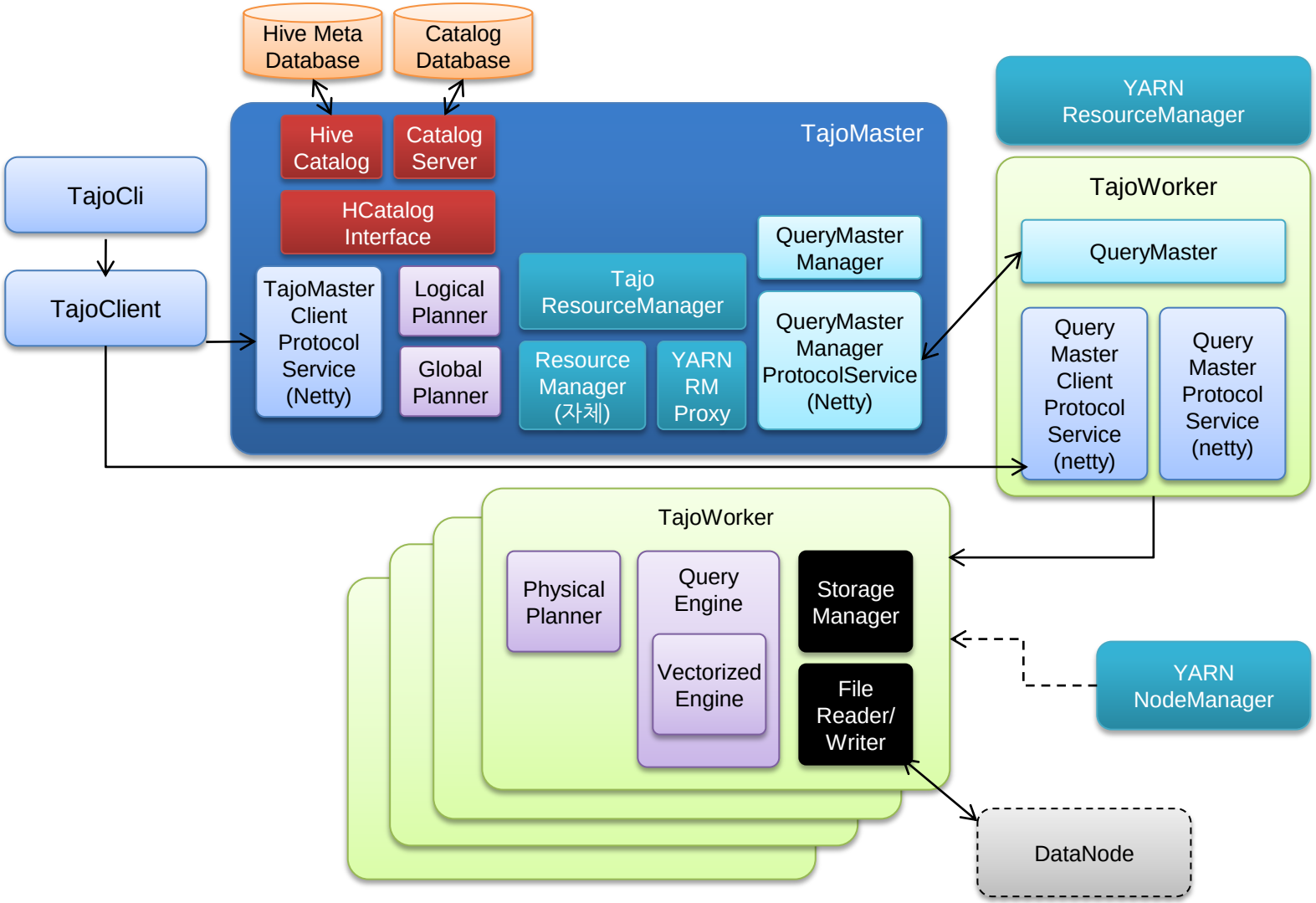
Tajo 설계 및 구현 목표

- 확장성
- 성능
- 내고장성
- 고도의 질의 계획 최적화
- 유연한 분산 처리 플랫폼
- Scale-up 환경에서 하드웨어 자원 활용 극대화

Tajo 아키텍쳐

- **마스터-워커 모델 + 질의 별 Query Master 동작**
  - Protocol Buffer + Netty, Boost ASIO 기반의 RPC 기반
- **Tajo Master**
  - 클라이언트 및 어플레케이션 요청 처리
  - 카탈로그 서버
    - 테이블 스키마, 물리적인 정보, 각종 통계
    - JDBC 이용 외부 RDBMS를 저장소로 사용
  - Query 파서, 플래너, 최적화, 클러스터 자원 관리, Query Master 관리
- **Query Master (각 질의 별 동작)**
  - 분산 실행 계획 생성
  - Execution Block (질의 실행 단계) 제어
  - 테스크 스케쥴링
- **Tajo Worker**
  - 스토리지 매니저, 로컬 질의 엔진
  - C++ 구현

Tajo 아키텍쳐



Tajo의 질의 계획 및 최적화 엔진

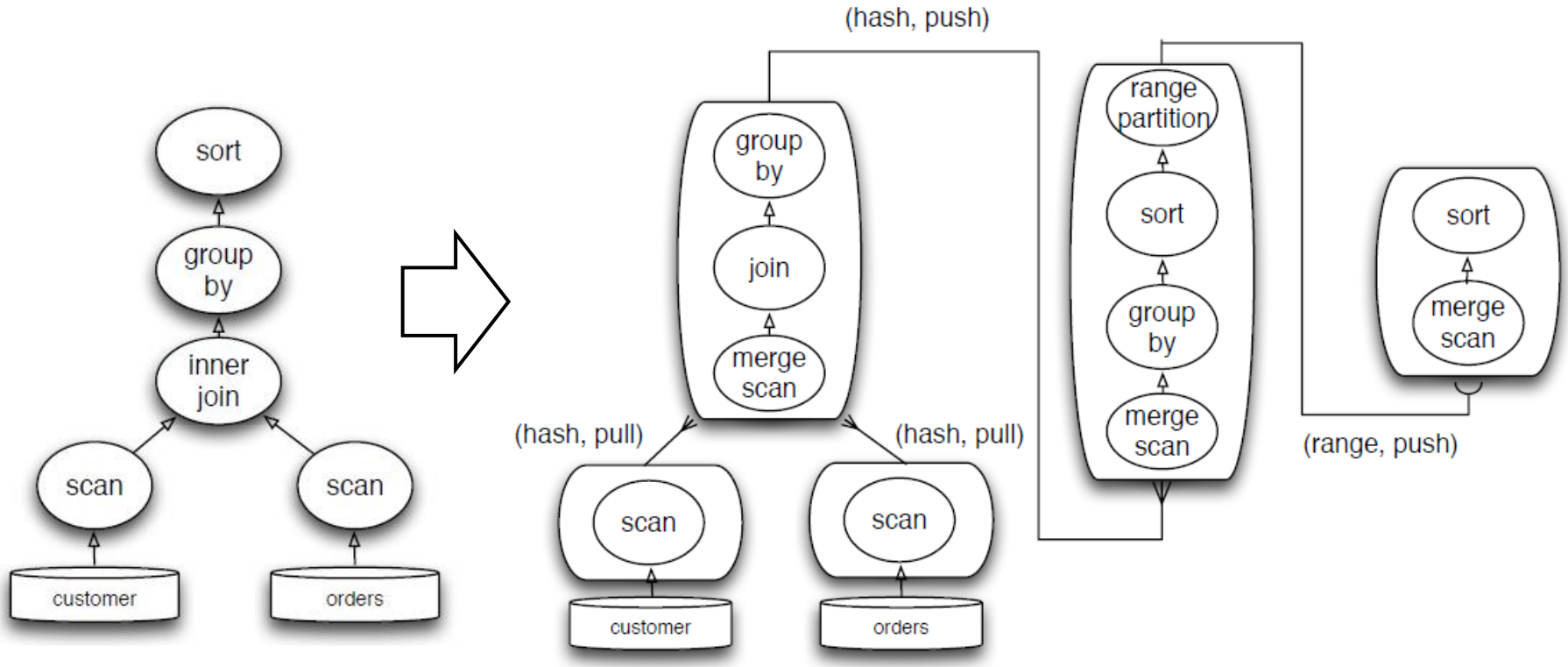
- **비용 기반 최적화 (Cost-based optimization)**
  - 시스템이 최적의 조인 순서를 탐색 및 선택
  - 사용자에게 의존하지 않음
- **확장 가능한 Rewrite Rule 엔진**
  - 기존 상용 DB 수준의 다양한 Rewrite rule로 확장 가능
- **적응형 최적화 (Progressive query reoptimization)**
  - 질의 실행 시간에 통계 정보를 기반으로 질의의 남은 부분 최적화
  - 나쁜 질의 계획 회피 가능

Tajo의 데이터 셔플 (Shuffle) 메커니즘

- **데이터 Shuffle 방법**
  - Hash
    - 노드들에게 할당된 hash key 값에 만족하도록 데이터 재분배
  - Range
    - 노드들에게 할당된 값 범위에 만족 하도록 데이터 재분배

- **데이터 전송 방법**
  - Pull
    - 중간 데이터를 저장하고 다른 워커가 끌어가는 방식
    - 데이터에 비해 자원이 충분치 않아 단계별로 데이터를 처리할 때
  - Push
    - 중간 데이터의 디스크 저장 없이 데이터를 다른 워커에게 전송
    - 자원이 충분하고 여러 데이터 처리를 동시에 파이프라이닝 가능할 때 사용

An Example of Distributed Execution Plan



(A join-groupby-sort query plan)

(A distributed query execution plan)

```
select col1, sum(col2) as total, avg(col3) as
averagefrom r1, r2 where r1.col1 = r2.col2 group by
col1 order by average;
```

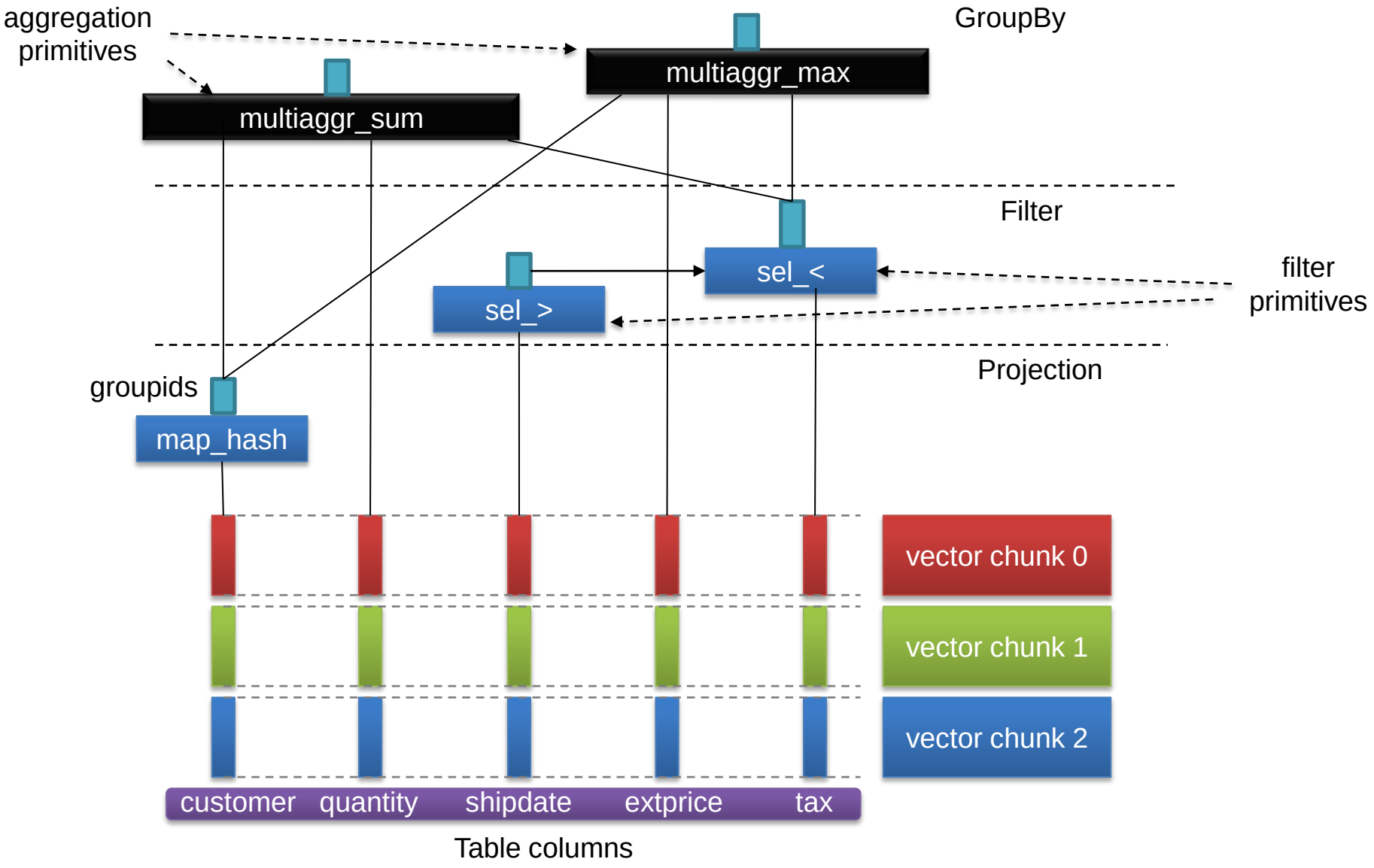
JIT 및 Vectorized Engine

- **C++ 구현**
  - 직접 로우 레벨 최적화 가능
  - SIMD 명령
  - 메모리 효율적인 관리 (No Garbage Collection, low memory footprint)

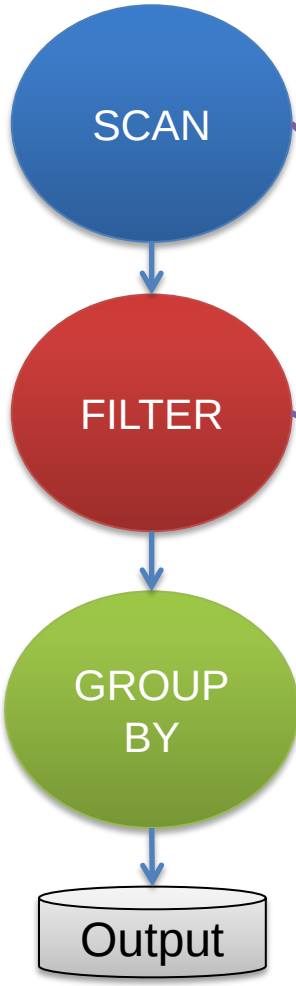
- **Vectorized Engine**
  - 데이터를 CPU Cache 크기에 맞춘 Vector들로 유지
  - Vector 단위 (원시 타입 배열) 데이터 처리
  - 각 vector 마다 작은 loop를 반복하여 데이터 처리
  - CPU Pipelining 향상, SIMD 적용 가능, 높은 CPU Cache 적중률

- **JIT using LLVM**
  - 실행 시간 (runtime)에 필요한 처리 primitive 생성 및 캐쉬
    - 각 머신 별 최적화된 코드 생성 가능
    - 인터프리트 오버헤드 제거
    - Branch misprediction 감소 -> CPU Pipelining 기회 증대
  - 실행 시간에 Physical Execution 코드 생성

Vectorized Query Processing



JIT Query Compilation




```
const float vec_num = 1000;
float l_extended_price[vec_num], l_discount[vec_num]
long l_shipdate[vec_num], l_quantity[vec_num]
int selected[vec_num];
date date1 = date("1994-01-01");
date date2 = date("1995-01-01");
float val1 = 0.05; val2= 0.07;
double output1[vec_num];
double output2[vec_num];

do {

  fetch_vectors(l_extended_price, l_discount, l_shipdate,
                l_quantity);

  bool_geq_date_col_date_val(l_ship_date, date1, selected)
;
  bool_lt_date_col_date_val(l_ship_date, date2, selected);
  bool_geq_float_col_float_val(l_discount, 0.05, selected)
;
  bool_lt_float_col_float_val(l_discount, 0.07, selected);
  bool_gt_float_col_float_val(l_quantity, 24, selected);

  map_mul_float_col_float_col(output1, l_discount,
  l_quantity, selected);
  agg_sum_float(output2, output1, selected);

  write_outputbuffer(output2);
} while (hasNext());
```

An example of runtime generated physical execution plan

Tajo 로드맵

- **2013. 10월 초: Apache Tajo 0.8 Release**
  - 자체 분산 처리 엔진
  - 기존 Java 기반 처리 엔진
  - SQL 표준 거의 지원
  - HiveQL 모드 지원
  - Query Rewrite Rule 엔진 탑재 + 기초적인 Rewrite Rule
- **2014. 1월: Apache Tajo 1.0 Release**
  - 다수 Rewrite Rule 탑재
  - Cost-based Optimization
  - Adaptive Query Reoptimization
  - JIT Query Compilation & Vectorized Engine

벤치 마크 테스트

- Tajo 자바 엔진 버전, Impala 0.7, Hive 0.10
- 2013. 01 벤치 마크

- Experiment Environments
  - 100GB TPC-H Data Set
    - http://www.tpc.org/tpch/spec/tpch2.16.0.pdf
    - 5 Cluster Nodes
    - Intel i5
    - 16GB Memory
    - 2TB HDD x 5
    - 1G Ethernet
    - Hadoop 2.0.2-alpha

벤치 마크 테스트

	Q1	Q3	Q6
TAJO	112 sec	330 sec	121 sec
Impala	89 sec	414 sec	141 sec
Hive		827 sec	346 sec

TPC-H Q1, Q3, Q6 벤치마크 비교표

GRUTER: YOUR PARTNER IN THE BIG DATA REVOLUTION

Phone +82-70-8129-2950
Fax +82-70-8129-2952
E-mail contact@gruter.com
Web www.gruter.com

Gruter, Inc.
5F Sehwa Office Building 889-70 Daechi-dong, Gangnam-gu, Seoul, South Korea 135-839