Hadoop Practice

19기 엔지니어링 김동진 이경윤

01 / 하둡 에코시스템

- MapReduce
- Hive

02 / 실습

- MapReduce 실습
- Hive 실습

MapReduce 정의

-대용량 데이터 처리를 위한 분산 프로그래밍 모델. 대량의 데이터를 병렬로 분석이 가능하다.
-Map 그리고 Reduce라는 두 개의 메소드로 구성
-Map: key - value의 형태로, 흩어져있는 정보를 연관성 있는 데이터 분류로 묶는 작업
-Reduce: Map화 한 작업 중 중복 데이터를 제거하고 원하는 데이터를 추출

Mapper Reducer Example

```
Let map(k, v) =
        emit(k.toUpper(), v.toUpper())

('foo', 'bar') -> [('FOO', 'BAR')]
('foo', 'other') -> [('FOO', 'OTHER')]
('baz', 'more data') -> [('BAZ', 'MORE DATA')]
```

대문자로 바꾸는 함수 (Pseudo Code)

Mapper Reducer Example

```
let reduce(k, vals) =
    sum = 0
    foreach int i in vals:
        sum += i
    emit(k, sum)

('bar', [9, 3, -17, 44]) -> ('bar', 39)
('foo', [123, 100, 77]) -> ('foo', 300)
```

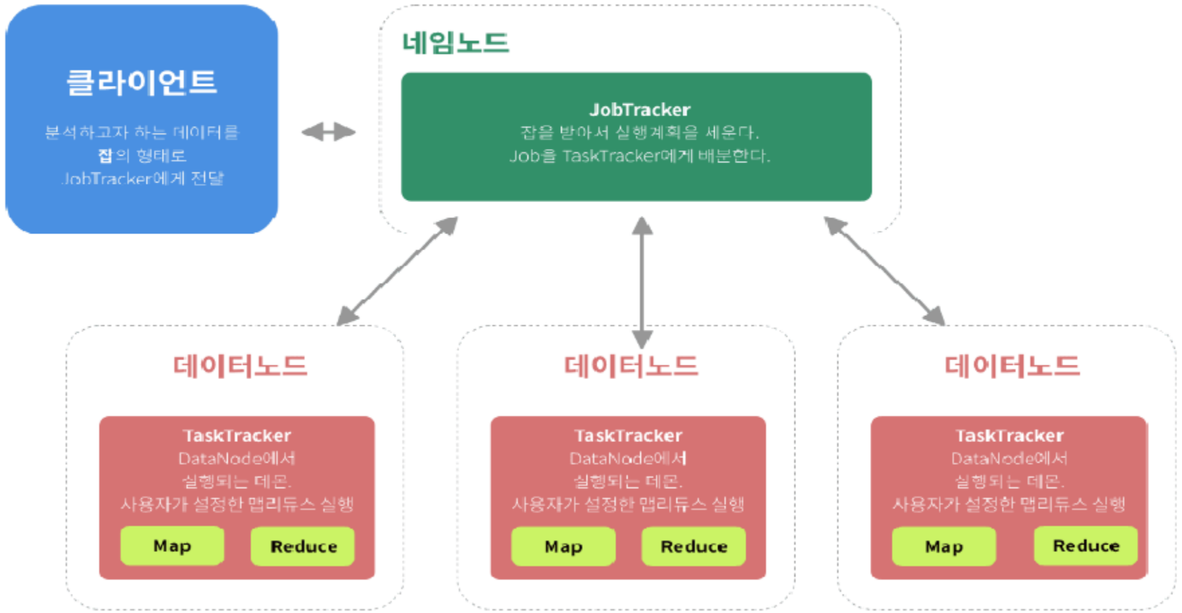
키 값과 관련있는 value를 모두 더하는 함수

MapReduce 특징

-일괄 처리 시스템
-데이터의 입출력과 병렬처리 등 기반 작업을 프레임워크가 알아서 처리
-타입은 프로그래머가 선택, map 함수와 reduce 함수 또한 프로그래머가 작성

MapReduce 구성

맵리듀스



MapReduce 구성

-잡
클라이언트가 하둡으로 실행을 요청하는 MapReduce 프로그램의 작업 단위

-클라이언트
사용자가 실행한 맵리듀스 프로그램
혹은 하둡에서 제공하는 맵리듀스 API
구현된 Map Reduce 잡을 제출하는 실행주체
사용자는 맵리듀스 API로 맵리듀스 프로그램을 개발하고 개발한 프로그램을 하둡에서 실행할 수 있다.

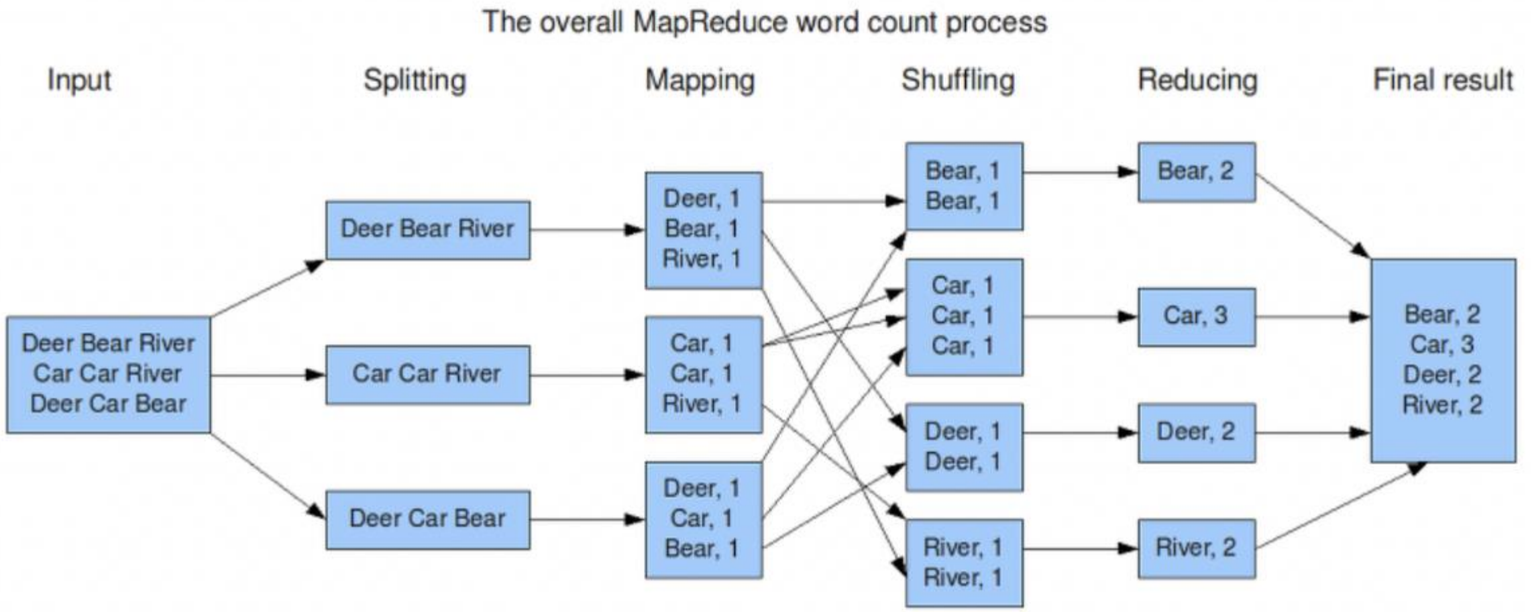
MapReduce 구성

-잡 트래커
전체 하둡 클러스터에서 하나의 잡트래커 실행
사용자가 잡을 요청하면 잡트래커는 잡을 처리하기 위한 맵과 리듀스의 사용계획을 계산하고 어떤 태스크트래커에서 실행할지 결정하여 잡을 할당한다.
잡트래커와 태스크트래커는 하트비트 메서드로 네트워크 통신을 하며 작업의 실행 정보를 주고 받는다. 태스크트래커에 장애가 발생하면 잡트래커는 다른 대기중인 태스크트래커를 찾아 태스크를 재실행 하게 된다.

MapReduce 구성

-태스크 트래커
잡을 받아 요청 받은 맵과 리듀스 개수만큼 맵 태스크와 리듀스 태스크를 생성한다.
하둡의 데이터노드에서 실행되는 데몬
맵 태스크, 리듀스 태스크가 생성되면 새로운 JVM을 구동해 맵 태스크 , 리듀스 태스크를 실행한다. (하나의 데이터노드만으로 여러개의 JVM을 통해 데이터 분석 병렬처리가 가능)

MapReduce 실행과정



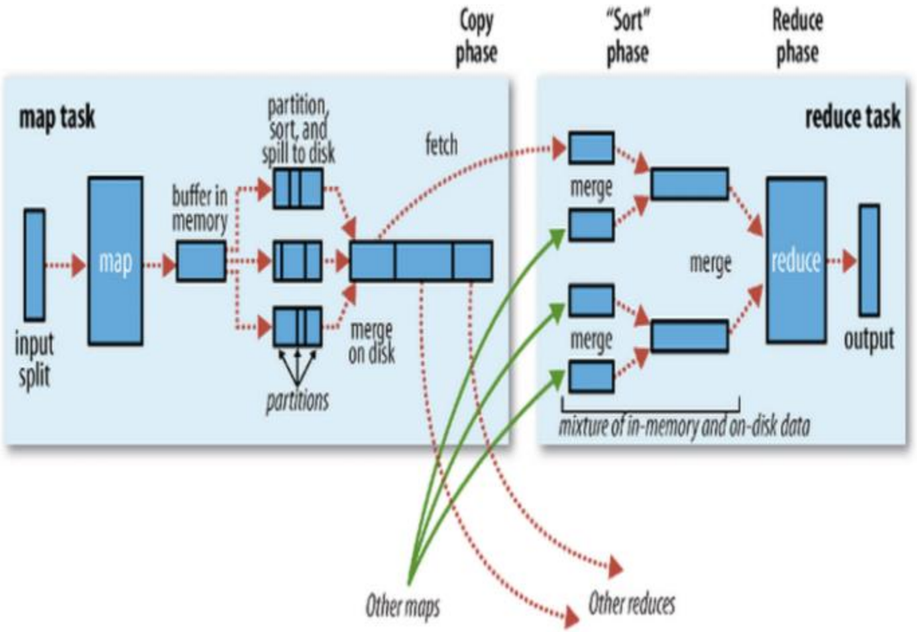
MapReduce 실행과정

1.Input
-데이터를 입력하는 과정
2.Splitting
-데이터를 쪼개어 HDFS에 저장하는 과정
-큰 데이터가 들어오면 64MB 단위 블록으로 분할
3.Mapping
-데이터를 key, value 쌍의 형태로 연관성 있는 데이터 분류를 묶는 작업
-필요한 분석 대상만 추출
-잘못된 레코드를 제거
4.Shuffling

5.Reducing
-모든 값을 합쳐서 우리가 원하는 값을 추출할 수 있다.

MapReduce 실행과정

4. Shuffling
   -맵 함수의 결과를 취합하기 위해 리듀스 함수로 데이터를 전달하는 과정.
   -멥 태스크와 리듀스 태스크의 중간 단계
   -맵 함수 결과에 대한 병합 (Sort & Merge) 작업이 일어남.
   -각 서버에 나뉜 데이터들을 키를 중심으로 합쳐 리듀스 함수에 전달한다.



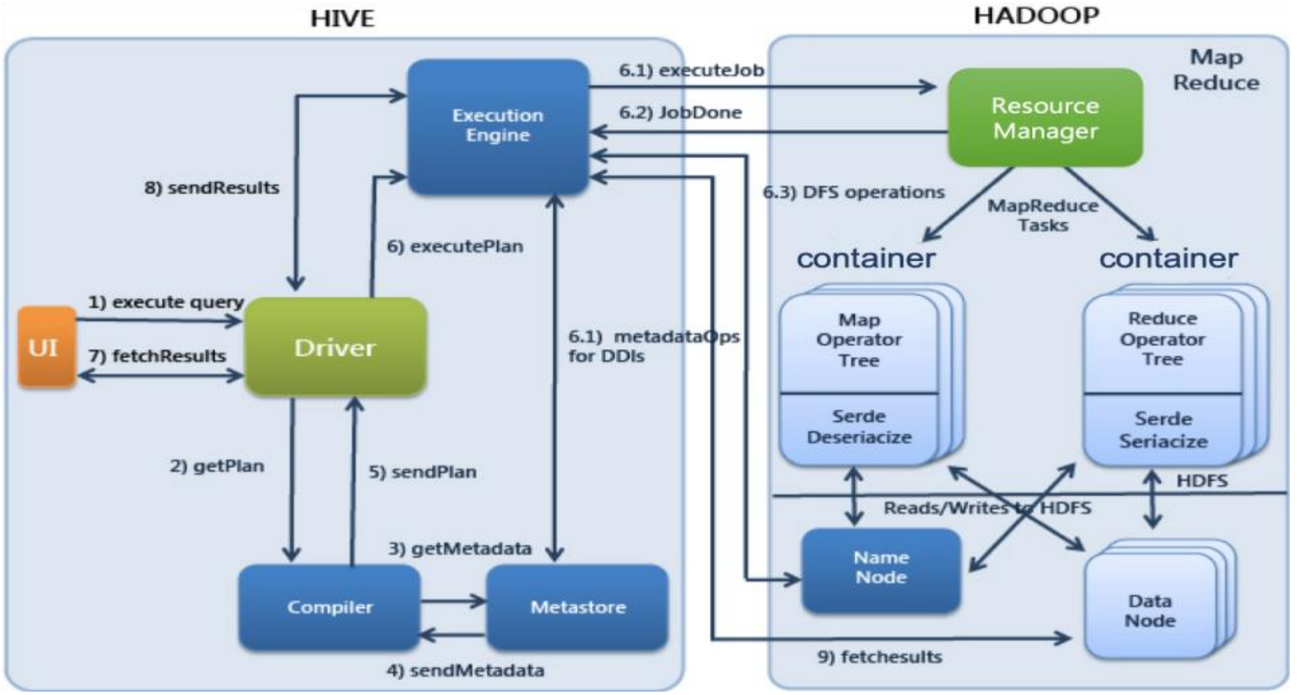
Hive 정의

-Hadoop은 기존의 데이터베이스 환경이 가지고 있던 한계를 넘는 기능을 제공해 주었지만
복잡한 맵리듀스 코드를 작성해야 하고
테이블같은 논리적인 개념이 없어서 파일 단위 또는 디렉토리 단위로 데이터를 관리하였습니다.
-SQL기반의 접근을 통한 데이터 처리와 데이터베이스, 테이블, 파티션과 같은
논리적 레벨의 카탈로그 및 메타데이터를 제공한다.

```java
public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
{
    // ...
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable> {
    // ...
 }

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
 }

}
```

```sql
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

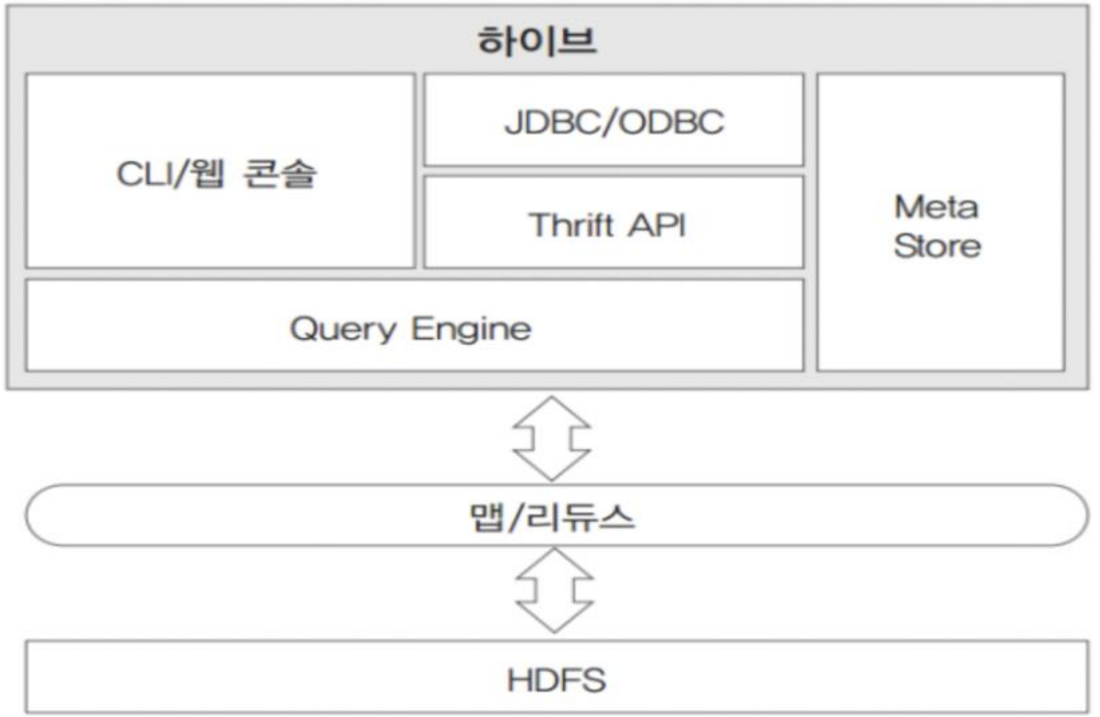
Hive 구성요소


HIVE
HADOOP
Map Reduce
Execution Engine
Resource Manager
6.1) executeJob
6.2) JobDone
8) sendResults
6.3) DFS operations
MapReduce Tasks
6) executePlan
container
container
1) execute query
UI
7) fetchResults
Driver
6.1) metadataOps for DDls
Map Operator Tree
Serde Deseriacize
Reduce Operator Tree
Serde Seriacize
2) getPlan
5) sendPlan
HDFS
Reads/Writes to HDFS
3) getMetadata
Name Node
Data Node
Compiler
Metastore
9) fetchesults
4) sendMetadata

Hive 구성요소

-UI
    사용자가 쿼리 및 기타 작업을 시스템에 제출하는 사용자 인터페이스
    CLI, Beeline, JDBC 등
-Driver
    쿼리를 입력받고 작업을 처리
    사용자 세션을 구현하고, JDBC/ODBC 인터페이스 API 제공
-Compiler
    메타 스토어를 참고하여 쿼리 구문을 분석하고 실행계획을 생성
-Metastore
    디비, 테이블, 파티션의 정보를 저장
-Execution Engine
    컴파일러에 의해 생성된 실행 계획을 실행

Hive 아키텍쳐


하이브
CLI/웹 콘솔
JDBC/ODBC
Thrift API
Query Engine
Meta Store
맵/리듀스
HDFS

Hive 실행순서

-사용자가 제출한 SQL문을 드라이버가 컴파일러에 요청하여 메타스토어의 정보를 이용해 처리에 적합한 형태로 컴파일

-컴파일된 SQL을 실행엔진으로 실행

-리소스 매니저가 클러스터의 자원을 적절히 활용하여 실행

-실행 중 사용하는 원천데이터는 HDFS등의 저장장치를 이용

-실행결과를 사용자에게 반환

Hive 특징

1) Hive 1.0

-SQL을 이용한 맵리듀스 처리
-파일 데이터의 논리적 표현
-빅데이터의 배치 처리를 목표
-MR engine을 사용(default engine이 MR)

2) Hive 2.0

-LLAP(Live Long and Process) 구조 추가
-Spark 지원 강화
-CBO 강화
-HPLSQL 추가
-Tez Engine이 추가(default engine이 Tez로 변경)

Hive 특징

-LLAP

작업을 실행한 데몬을 계속 유지하여, 핫 데이터를 캐싱하여 할 수 있어 빠른 속도로 데이터를 처리할 수 있다.

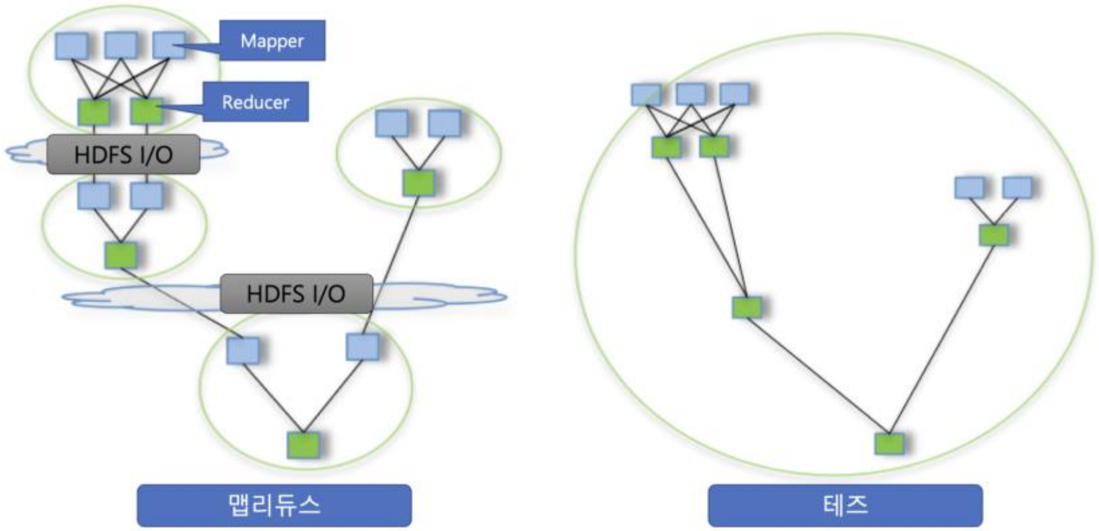
-HPLSQL

오라클의 PL/SQL과 비슷한 Procedural SQL을 지원합니다. 재사용 가능한 스크립트 작성을 목표로 개발 되었다.

-TEZ engine

테즈(TEZ)는 YARN 기반의 비동기 사이클 그래프 프레임워크이다.
테즈는 맵단계 처리 결과를 메모리에 저장하고, 이를 리듀스 단계로 바로 전달한다.
리듀스 작업의 결과를 맵단계를 거치지 않고 리듀스 단계로 전달하여 IO 오버헤드를 줄여서 속도를 높일수 있다.
MR에 비해 30% 정도 향상된 성능을 보인다.

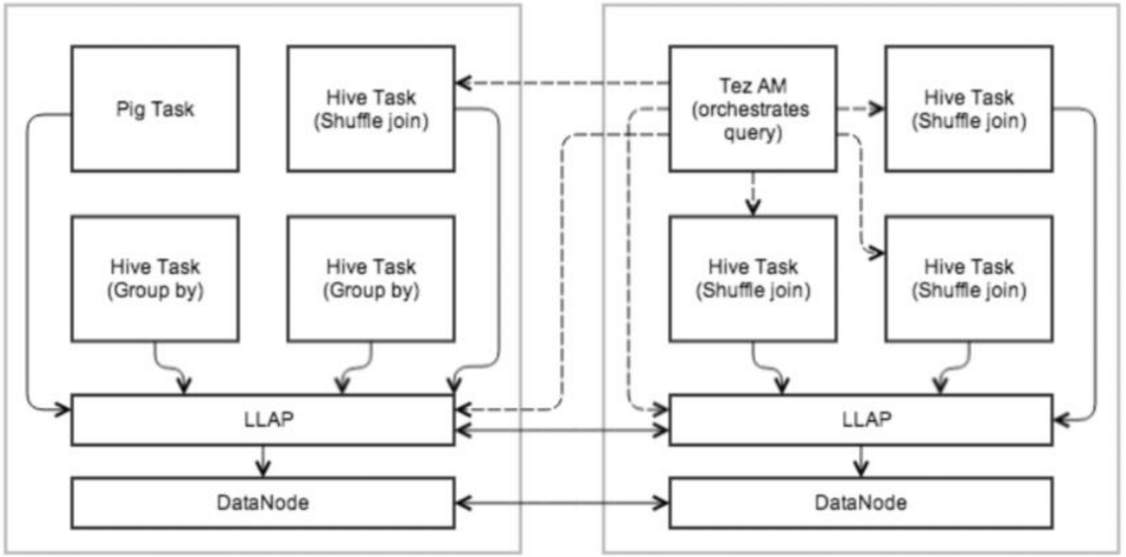
TezEngine



맵리듀스는 하나의 작업이 여러 단계의 맵,
리듀스를 거치며
중간 작업 결과를 HDFS에 쓰고,
다시 맵단계에서 파일을 읽어서 처리한다.

테즈는 맵단계 처리 결과를 메모리에 저장하
고, 이를 리듀스 단계로 바로 전달한다.

LLAP



LLAP는 작업을 도와주는 보조도구 이다.
실제 작업을 처리하는 MR, TEZ 같은 작업 엔진이 아니다.
또한 HDFS같이 데이터를 영구히 저장하지 않습니다.

HPLSQL

```
FOR i IN 1..10 LOOP
  DBMS_OUTPUT.PUT_LINE(i);
END LOOP
```

for를 이용한 루프문이나 커서 등을 이용할 수 있다.

Hive 3.0

맵리듀스 엔진, 하이브 CLI를 제거하고 TEZ엔진과 비라인을 이용하여 작업을 처리하도록 수정되었다.

롤을 이용한 작업 상태 관리(workload management)
-> SQL을 이용하여 워크로드 관리를 위한 롤을 생성하고 이를 적용하여 작업의 부하에 따라 쿼리의 성능, 실행 여부를 제어할 수 있습니다.

구체화 뷰(Materialized View) 추가

쿼리 결과를 캐싱하여 더 빠른속도로 작업 가능

테이블 정보 관리 데이터베이스 추가
-> 하이브 메타스토어에서가 아닌 데이터베이스에서 전체 테이블의 칼럼 정보, 통계정보 등을 확인할 수 있다.

Practice

Thank You