정형 빅데이터 수집 모듈 구현 및 비교

*장동훤, 이민우, 김우생
광운대학교 전자정보공과대학 컴퓨터소프트웨어학과
e-mail : hwonttl@gmail.com

Implementation and comparison with Structured data collection modules

*Dong-Hwon Jang, Min-Woo Lee, Woosaeng Kim
School of Computer Software
KwangWoon University

요 약

빅데이터 시대의 대두에 따라 기존의 관계형 데이터베이스로는 처리하기 어려운 형태의 데이터가 발생하였다. 이런 성질의 데이터를 저장, 활용하기 위한 방법으로 Apache 하둡이 널리 사용되고 있다. 기존의 RDBMS 상의 데이터를 하둡 데이터 분석의 원천 데이터로 활용하려고 하는 경우, 혹은 데이터 크기와 복잡도의 증가로 저장 방식을 바꿔야 하는 경우 데이터를 HDFS(Hadoop Distributed File System)으로 전송해야 한다. 본 논문에서는 정형 데이터 수집 모듈인 Sqoop과 Nosqoop4u의 개발을 통하여 데이터 전송 성능을 비교하였다.

1. 서 론

인터넷과 컴퓨터 정보 처리 기술의 발달, 무엇보다 SNS(Social Network Service)의 등장으로 인해 '빅데이터' 라는 용어가 현대 사회의 키워드로 떠올랐고, 이에 따라 대용량이며 높은 복잡도를 가지는 데이터의 저장 및 활용, 분석 기술 수요가 나날이 증가하는 추세를 보이고 있다.

가장 많이 채택되는 데이터 저장 및 운용 방법 중의 하나였던 RDBMS는 비용, 처리 속도 문제 등 대용량 데이터를 다루기 어렵다는 한계에 부딪혔고, 이를 처리하기 위해서 기존과 다른 형태의 데이터 저장 방식이 필요하다. 이러한 빅데이터를 저장, 활용하기 위해 하둡의 HDFS가 널리 사용되고 있으며, 기존 데이터의 크기와 복잡도가 증가하여 저장 방식을 바꿔야만 하거나, 데이터 분석을 위한 원천 데이터로 활용할 목적으로 RDBMS상의 데이터를 하둡 HDFS로 전송 시켜야만 하는 경우가 발생하였고, 이를 위한 데이터 전송 모듈이 등장하였다.

본 논문에서는 이러한 데이터 전송 모듈 중 대용량 데이터 전송 솔루션인 Sqoop과 Nosqoop4u의 설계와 기능에 대하여 기술하고[1,2], 구현을 통하여 데이터 전송 기능의 성능을 비교한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 소개하고, 3장에서 데이터 전송 모듈인 Sqoop과 Nosqoop4u의 시스템 설계에 대해서 설명한다. 4장에서는 두가지 모듈을 구현하고 성능을 비교한다. 마지막 5장은 본 논문에 대한 결론과 향후 연구 방향에 대하여 기술한다.

2. 관련 연구

하둡 에코시스템은 아래의 그림 1과 같이 구성된다[3,4]. 정형 데이터 수집 모듈인 Sqoop은 대용량 데이터 전송 솔루션이며, HDFS, RDBMS, DW, NoSQL 등 다양한 저장소에 대용량 데이터를 신속하게 전송할 수 있는 방법을 제공한다. Hiho는 Sqoop과 같은 대용량 데이터 전송 솔루션이며, Hadoop에서 데이터를 가져오기 위한 SQL을 지정할 수 있으며, JDBC 인터페이스

를 지원한다. 결과 데이터 액세스 모듈인 HBase는 HDFS 기반의 비관계형 분산 데이터 베이스이다. 실시간 랜덤 조회 및 업데이트가 가능하고 각각의 프로세스들은 개인의 데이터를 비동기적으로 업데이트할 수 있다. 데이터 저장 모듈인 HDFS는 마스터, 슬레이브 구조의 분산파일 시스템이고 데이터 처리 모듈인 Map Reduce는 분산환경에서 병렬처리를 하기 위한 시스템이다. Pig는 대용량 데이터 집합을 분석하기 위한 플랫폼으로 높은 수준의 스크립트 언어로 구성되어 있다. Hive는 Hadoop에서 동작하는 데이터 웨어하우스 인프라 구조로서 데이터 요약, 질의, 분석 기능을 제공한다. 분산 코디네이터인 Zoopeeker는 분산 환경에서 서버들간에 상호 조정이 필요한 다양한 서비스를 제공하는 시스템이다. Avro는 데이터 직렬화를 지원하는 프레임워크이다. 데이터 분석과 마이닝에 사용되는 Mahout은 Hadoop 기반으로 데이터 마이닝 알고리즘을 구현한 오픈 소스이다. 워크 플로우 관리 모듈인 Oozie는 Hadoop 작업을 관리하는 워크 플로우 스케줄러 시스템으로 Map Reduce 작업이나 Pig 작업 같은 특화된 액션들로 구성된 워크 플로우를 제어한다.

그림 1. 하둡 에코시스템

3. 설 계

Sqoop Import 동작에 관한 설계는 그림 2와 같다. 전송되어질 데이터셋은 맵퍼의 개수에 따라 여러 파티션으로 분할되며, 각 맵퍼가 담당하는 레코드 수는 RDBMS 테이블의 기본 키를 이용하여 판단한다. 각각의 맵퍼가 나누어진 파티션들에 대해 Map 작업을 병렬적으로 수행하고, 이렇게 만들어진 결과 데이터들은 설정한 데이터 저장방식(HDFS, HBase, Hive)에 맞게 변형되어 목표 저장소에 저장되도록 설계 했다[1,2].

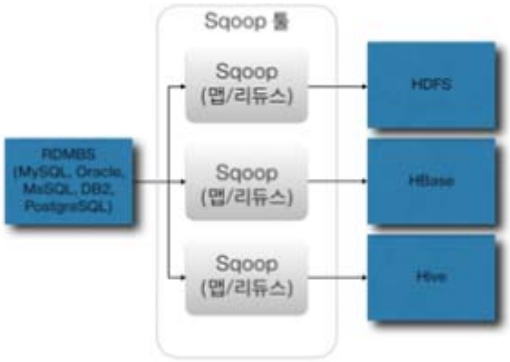

그림 2. Sqoop Import의 동작

Nosqoop4u의 동작에 관한 설계는 그림 3과 같다. Nosqoop4u의 동작은 Sqoop에 비해서도 더욱 간단하여, Sqoop에서 이루어지는 파티션 분할 및 Map 작업을 수행하지 않아 병렬 처리를 할 수는 없다. 단순히 JDBC API를 통해 RDBMS와 연결을 생성하고, 입력한 SQL 질의에 대한 응답을 지정한 위치에 저장한다. 본 논문에서는 MySQL 상에 테이블을 임의로 만들어 소스 데이터로 사용할 수 있도록 하였고, Sqoop의 맵퍼의 수는 1개, 출력 위치는 HDFS로 설계했다.

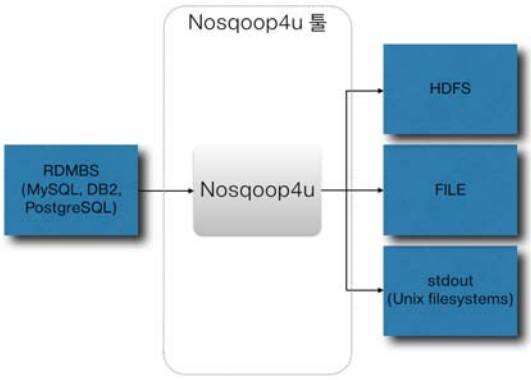

그림 3. Nosqoop4u의 동작

4. 구현 및 성능 비교

4. 1. Sqoop

Sqoop은 JDBC API를 사용하기 때문에 우선 JDBC 드라이버를 설치하였다. Sqoop과 JDBC 모두 설치가 완료되었다면 간단한 Sqoop 환경 설정이 필요하다. 리눅스 상의 .bash_profile 파일은 표1과 같다.

표 1. Sqoop 환경 설정

```
# Sqoop
export HADOOP_COMMON_HOME="Hadoop directory"
```

```
export HADOOP_MAPRED_HOME="Hadoop directory"
export SQOOP_HOME="Sqoop directory"
export PATH=$PATH:$SQOOP_HOME/bin
```

두 번째로 설치된 JDBC 드라이버 mysql-connector-java-5.1.17.jar 파일을 Sqoop이 설치된 폴더의 하위 폴더인 lib에 복사해 실행을 위한 환경설정을 한다. Sqoop은 하둡의 맵리듀스 기능을 사용하므로 하둡이 기동되고 있는 상태에서만 동작시킬 수 있다. Sqoop Import의 실행에 사용한 인자는 표2와 같다.

표 2. Sqoop Import의 사용

```
$ sqoop import
--connect     #JDBC 연결 주소
--table       #읽을 테이블
--username    #인증 사용자명을 설정
--password    #인증 비밀번호를 설정
-m,--num-mappers      #병렬로 가져오기 위하여
                       n개의 맵 태스크를 사용
```

4. 2. Nosqoop4u

Nosqoop4u는 Sqoop과 다르게 하둡의 맵리듀스에 의존하지 않으므로 하둡이 실행되고 있을 필요는 없지만, HDFS에 파일을 전송할 경우엔 하둡이 실행되고 있어야 한다. 본 논문에서는 jruby 1.6.4 버전을 사용하였다. Nosqoop4u의 실행에 필요한 파라미터로는 파일의 출력 형식과 그 위치, JDBC URL, DB 사용자 이름, DB 패스워드, JDBC 드라이버 클래스, 출력될 내용을 위한 쿼리문, 필드 별로 데이터를 구분해줄 구분자, fetch 사이즈가 있다. 실제 동작을 위해 사용한 인자는 표3과 같다.

표 3. Nosqoop4u의 사용

```
nosqoop4u options
  -o, --output       # 출력 위치(HDFS, FILE, stdout)
  -c, --connect url # jdbc 연결 url
  -u, --user         # db 사용자 이름
  -p, --pass         # db 사용자 패스워드
  -d, --driver       # JDBC 드라이버 클래스
  -e, --query        # sql 질의
  -F, --delim        # 구분자 (default: ^A)
  -f, --fetch        # fetch 크기
```

4. 3. 성능 비교

그림 3은 동일 환경에서 성능 비교를 위해 임의로 작성한 2가지 속성의 약 10만 레코드가 삽입되어있는 MySQL상의 테이블을 전송한 결과이다. 테스트는 단일머신으로 실행되었지만 보다 실제 완전 분산 환경과 비슷한 결과를 얻기 위해 하둡을 가상 분산 모드로 설정하였다. 가상 분산 모드는 단일머신 상에서 완전 분산 모드와 비슷한 동작을 가능하게 해준다. 실행 환경의 OS는 Ubuntu Linux LTS 12.04 32bit 이며, CPU는 AMD Athlon Dual core 4800 2.5GHz, RAM은 2 GB, 하둡은 1.0.3 버전을 사용하였다. 테스트를 위해 임의로 만든 MySQL 테이블의 용량은 517KB이다. Sqoop은 1.4.4, Nosqoop4u는 0.1.2 버전을 사용하였다. 테스트 결과 Sqoop은 29.7초가 걸린데 비해 Nosqoop4u는 5.1초밖에 걸리지 않았다.

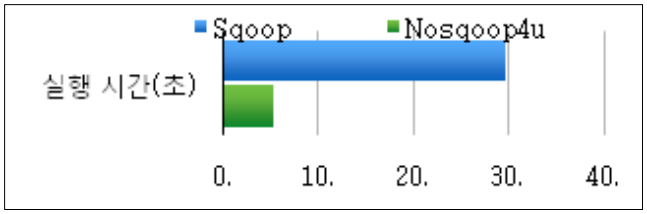

그림 3. Sqoop과 Nosqoop4u의 성능 비교 결과

표 4는 Sqoop과 Nosqoop4u의 기능 차이를 정리한 내용이다. Sqoop은 지원하는 RDBMS의 종류, Export 기능, HBase와 Hive 통합 지원, 맵리듀스를 통한 병렬 처리 등 Nosqoop4u에 비해 상당히 많은 기능을 가지고 있다. 반면 Nosqoop4u는 맵리듀스를 사용하지 않고 작동 방식 또한 단순하여 완전 분산 환경이 아닌 단일 머신 상에서의 데이터 전송 성능은 우수하다.

표 4. Sqoop과 Nosqoop4u의 차이점

	Sqoop	Nosqoop4u
지원 RDBMS	MySQL, MsSQL, Oracle, PostgreSQL, DB2	MySQL, PostgreSQL, DB2
전송 기능	RDMBS - HDFS 간 양방향 데이터 전송	RDBMS -> HDFS 단방향 데이터 전송
HBase, Hive와 통합	지원	미지원
맵리듀스 기능 사용여부	사용	미사용
전송 데이터	특정 테이블 혹은 데이터베이스의 모든테이블	쿼리에 대한 응답결과

5. 결론

단지 기존 RDBMS의 소규모 데이터 전송이 목적이라면 동작이 단순한 Nosqoop4u를 채택하는 것이 좀 더 빠르게 결과를 얻을 수 있었다. 양방향 전송 등 다양한 기능이 활용되어야하는 환경이거나 이미 하둡을 완전 분산 모드로 운용하고 있어 매우 큰 용량의 데이터를 하둡의 병렬 처리 기능을 이용하여 효율적으로 전송해야 하는 경우 Sqoop을 사용하는 것이 좋다. 향후에는 두 모듈을 이용해 수집한 정형 데이터를 분석해보고자 한다.

감사의 글

본 논문은 2014년도 KWIX(KwangWoon IT Exhibition)로 지원 되었습니다.

참고 문헌

[1] http://sqoop.apache.org/docs/1.4.0-incubating/SqoopUserGuide.html

[2] http://www.ibm.com/developerworks/library/bd-sqoopconduit

[3] 정재화, “시작하세요. 하둡 프로그래밍”, 위키북스, 2012.

[4] 한기용, “Do it 직접해보는 하둡 프로그래밍”, 이지스퍼블리싱, 2012.