한국컴퓨터정보학회지
제21권 제1호, 2013. 6.

오픈소스 기반의 실시간 빅데이터 질의 기술 동향

• 심탁길(SK C&C)

I. 서론

하둡(Hadoop)은 저비용으로 대용량 데이터를 저장하고 빠르게 처리할 수 있는 시스템이며 빅데이터 구현을 위한 핵심적인 플랫폼 기술로 사용 되고 있다. 또한 하이브(Hive)는 사용자에게 친숙한 SQL이라는 질의 기술을 이용하여 하둡상의 저장된 데이터를 쉽게 처리하고 분석할 수 있도록 해주는 도구로 널리 사용되고 있다. 이러한 하둡과 하이브는 대용량 데이터를 배치 처리하는 데 최적화 되어 있지만, 실제 업무에서는 배치 처리뿐만 아니라, 데이터를 실시간으로 조회하거나 처리해야 하는 일들이 많다. 실시간 처리라는 측면에서 하둡의 제약사항을 극복하기 위한 다양한 시도가 있었으며, 이 중에 최근 주목 받고 있는 것이 실시간 SQL 질의 분석 기술이다. 이 기술은 하둡상에 저장된 대용량 데이터를 대화형식의 SQL 질의를 통해서 처리하고 분석하는 기술들이며, 본 논문에서는 최근에 오픈소스로 공개된 SQL 질의 기술인 임팔라를 살펴보고자 한다.

II. 관련 연구

1. 하둡과 하이브

하둡은 구글의 분산파일시스템인 GFS(Google Filesystem)와 맵리듀스(MapReduce)의 오픈 소스 구현체로서 빅데이터를 저장 관리하고 처리하기 위한 핵심 플랫폼 기술이다. 2005년 아파치 재단의 검색엔진 프로젝트인 너치(Nutch)의 하위프로젝트로 시작되었으며, 인터넷 포털인 야후의 적극적인 후원을 통해 비약적으로 발전하여서 오늘날 빅데이터 저장 및 처리에 근간이 되는 시스템이 되었다.

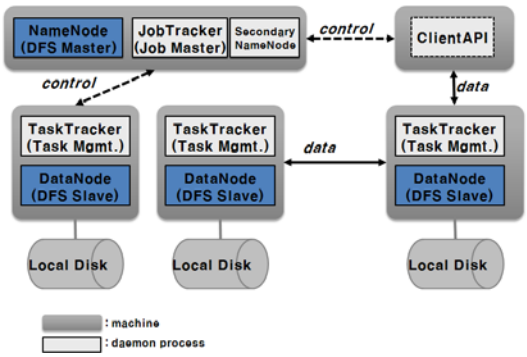

그림 1. 아파치 하둡 아키텍처

하둡의 분산 병렬 처리 프레임워크인 맵리듀스는 빅데이터를 빠르고 효과적으로 처리할 수 있는 방법을 제공하였지만, 개발자가 아닌 데이터 분석가들에게는 여전히 사용이 쉽지 않은 시스템이었다. 또한 개발자라고 하더라도 매번 데이터 처리 요청을 받을 때마다 맵리듀스 코드를 만든다는 것은 생산성이나 유지보수 측면에서 좋은 방법이 아니었다. 이러한 문제점들을 개선하기 위해서 만들어진 프로젝트가 아파치 하이브이다. 페이스북에 의해서 개발된 하이브는 사용자에게 친숙한 SQL 질의 기능을 제공하였다. 하이브에서 작성한 SQL 질의는 하이브 엔진에 의해서 맵리듀스 코드로 변환되어 하둡상에서 실행된다. 그래서 사용자는 맵리듀스 코드를

작성하지 않아도, 하이브를 이용하면 하둡상에 저장된 빅 데이터를 마치 테이블상의 데이터를 조회하는 것처럼 쉽게 처리하고 분석할 수 있게 되었다.

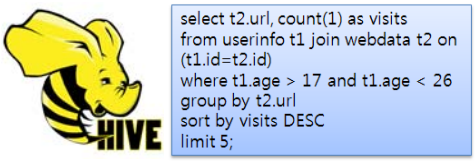

그림 2. SQL 질의 엔진 하이브

하이브는 빅데이터 기반의 데이터웨어하우징 시스템을 구축하는 데 많이 사용되고 있으며, 개발 생산성 및 유지보수 측면에서 효과가 검증된 솔루션이라고 할 수 있다. 오늘날에는 하둡 기반의 SQL 질의 기술로는 사실상의 업계 표준(defacto standard)이 되었다. 우수한 생산성과 유지보수 그리고 지속적인 성능 개선을 통하여 맵리듀스 코드와 비슷한 수준까지 성능이 개선 되었지만, 하이브 작업도 맵리듀스이기 때문에 태생적인 성능 한계를 극복할 수는 없었다. 맵리듀스 작업은 스케쥴링, 그리고 맵에서 리듀스로 넘어가는 과정에서 발생하는 네트워크 데이터 전송과 정렬 연산을 수행해야 하기 때문에 필연적으로 지연이 발생하는 구조이다. 즉 실시간 용도로 사용하지 못하고 현재로서는 배치 처리 용도로만 사용할 수 밖에 없다. 개발자 커뮤니티에서는 새로운 컬럼 파일 포맷, 로컬 쿼리 실행기 등 다양한 방법을 사용하여 개선 활동을 진행하고 있다.

2. 빅쿼리(BigQuery)와 드레멜(Dremel)

구글은 클라우드 기반의 빅데이터 서비스인 빅쿼리(https://developers.google.com/bigquery)를 2011년 초에 대중에게 공개하였다. 이 서비스를 이용하면, 인터넷으로 비정형 데이터를 업로드 한 후, 마치 데이터베이스의 테이블을 생성하고 SQL 질의를 하는 것과 같은 방식으로 데이터를 조회하고 분석할 수 있다.

크기 3.9GB, 1억 4천만개의 레코드로 구성된 데이터를 대상으로 성능 실험을 해보았다. 그림 3처럼 몸무게 많이 나가는 상위 10개의 레코드를 조회하는 SQL 문을 제작하여 질의를 수행 했을 때, 결과 값이 나오는 데 약 4.7초의 시간이 걸렸다.

```
SELECT WEIGHT_POUNDS, STATE, YEAR,
GESTATION_WEEKS
FROM publicdata:samples.natality
ORDER BY WEIGHT_POUNDS DESC LIMIT 10;
```

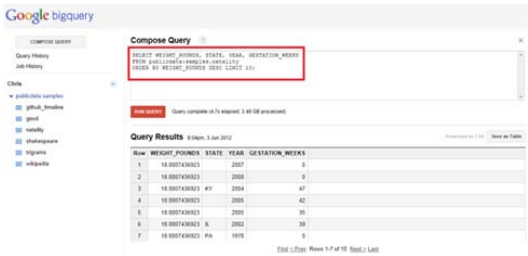

그림 3. 빅쿼리 질의 성능

구글의 빅쿼리는 하이브와 유사하게 SQL 질의를 이용하여 빅데이터를 분석할 수 있도록 해주는 서비스로 볼 수 있지만, 배치가 아닌 실시간으로 데이터를 분석할 수 있다는 것이 중요한 차이점이다.

빅쿼리 서비스의 내부 플랫폼으로 사용된 기술이 드레멜(Dremel)이다. 구글은 드레멜 기술과 관련된 내용을 2010년에 논문으로 대중에게 공개했다(http://research.google.com/pubs/pub36632.html)드레멜은 구글 파일시스템상에 저장된 빅데이터를 실시간으로 처리하고 분석 할 수 있는 대화형 SQL 질의 기술이다. 이후 하둡 프로젝트처럼 개발자 커뮤니티는 구글의 드레멜과 유사한 시스템을 오픈소스 프로젝트로 개발하려는 시도를 하였으며, 현재 아래와 같은 오픈소스 프로젝트들이 개발 되고 있다.

① 아파치 드릴(Drill): 하둡 전문 회사인 맵알(MapR)과 주축이 되어 진행하고 있는 프로젝트이다. 드레멜의 아키텍처와 기능을 동일하게 구현한 오픈 소스 버전의 드레멜이다.

② 아파치 스팅거(Stinger): 하둡 전문 회사인 호튼웍스에서 개발을 주도하고 있다. 새로운 엔진을 개발하기 보다는 기존의 하이브 코드를 최대한 이용하여 성능을 개선하는 식으로 개발이 진행되고 있다.

③ 샤크(Shark): 인메모리 기반의 대용량 데이터 웨어 하우징 시스템이며, 하이브와 호환되기 때문에 하이브 SQL 질의와 사용자 정의 함수(User Defined Fuction)를 사용할 수 있다.

④ 아파치 타조(Tajo): 고려대학교와 국내 빅데이터 전문

회사인 그루터(Gruter)의 개발자들이 주축이 되어 진행하고 있는 프로젝트이다.

⑤ 임팔라(Impala): 하둡 전문 회사인 클라우데라(Cloudera)에서 개발을 주고하고 있다.

⑥ 프레스토: 페이스북에서 자체적으로 개발하여 사용하고 있는 하둡 기반의 데이터웨어하징 엔진이다. 2013년 가을에 오픈 예정이다.

III. 본 론

1. 임팔라 개요

오픈소스 기반의 실시간 질의 기술 중 먼저 대중에게 공개된 기술이 임팔라이다. 임팔라를 제작한 클라우데라는 드레멜(Dremel)의 논문 “Interactive Analysis of Web-Scale Datasets”을 읽은 후 하둡 상에서 실시간, 애드혹(ad-hoc) 질의가 가능할 것 같다는 기술적 영감을 얻어서 개발을 시작했다. 이후 2012년 10월에 시험(Proof of Concept) 버전을 대중에게 공개했으며, 2013년 5월에 정식 버전(1.0)을 배포했다. 임팔라는 분석과 트랜잭션 처리를 모두 지원하는 것을 목표로 만들어진 SQL 질의 엔진이다. 하둡과 에이치베이스에 저장된 데이터를 대상으로 SQL 질의를 실행 할 수 있다. 고성능을 낼 수 있도록 자바대신 C++언어를 이용하였으며, 맵리듀스를 사용하지 않고 실행 중에 최적화된 코드를 생성하여 데이터를 처리한다.

오픈소스 임팔라의 주요 구성 요소는 다음과 같다.

① 클라이언트: ODBC, JDBC 클라이언트, 임팔라쉘 같이 임팔라에 접속해서 테이블 관리, 데이터 조회 등의 작업을 수행한다.

② 메타스토어: 임팔라로 분석할 대상 데이터들의 정보를 관리하며, 하이브의 메타데이터를 같이 사용한다.

③ 임팔라 데몬: 시스템에서는 ImpalaD 로 표시되며, 클라이언트의 SQL 질의를 받아서 데이터 파일들의 읽기/쓰기 작업을 수행한다. 임팔라 데몬은 내부적으로 질의 실행계획기, 질의 코디네이터, 질의 실행엔진으로 구성된다.

④ 스테이트스토어: 임팔라 데몬들의 상태를 체크하고 건강(health)정보를 관리해주는 데몬이다. 임팔라데몬에 장애가 생겼을 경우, 다른 데몬들에게 이 사실을 알려줘서 이후부터는 장애가 발생한 데몬에게는 질의 요청이 가지 않도록 해준다.

⑤ 스토리지: 분석 할 데이터를 저장하는 현재는 에이치베이스, 하둡분산파일시스템 두 가지를 지원한다.

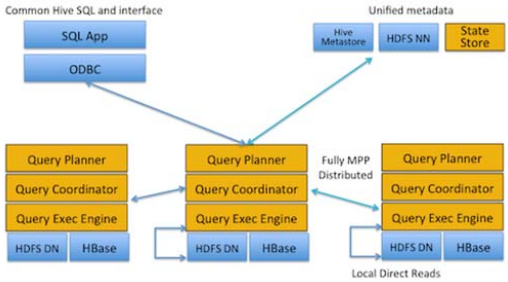

그림 4. 임팔라 아키텍처

2. 임팔라 동작 방식

모든 노드에 임팔라 데몬이 구동되며, 사용자는 이 데몬들이 구동된 임의의 노드에 JDBC나 ODBC또는 임팔라쉘을 이용하여 질의를 요청할 수 있다. 그러면 사용자의 질의는 데이터의 지역성을 고려해서 노드 전체로 분산되어서 수행된다. 하둡에서 잡트랙커(JobTracker)가 데이터 지역성을 고려해서 태스크를 태스크트랙커(TaskTracker)에게 할당하는 것과 유사한 방식이다. 사용자의 질의 요청을 받은 코디네이터 데몬은 분산되어 수행된 각 임팔라 노드들의 부분 결과들을 취합하여 결과 값을 만들어서 사용자에게 제공한다. 실제 운영환경에서는 라운드로빈 방식으로 사용자 질의를 분산시켜서 질의에 대해 전 노드들이 코디네이터 역할을 고르게 수행할 수 있도록 해야 한다.

2.1. 임팔라와 하이브

임팔라는 하이브의 SQL 언어와 메타스토어를 사용한다. 그래서 기존의 하둡상에서 하이브를 이용하여 SQL 질의를 한 사용자라면 쉽게 임팔라를 사용할 수 있다. 또한 하이브의 메타스토어를 참조하기 때문에, 하이브가 생성한 테이블 정의와 하둡상에 저장된 데이터를 사용해서 데이터를 바로 분석 할 수 있다. 임팔라는 속도 향상을 위해서 메타데이터를 메모리로 캐싱하는데, 하이브에서 테이블의 정의가 변경되더

라도 자동적으로 임팔라에게 반영되지 않는다. 이때는전체 테이블 정보를 업데이트 할 경우 Refresh, 일부만 업데이트 할 경우 Refresh <Table Name> 명령을 이용하여 변경된 메타 정보를 반영할 수 있다.

2.2. 임팔라와 하둡

임팔라에서는 하둡분산파일시스템을 데이터 저장매체로 사용하며 맵리듀스는 사용하지 않는다. 하둡분산파일시스템의 주요 속성인 데이터 확장성, 고가용성, 고성능 입출력이라는 장점이 임팔라에게도 그대로 적용된다. 사용자의 SQL 질의를 받은 임팔라 데몬은 하둡분산파일시스템의 마스터노드인 네임노드에게 데이터 블록들의 위치를 파악하여 각 임팔라노드들이 로컬 데이터노드로부터 데이터를 읽을 수 있도록 스케쥴링을 한다. 데이터 포맷은 하둡이 지원하는 파일 포맷인 압축파일, 텍스트파일, 시퀀스파일, 에이브로(AVRO) 파일 형식 등을 지원한다.

2.3. 임팔라와 에이치베이스(HBase)

에이치베이스는 데이터 저장소로 하둡을 사용하는 오픈소스 노에스큐엘(NoSQL) 솔루션이다. 대용량 데이터를 실시간으로 조회하고 저장할 수 있지만, SQL 기반의 질의 기능은 제공하지 않는다. 임파라를 이용하면 에이치베이스에 저장된 데이터를 SQL 질의를 통해서 조회하고 분석 할 수 있다. 임팔라는 에이치베이스의 클라이언트 API를 호출하기 위해서 자바네이티브인터페이스(Java Native Interface)를 사용한다. 임팔라가 에이치베이스의 데이터를 읽기 위해서는 먼저 하이브의 테이블을 생성한 다음에, 아래 그림처럼 하이브 테이블과 에이치베이스 테이블간의 매핑 작업을 해줘야 한다.

```
hive> create 'sample', 'bools', 'ints', 'floats', 'strings'
hive>enable 'sample'
```

```
hive> CREATE EXTERNAL TABLE sample (
      id int,
      bool_col boolean,
      tinyint_col tinyint,
      smallint_col smallint,
      int_col int,
      bigint_col bigint,
      float_col float)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES (
      "hbase.columns.mapping" =
      ":key,bools:bool_col,ints:tinyint_col,ints:smallint_col,ints:int_col,ints:
      bigint_col,floats :float_col
)
TBLPROPERTIES("hbase.table.name" = "sample");
```

그림 5. 하이브와 에이치베이스간 테이블 매핑 사례

3. 임팔라의 SQL 구문

임팔라는 기본적으로 하이브의 SQL을 이용한다. 하지만 임팔라가 모든 하이브 SQL문을 지원하는 것은 아니기 때문에 어떤 구문이 지원되는지 확인이 필요하다. 현재 버전의 임팔라에서 지원되는 주요 SQL 구문과 기능은 표 1과 같다.

표 1. 임팔라 SQL 구문

항목	설명
데이터 정의 언어 (Data Definition Language)	데이터베이스/테이블 생성: Create Database/Table
	테이블 변경/파티션 추가: Alter Table
	데이터베이스/테이블 삭제: Drop Database/Table
	데이터베이스 테이블 조회: Show Database/Table, Describe Database
데이터 조작 언어 (Data Manipulation Launguage)	데이터 조회: Select, Where, GroupBy, OrderBy 구문 지원
	데이터 입력: Insert into/overwrite 구문 지원
	데이터 변경 구문은 지원 안함
	데이터 삭제(Delete) 구문은 지원 안하나 테이블 삭제(Drop)시 데이터가 삭제됨
내장 함수 (Builtin Functions)	수학함수: 절대값(abs) 반환, 코사인값 반환(acos), 로그값 반환(log)등의 기능 제공
	타입변환: 날짜값 반환(day), 유닉스에포타임 변환(from_unixtime), 현재시간 반환(now)등 다수의 함수 제공
	조건문: if문 제공, case 등 분기 기능 제공
	문자열함수: 아스키코드값 변환(ascii), 문자열병합(concat), 정규표현식(regexp)

4. 임팔라 데이터모델

임팔라는 하둡분산파일시스템에 데이터를 저장한다. 어떤 저장 포맷을 사용하느냐에 따라 데이터 조회 시 처리 성능이 달라진다. 하둡의 기본 파일 포맷인 텍스트나 시퀀스파일은 로우단위의 데이터 저장 방식을 사용한다. 컬럼단위의 파일 저장 포맷인 RCFile을 사용할 경우, 데이터 처리 과정에서 발생하는 디스크 입출력의 양을 현저하게 줄일 수 있다. 로우단위로 저장 시, 테이블에서 하나의 컬럼을 읽던지, 전체 테이블을 읽던지 동일한 디스크 입출력이 발생한다. 반면 컬럼단위의 저장 포맷을 사용할 경우, 읽고자하는 컬럼만큼의 디

스크 입출력이 발생하기 때문에 처리 성능을 개선할 수 있다. 물론 전체 컬럼들을 모두 조회하는 질의는 저장 포맷에 의해 성능이 영향을 받지는 않는다.

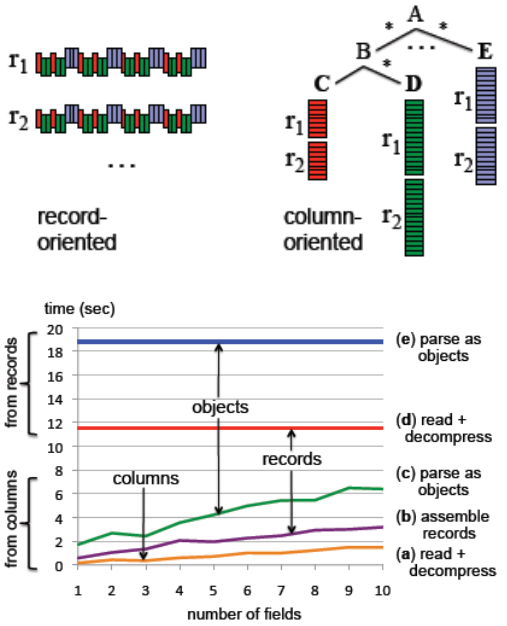

그림 6. 로우와 컬럼단위 저장 방식의 성능 비교
(출처 구글 드레멜 논문)

그림 6의 (a), (b), (c)는 컬럼 파일 포맷에서 무작위로 선택된 컬럼의 개수에 따른 처리 시간이며 (d), (e)는 기존의 로우 단위 파일 포맷을 사용해서 읽을 때 걸리는 시간이다. 테스트 결과에 의하면, 컬럼 파일 포맷을 사용했을 때 처리 시간이 적게 걸린다. 적은 수의 컬럼을 읽을수록 훨씬 빠른 처리 성능을 보여주고 있다. 반면 기존 방식인 로우 파일 포맷을 사용해서 레코드를 읽을 때는 하나의 컬럼에 접근해도 항상 전체를 읽는 것과 같은 처리 시간을 보여주고 있다. 컬럼 파일 포맷을 사용하는 것이 효율적이라는 것은 증명이 된 셈이다. 다만 하둡 상에 저장된 파일이 처음부터 컬럼 파일 포맷을 사용하지 않았을 경우, 파일 포맷을 변경 작업을 해줘야 한다.

5. 성능 비교

성능 테스트는 자바로 작성한 맵리듀스 작업, 하이브 작업, 임팔라 작업 등 3가지 유형으로 진행하였으며, 파일 포맷은 일반 텍스트와 컬럼 파일 포맷인 RCFile을 사용했다. 성능 비교 테스트에 사용된 서버는 12대이며 각 서버 사양을 포함한 테스트 환경은 다음과 같다.

- **서버 사양**
  - 프로세서: 4Cores * 2CPU(인텔 ZEON)
  - 메모리: 32GB
  - 네트워크: 1Gbps * 2Ports
  - 디스크: 1TB * 8개, 7200RPM SATA2, JBOD
  - 운영체제: Centos 6.2 64비트

- **테이블 구성 정보(하이브)**
  - 입력 테이블 input은 35개의 컬럼들로 구성되며, 데이터는 그림 7과 같이 날짜(data1)-시간(time)-타입(type)-위치(loc)의 4단계 파티션 들로 나뉘어 저장된다. 결과 테이블 output은 17개의 컬럼들로 구성되어 있다.

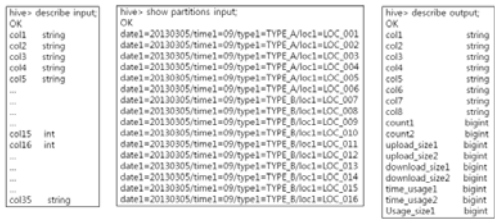

그림 7. 테이블/파티션 구성 정보

- **작업 정보**
  - 맵리듀스 작업은 35개의 컬럼들로 이루어진 텍스트 데이터 약 96GB를 맵퍼에서 읽은 다음, 이 중 약 8개의 컬럼들을 추출하며 리듀서로 보낸다. 리듀서는 8개의 컬럼 값들을 조합하여 9개의 집합 연산 결과 값을 만들어 낸다. 맵리듀스 작업의 결과 데이터의 크기는 약 500MB 정도이다.
  - 하이브 작업은 맵리듀스 로직을 그림 8같은 유형의 SQL 문으로 작성하여 실행하였다.

```
SELECT
    col, col2, col3, ... ... ...
    , sum(decode(col10,'0',... ...) ... ... ... ...) AS count1
    , sum(decode(col11,'1',... ...) ... ... ... ...) AS count2
    , sum(decode(col12,'0',... ...)... ... ... ...) AS upload_size1
    , sum(decode(col13,'0',... ...)... ... ... ...) AS upload_size2
    , sum(decode(col14,'0',... ...)... ... ... ...) AS download_size1
    , sum(decode(col15,'0',... ...)... ... ... ...) AS download_size2
    , sum(decode(col16,'0',... ...)... ... ... ...) AS time_usage1
    , sum(decode(col17,'0',... ...)... ... ... ...) AS time_usage2
    , sum(decode(col18,'0',... ...)... ... ... ...) AS usage_size1

FROM input

WHERE col1='APP' AND col10+col11 > 0 AND col2='20130501' AND col3 IS NOT NULL

GROUP BY col1, col2, col3 .....
```

그림 8. 하이브 SQL 질의문

- 임팔라 작업은 하이브 작업의 SQL문과 거의 유사하나, 현재의 임팔라 버전에서는 하이브 SQL 문에 있는 'decode' 같은 일부 사용자 정의 함수(User Defined Function)을 사용할 수 없다. 따라서 이 부분을 대체 SQL 구문으로 수정을 했으며, 이외에는 모두 동일하다.

```
SELECT
    col, col2, col3, ... ... ...
    , sum(CASE WHEN col10 NOT IN ('1','2') THEN  1 ELSE 0 END) AS count1
    , sum(CASE WHEN col11 NOT IN ('1','2') THEN  1 ELSE 0 END) AS count2
    , sum(CASE WHEN col12 ... ... ... ...) AS upload_size1
    , sum(CASE WHEN col13 ... ... ... ...) AS upload_size2
    , sum(CASE WHEN col14 ... ... ... ...) AS download_size1
    , sum(CASE WHEN col15 ... ... ... ...) AS download_size2
    , sum(CASE WHEN col16 ... ... ... ...) AS time_usage1
    , sum(CASE WHEN col17 ... ... ... ...) AS time_usage2
    , sum(CASE WHEN col18 ... ... ... ...) AS usage_size1

FROM input

WHERE col1='APP' AND col10+col11 > 0 AND col2='20130501' AND col3 IS NOT NULL

GROUP BY col1, col2, col3 .....
```

그림 9. 임팔라 SQL 질의문

각 작업별 수행 시간은 그림 10과 같다.

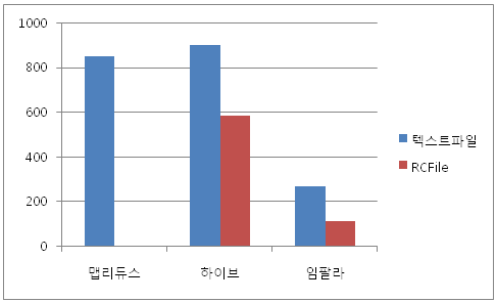

그림 10. 작업 유형 별 처리 시간

일반 텍스트 파일을 대상으로 성능 비교를 했을 때, 맵리듀스 작업 수행 시간은 850초, 하이브 작업은 901초, 임팔라 작업은 271초 걸렸다. 컬럼 파일(RCFile)을 대상으로 작업 수행 시 하이브는 약 585초, 임팔라는 110초 정도 걸렸다. 하이브의 질의 실행 엔진이 최적화 되었기 때문에 자바 코드로 작성한 맵리듀스 작업 대비 처리 성능적인 차이는 거의 없었다. 임팔라는 하이브 대비 처리 성능이 약 5~6배 정도 빨랐으며, 분석 대상 입력의 파일의 크기가 작을 경우, 본 테스트 결과보다 훨씬 더 빠른 처리 성능이 나올 것으로 예상된다.

IV. 결 론

하이브는 하둡에 저장된 다양한 형태의 대용량 데이터를 효율적으로 처리하고 분석하는 표준 SQL솔루션으로 채택되고 있지만, 처리 속도로 인하여 실시간이 아닌 주로 배치처리에 사용되고있다. 임팔라와 같은 기술은 이러한 하이브의 제약을 극복한 기술이라고 할 수 있으며, 준 실시간으로 하둡상의 대용량 데이터를 SQL 질의어로 분석할 수 있는 솔루션이라고 할 수 있다. 운영 시스템에 적용하기 위해서는 사용자 함수 지원, 데이터 조인 시 메모리 문제 등 여러가지로 개선해야 할 사항들이 아직 많다. 하지만 중요한 빅데이터 기술 트랜드로 최근 큰 관심을 받고 있고, 많은 개발자가 개발과 테스트에 참여하고 있기 때문에 점차적으로 기능과 안정성이 개선될 것으로 보이며, 향후 빅데이터 분석에 중요한 축을 담당할 것으로 예상된다.

참고문헌

[1] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, "Dremel: Interactive Analysis os Web-Scale Datasets" Proc. of the 36th Int'l Conf on Very Large Data Bases (2010), pp. 330-339

[2] Jeffrey Dean, Sanjay Ghemawat "MapReduce: Simplified Data Processing on Large Clusters" OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

[3] 심탁길, 김우현, "하둡 완벽 가이드," *한빛 미디어*, 100-120쪽, 2013년.

[4] 클라우데라 웹사이트, http://www.cloudera.com

[5] 아파치 하둡 웹사이트, http://hadoop.apache.org

[6] 아파치 하이브 웹사이트, *http://hive.apache.org*

저자소개

심 탁 길

1999: 숭실대학교
전자계산학과 공학사.
2006: 한양대학교
컴퓨터공학과 공학석사.
현 재: SK C&C 부장,
수석아키텍트
관심분야: 빅데이터/클라우드