JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 21, No. 11 : 2067~2075 Nov. 2017

분산병렬처리 환경에서 오토매핑 기법을 통한 NoSQL과 RDBMS와의 연동

김희성[1] · 이봉환[2*]

Interoperability between NoSQL and RDBMS via Auto-mapping Scheme in Distributed Parallel Processing Environment

Hee Sung Kim[1] · Bong Hwan Lee[2*]

[1]Analytics & Decision Department, KSTEC Inc.
[2*]Department of Electronics, Information and Communications Engineering, Daejeon University, Daejeon 34520, Korea

요 약

최근 빅데이터가 주목받게 되면서 빅데이터를 처리하기 위한 시스템들도 중요하게 여겨지고 있다. 빅데이터 처리 시스템으로 분산파일시스템인 Hadoop과 비정형 데이터 처리를 위한 NoSQL 데이터 스토어가 주목받고 있다. 하지만 아직까지 NoSQL을 사용함에 있어 어려움이나 불편함도 존재한다. 저용량 데이터인 경우 NoSQL의 MapReduce는 불필요한 작업시간을 소모하게 되며, RDBMS 보다 상대적으로 많은 데이터 탐색 시간이 소요되기도 한다. 본 논문에서는 이러한 NoSQL의 문제점을 해결하기 위해 NoSQL과 RDBMS 간의 연동 기법을 제안하였다. 개발한 오토매핑 기법은 처리할 데이터의 양에 따라 적합한 데이터베이스를 사용하게 하여 결과적으로 검색시간을 빠르게 할 수 있다. 실험 결과 제안한 데이터베이스 연동 기법은 특정 데이터 셋의 경우 검색시간을 최대 35%까지 줄일 수 있다.

ABSTRACT

Lately big data processing is considered as an emerging issue. As a huge amount of data is generated, data processing capability is getting important. In processing big data, both Hadoop distributed file system and unstructured date processing-based NoSQL data store are getting a lot of attention. However, there still exists problems and inconvenience to use NoSQL. In case of low volume data, MapReduce of NoSQL normally consumes unnecessary processing time and requires relatively much more data retrieval time than RDBMS. In order to address the NoSQL problem, in this paper, an interworking scheme between NoSQL and the conventional RDBMS is proposed. The developed auto-mapping scheme enables to choose an appropriate database (NoSQL or RDBMS) depending on the amount of data, which results in fast search time. The experimental results for a specific data set shows that the database interworking scheme reduces data searching time by 35% at the maximum.

키워드 : 하둡, NoSQL, RDBMS, 빅데이터, 오토매핑

Key word : Hadoop, NoSQL, RDBMS, Big data, Auto-mapping

Received 16 August 2017, **Revised** 13 October 2017, **Accepted** 14 October 2017

* **Corresponding Author** Bong Hwan Lee(E-mail:blee@dju.kr, Tel:+82-42-280-2553)
Department of Electronics, Information and Communications Engineering, Daejeon University, Daejeon 34520, Korea

Open Access https://doi.org/10.6109/jkiice.2017.21.11.2067 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

인류 탄생으로부터 2003년 까지 인간의 문명이 만들어낸 데이터 양은 약 5엑사바이트(약 5000TB)로 추정된다. 하지만 오늘날에는 하루에 약 8~10엑사바이트가 생성된다. 즉, 지난 수 천년간의 데이터 양 보다 오늘날 하루에 생성되는 데이터가 훨씬 많다는 것이다. 데이터 증가와 함께 대두되는 문제는 이러한 많은 양과 다양한 종류의 데이터를 처리하는 기술이다. 많은 기업이나 국가기관들은 '빅데이터(Big data)'라 불리는 방대한 양의 데이터에 초점을 두고, 빅데이터 처리와 분석 및 저장 기술 개발에 노력하고 있다.

대용량 데이터 처리에 있어서 중요하게 여겨지는 부분은 방대한 데이터를 저장하는 데이터 스토어이다. 기존의 RDBMS(관계형데이터베이스)는 대량의 데이터를 처리할 경우에 성능에 문제가 발생할 수 있고, 컬럼을 정의하는 데에 있어서 빅데이터 처리에는 한계가 나타나는 것으로 드러났다. 빅데이터는 데이터의 양이 많은 만큼 정형화 되어 있다고 보기 힘들고, 그로 인해 기존의 RDBMS로 처리하는데 한계가 있다는 것이다[1, 2]. 이러한 문제점을 보완하기 위해 데이터의 연관성과는 상관없는 비정형화된 데이터를 처리하기 위한 데이터베이스가 필요했고, NoSQL이라는 데이터스토어가 주목을 받게 되었다. NoSQL은 데이터의 일관성 보다 가용성과 확장성에 중점을 두고 유동적인 스키마 처리를 통해 방대한 데이터 처리에 유리하도록 설계된 데이터 저장기술이다.

본 논문에서는 분산병렬처리 환경에서 대용량 로그(Log) 데이터 분석 시에 NoSQL의 제한적인 문제를 해결하고자 NoSQL과 RDBMS를 연동하는 오토매핑 방식을 제안하고, 데이터의 종류와 용량에 따른 성능을 평가해 보고자 한다.

Ⅱ. 관련 연구

2.1. 로그 수집 툴 Flume

Flume은 여러 서버(agent)에 있는 로그들을 하나의 수집서버(collector)로 모으는 역할을 수행하는 로그 수집기이다. Flume은 지정된 서버로부터 로그를 수집하여 Hadoop과 같은 중앙 저장소에 적재하여 분석할 때에 많이 사용된다.

Flume은 분산된 구조로 확장할 수 있기 때문에 데이터양이 증가해도 기존 agent의 설정이나 구조를 변경하지 않고 쉽게 확장할 수 있다는 장점이 있다. 또한, 이기종 디바이스에 탑재가 가능하고 Hadoop 시스템과 호환성이 높다. Flume은 수많은 논리 노드로 구성되어 있으며 크게 agent와 collector로 구분할 수 있다. Agent 노드는 일반적으로 로그를 생산하는 디바이스 혹은 시스템에 설치된다. 이 노드는 데이터의 초기 시작점으로 Flume에 설정되며, collector 노드로 데이터를 전송한다. Collector 노드는 분산된 데이터에서 agent로부터 데이터를 수집하고 HDFS에 데이터를 전송하여 저장한다.

예전에는 collector와 agent를 따로 구분하였지만 최근에는 모든 서버를 agent라 통칭한다. Agent는 기본적으로 source, channel, sink로 구성되어 있다. Source는 지정한 source를 통해서 이벤트를 받아 channel로 전달하는 역할을 하며, channel은 메모리, 파일, DB 등의 이벤트를 임시로 보관한다. Sink는 channel에서 이벤트를 읽어와 출력 대상에 작성하는 역할을 하며, sink가 이벤트의 전달을 완료하면 channel에서는 이 이벤트를 삭제한다. Agent간 연결에 있어서 sink는 다른 agent의 source로 연결되며, 다수의 agent sink가 하나의 agent source로 연결될 수 있다. 또한, 하나의 source는 다수의 channel로 나뉘어 각각 다른 목적지로 이벤트의 전달이 가능하다.

Source는 받은 이벤트를 replicating과 multiplexing이라는 절차를 거쳐 다수의 channel에 전달할 수 있는데, replicating의 경우 하나의 source에서 다수의 channel로 모든 값을 전달하며, multiplexing은 헤더 값을 기준으로 선택적으로 다수의 channel에 이벤트를 전달할 수 있다.

2.2. 분산병렬처리 시스템

2.2.1. HDFS - Hadoop Distributed File System

HDFS는 수십 TB(테라바이트) 혹은 PB(페타바이트) 이상 대용량의 데이터를 분산된 서버에 저장하고 다수의 클라이언트로 저장된 데이터 처리의 속도를 향상시킬 수 있도록 설계된 파일 시스템이다. HDFS에 저장된 데이터는 물리적으로는 분산되어 있고, HDFS에서 제공하는 API를 이용하면 분산된 서버의 로컬 디

스크를 통해 파일의 읽기 및 저장과 같은 제어를 처리할 수 있다.

HDFS는 하나의 네임노드(name node)와 다수의 데이터노드(data node)로 구성되어 있다. 네임노드는 HDFS의 디렉토리와 파일 정보를 유지하고 있다. 즉, 자체에 데이터를 저장하기 보다는 HDFS를 구성하는 파일, 디렉토리 등의 메타 데이터를 메인 메모리에 유지함으로써 HDFS를 전반적으로 관리하는 역할을 한다[3, 4]. 또한 데이터노드는 실제로 분산된 데이터가 저장되어 있는 노드로 네임노드에서 관리하는 HDFS 정보에 따라 물리적으로 데이터를 저장한다. HDFS에서는 데이터 저장을 위해 블록 단위를 이용한다. 블록들이 어떠한 데이터노드에 있는지에 대한 정보는 네임노드에서 관리하며, 저장된 데이터가 필요한 경우 각 데이터노드의 해당 블록을 통해 사용한다.

2.2.2. MapReduce

MapReduce는 분산 파일을 처리, 분석하기 위한 프레임워크로 Map과 Reduce 두 가지 작업으로 이루어진다. Map은 흩어져 있는 데이터를 key와 value의 형태로 데이터의 연관성을 기준으로 분류하고 묶는 작업이며, Reduce는 Map 작업을 거친 데이터에서 중복 데이터를 제거하고 특정한 데이터를 검출하는 작업을 말한다[5, 6]. Map 작업을 수행하는 모듈을 Mapper, Reduce 작업을 수행하는 모듈을 Reducer라 한다. Mapper는 입력받은 데이터를 분석하여 새로운 key와 value값을 갖는 형태로 구성한 뒤 Output Collector 객체를 통해 반환하며, Record Reader로부터 한 줄씩 데이터를 입력받아 각각의 데이터를 다수의 (key, value) 결과 값으로 생성한다. 또한, Reducer는 key값이 같은 모든 정보들을 Input으로 가지며, 같은 key값을 갖는 모든 데이터를 리스트의 형태로 입력받은 뒤 이들의 조합에서 원하는 결과를 분석한다.

2.3. 데이터스토어

2.3.1. RDBMS – MySQL

RDBMS는 관계형 데이터베이스관리시스템으로 행(row)과 열(column)로 구성되어 있는 테이블들의 관계를 이용해 다양한 요구를 처리하는 데이터스토어를 말한다. 단일 테이블 열람만으로 검색하기 어려운 결과를 테이블들의 삽입, 탐색, 수정, 삭제 뿐만 아니라 조인과 같은 질의(query)를 이용해 데이터를 처리할 수 있게 한다.

MySQL은 오픈 소스 RDBMS로 표준 데이터베이스 질의 언어인 SQL(Structured Query Language)을 사용한다. MySQL은 속도, 유연성, 사용의 용이성 등의 목적으로 개발되었으며, 다중 사용자와 다중 스레드를 지원하고 다양한 언어를 위한 API를 제공한다. 유닉스나 리눅스, 윈도우 등 대부분의 운영 체제를 지원하고 상호 연동이 간편하여 일반적으로 많이 이용되고 있는 데이터베이스이다. MySQL은 DBMS중에서 처리 속도가 상당히 빠른 편이고 설치 및 사용이 쉬우며, 보안이 뛰어나다. 또한, 대용량의 데이터 처리에 있어서도 타 RDBMS 보다 유리하다.

2.3.2. NoSQL 데이터베이스 – HBase

NoSQL은 Not only SQL의 약어로 기존의 관계형 데이터베이스가 대규모의 데이터를 저장하기 위해 스케일 아웃(Scale Out)방식으로 확장시키기 어렵다는 한계를 극복하고 대용량의 데이터를 처리하기 위해 개발되었다. NoSQL은 인덱스와 데이터가 분리되어 별도로 운영되며 스키마에 대한 정의가 자유롭다[7-9]. 스키마에 대한 정의가 자유롭다는 것은 스키마가 유동적이고 키 부분의 타입만 동일하고 값에 해당하는 컬럼의 타입이나 이름 등을 자유롭게 허용하는 방식이다(그림 1).

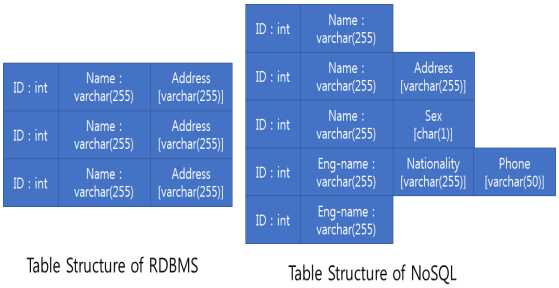

Fig. 1 Table Structure of RDBMS and NoSQL

HBase는 Apache 프로젝트의 산물로 Hadoop 플랫폼을 위한 비관계형 분산 데이터베이스이다. 구글의 빅테이블(BigTable)을 모델로 삼았으며, Java로 작성되어 있고 Hadoop의 분산파일시스템인 HDFS 위에서 동작한다. 본 연구에서는 NoSQL 가운데 Hadoop 플랫폼에서 우수한 성능을 보이는 HBase를 사용한다. 그리고

Zookeeper라는 Apache의 동기화 보장시스템을 이용하여 고 가용성이 보장되며, 무정지 방식을 제공하여 대량의 분산된 데이터를 저장한다.

Ⅲ. MySQL과 HBase를 이용한 Log 분석

3.1. 시스템 구성

3.1.1. 분산병렬처리 시스템(Hadoop) 환경 구현

분산병렬처리는 다수의 노드를 이용해 고성능의 효과를 내기 위한 시스템이며, Scale Out 방식의 노드 확장으로 성능을 향상시킬 수도 있다[10]. 본 논문에서는 1대의 네임노드와 3대의 데이터노드를 이용하여 시스템 환경을 구성했으며, 로그 수집을 위해 1대의 Flume Agent 노드를 구성하였다. 노드들의 사양 및 정보는 표 1과 같다.

Table. 1 System specification of each node

	CPU	RAM	HDD	OS	Tools
Name Node	Quad Core 2.80GHz	16.0GB	1TB	ubuntu 12.04	Flume-1.4.0 Zookeeper-3.4.5 Hadoop-1.2.1 Hbase-0.94 MySQL-5.5.38 R-Hive-2.0
Data Node	Quad Core 2.80GHz	16.0GB	1TB	ubuntu 12.04	Zookeeper-3.4.5 Hadoop-1.2.1 Hbase-0.94
Data Node	Quad Core 2.80GHz	16.0GB	1TB	ubuntu 12.04	Zookeeper-3.4.5 Hadoop-1.2.1 Hbase-0.94
Data Node	Quad Core 2.80GHz	16.0GB	1TB	ubuntu 12.04	Zookeeper-3.4.5 Hadoop-1.2.1 Hbase-0.94
Agent Node	Quad Core 2.80GHz	16.0GB	1TB	ubuntu 12.04	Flume-1.4.0 Snort

네임노드에는 Flume collector를 설치하여 Flume agent로부터 로그를 수집하도록 하였다. 또한, Hadoop의 모든 노드에 HBase를 설치하여 분산데이터베이스 환경을 구성하고 네임노드에는 HBase의 HMaster서버를, 데이터노드에는 HBase의 region 서버를 구성하였다. 네임노드에서 데이터베이스 연동을 통한 성능을 검증하기 위해 MySQL을 설치하였고, 분석 및 처리를 위한 Hive(RHive)도 구성하였다. 그림 2는 본 연구에서 구현한 분산병렬처리시스템의 구성도를 나타낸다.

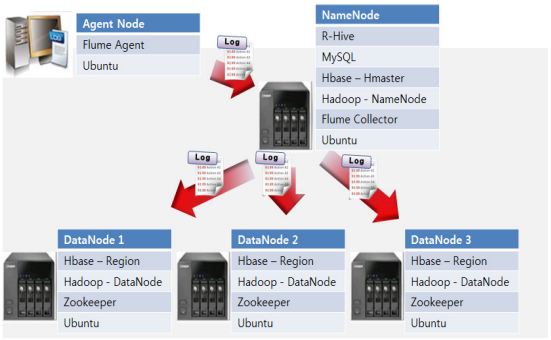

Fig. 2 Configuration of a distributed parallel system

3.1.2. Flume의 로그 수집 절차

Flume은 데이터 수집 도구로서, 분산 환경에서 생성되는 로그 데이터를 agent 노드와 collector 노드를 이용해 전송/수집하는 툴이다. Agent 노드는 분산된 노드들에 설치되며 사용자 설정에 따른 데이터를 전송하는 역할을 하며, 전송된 데이터는 collector 노드에 수집되어진다.

3.1.3. 데이터베이스 시스템으로의 데이터 저장

로그 발생 시 agent는 collector로 로그 데이터를 전송하고, 데이터를 수집한 collector는 정규식을 이용해 HBase로 데이터를 저장한다. 정규식이란 문자열의 검색, 치환, 추출을 위한 패턴으로 언어별 사용법은 조금씩 다르다. 로그 데이터를 HBase에 저장할 때 로그 데이터의 날짜, 시간, 출발지 IP, 목적지 IP, 출발지 포트, 목적지 포트 등을 나누어 저장하기 위해 정규식을 사용한다. 또한, 이러한 작업은 Hive 등에서 데이터 분석이용이하도록 한다.

3.1.4. Hive를 통한 데이터 분석 모듈

Hive는 Hadoop 데이터 및 HBase의 테이블과 연동하며 SQL과 유사한 Hive-QL을 이용하여 조회한다. HBase의 테이블을 Hive와 연동하기 위해 쿼리를 입력한다. 이를 통해 HBase의 테이블을 Hive와 연결하여 Hive에서도 HBase의 데이터를 조회할 수 있게 된다.

Hive의 명령 쿼리인 Hive-QL은 기존 RDBMS의 SQL과 매우 유사하다. Hive-QL 또한 SQL의 테이블 생성, 데이터 조회, 테이블 간 조인을 지원한다[11, 12]. 생성하는 테이블의 종류로는 관리(managed) 테이블과 외부(external) 테이블이 있는데, 관리 테이블은 Hive가 직접 데이터 파일을 관리하며 테이블 삭제 시 메타 정보와 함께 파일까지 삭제한다. 외부 테이블은 기존 파일을 테이블의 데이터로 사용하며 테이블 삭제 시 메타 정보만 삭제한다.

3.2. MySQL과 HBase 연동

HBase와 MySQL을 연동하기 위해 Java 프로그래밍과 Thrift, Flume을 이용할 수 있다. Flume의 replicating 기능과 Thrift 서버 생성을 통해 데이터의 동시 전송이 가능하다. HBase의 데이터 적재는 Flume의 Hbase sink를 통해 전달된다. Flume의 collector 설정에서 Hbase sink를 통해 source에 데이터를 전송받도록 하였다. Agent에서는 Flume의 replicating 기능을 통해 하나의 source를 다수의 channel로 분할/복사가 가능하며, agent 노드의 source에 Hbase sink channel과 Thrift sink channel로 나누어 동일한 데이터를 전송하도록 한다. 여기서 Thrift sink란 포트 번호를 통해 해당 포트에 존재하는 Thrift 서버로 이벤트를 전달하는 역할을 한다. 로그 데이터 생성 후 Hbase sink를 통해 HBase 데이터베이스로 데이터를 적재하며, 동시에 Thrift 서버로 이벤트와 함께 데이터를 전달한다. Thrift 서버는 Apache에서 제공하는 프로젝트로 Java 언어를 통해 쉽게 구현이 가능하다. Java 소스에서 서버를 생성하고 포트를 지정해 준 뒤, Flume의 설정을 replicating으로 변환하고 Thrift sink를 통해서도 데이터를 전달하도록 하면 로그 데이터 생성 시 이벤트가 발생한다. 그림 3은 Flume을 통한 HBase와 MySQL의 연동 방식을 보여준다.

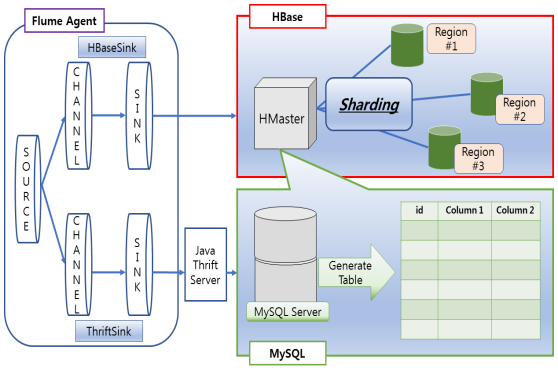

Fig. 3 Interworking SQL and HBase

Ⅳ. 시스템 구현 및 성능 평가

4.1. 오토매핑 미적용 데이터베이스 시스템

4.1.1. HBase와 MySQL 성능 비교

본 논문에서는 단일 서버의 MySQL과 분산시스템의 HBase를 병행하여 사용하기 위해서 두 종류의 데이터베이스를 모두 설치하고 상황에 따라 효율적인 데이터베이스로의 접근을 목표로 한다. 한 시스템에 두 개의 데이터베이스를 설치하게 되면 용량적인 제한이 있을 수 있지만 빅데이터의 빠른 처리시간을 위해 MySQL과 HBase를 병행하여 사용한다.

검색 절차에 있어 쿼리에 조건문을 주게 되면 MySQL은 데이터베이스를 스캔한 뒤 조건절을 분석하며 데이터에 접근하게 된다. 또한, HBase는 MapReduce 작업을 거쳐 분산시스템에 저장되어 있는 데이터를 Map 작업을 한 다음 조건절을 분석하고 나온 결과를 Reduce 작업으로 데이터를 출력한다.

그림 4에서와 같이 적은 용량의 데이터에서 MapReduce 작업은 불필요하게 많은 작업시간이 소요된다. 즉, 적은 양의 데이터인 경우 MapReduce 작업은 불필요하고 MySQL로 접근하는게 효율적이라는 것을 알 수 있다.

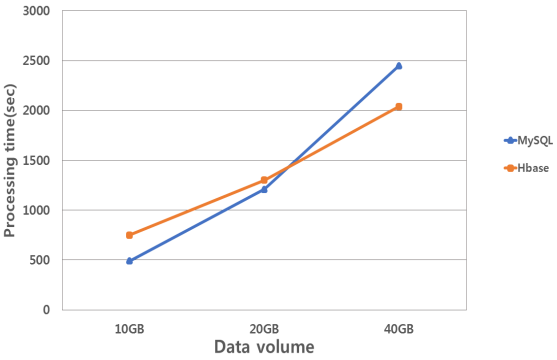

Fig. 4 Comparison of data retrieval time (MySQL vs HBase)

4.2. 오토매핑 모듈을 통한 데이터베이스 연동 시스템

4.2.1. 프로그래밍을 통한 데이터베이스 접근

오토매핑 모듈은 Java 언어로 작성되었고, HBase/Hive와 MySQL API를 이용해 각 데이터베이스에 접근한다.

이외에 데이터베이스 접근을 위해 Statement와 ResultSet 클래스가 사용되었다. Java에서 Thrift 서버를 운용하기 위해 org.apache.thrift 패키지에 있는 server, protocol, transport 패키지를 사용하였다. Java에서는 서버를 생성하며 포트번호를 설정하고, Flume의 Channel에서는 Thrift Sink로 연결하여 설정된 포트번호를 지정하면 생성된 Java의 Thift 서버로 이벤트가 전달된다.

4.2.2. 데이터베이스 접근 알고리즘

새로운 로그 데이터가 생성되면 Flume은 HBase로 데이터를 전송하여 HBase 데이터베이스에 적재하며, 동시에 Thrift 서버로 이벤트를 전달한다. 전달받은 Thrift 서버 즉, Java로 작성된 오토매핑 모듈에서는 그림 5와 같은 절차로 데이터베이스의 조회시간을 측정하고, 효율적인 데이터베이스를 선택한다.

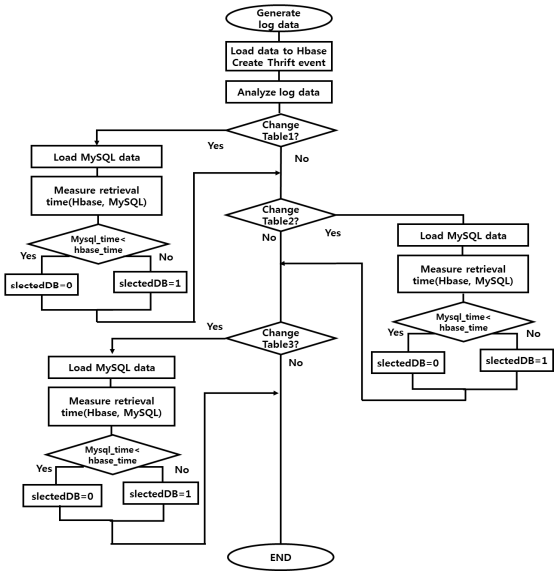

Fig. 5 Auto-mapping algorithm when log is generated

로그 데이터가 생성되면 Flume에서는 HBase sink를 통해 HBase 테이블에 데이터를 적재한다. 또한, 동시에 Thrift sink를 통해 Thrift 이벤트를 발생시키며, Java에서 생성한 Thrift 서버를 통해 오토매핑 모듈이 실행된다. 오토매핑 모듈에서는 Thrift 이벤트의 로그 데이터를 분석하여 변경되어야 할 테이블을 확인하며, MySQL의 해당 테이블에 쿼리를 통해 데이터를 입력한다. 쿼리문 실행이 끝나면 오토매핑 모듈은 MySQL과 HBase의 검색시간을 측정하며, 시간이 더 빠른 데이터베이스를 확인하고 해당 테이블 객체의 변수를 통해 테이블 정보를 갱신한다. 테이블 변경 여부는 모든 테이블에 거쳐 확인하며, 변경이 필요한 테이블에 한해 이러한 작업이 반복된다. 또한, 모든 테이블 변경 및 데이터 조회시간 갱신이 이루어지면 다시 Thrift 서버의 이벤트를 대기한다.

또한, 모토매핑 모듈은 테이블 객체의 데이터베이스 선택을 이용해 데이터베이스 접근에 있어 효과적인 성능을 기대할 수 있다. Thrift 서버를 통해 외부의 다른 플랫폼 환경에서 접근도 가능하며, 분석 및 모니터링 작업을 위한 웹페이지에서의 접근에도 선택된 데이터베이스를 확인하여 데이터를 조회하는 것이 가능하다. 데이터베이스 접근 알고리즘은 그림 6과 같다.

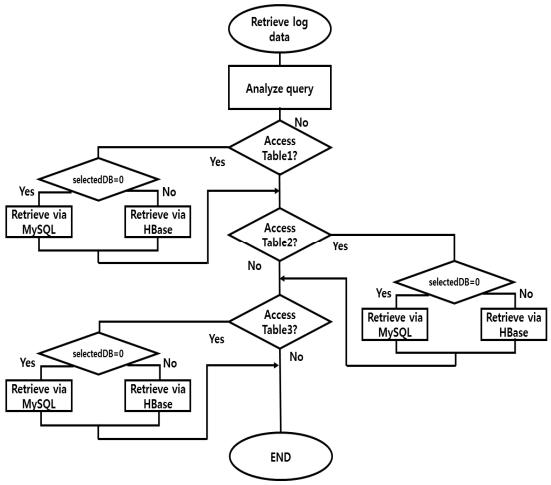

Fig. 6 Auto-mapping algorithm for database access

데이터를 조회하는 과정에서 오토매핑 모듈은 쿼리를 분석하여 어떠한 테이블로 접근하는지를 판단한다. 또한, 판단된 테이블 객체 정보를 이용해 어떠한 데이터베이스가 효율적인 데이터베이스인지를 확인하고 해당 데이터베이스로 테이블을 조회하게 된다. 다수의 테

이블 조회가 이루어질 경우도 오토매핑 모듈은 해당 테이블 객체 정보를 이용해 데이터베이스를 선택하며, 해당 데이터베이스로 조회하도록 한다.

4.3. 오토매핑 모듈의 성능 평가

로그 데이터가 생성되면 Flume은 HBase 적재와 함께 Thrift sink를 통해 Thrift 서버로 이벤트를 전송한다. 오토매핑 모듈의 Thrift 서버에서 이벤트를 전달받게 되면 전송된 데이터를 분석하여 다수의 테이블 중 갱신이 필요한 일부의 테이블을 판단하고 MySQL의 해당 테이블에 새로운 데이터를 적재한다. HBase의 경우 Flume을 통해 적재가 가능하다. 모듈에서 MySQL로 적재가 완료되면 검색시간을 측정하여 데이터베이스를 선택한다. 앞선 데이터베이스 성능 검증을 토대로 20GB 즉, 약 2000만 row까지의 데이터 탐색에 있어서는 MySQL의 데이터 조회 속도가 더 빠르다는 것을 알 수 있다. 성능 평가에 있어 로그의 IP별로 3개의 테이블을 따로 두었으며, 각 테이블에 해당하는 IP로 로그를 생성 한 후, 생성된 로그 테이블에 대한 MapReduce 작업만 거치고 조회시간을 비교해 데이터베이스를 선택하여 모든 테이블을 조회한다. 성능 평가는 두 가지 시나리오를 통해 측정하였다. 3개의 테이블은 IP에 따라 나뉘어져 있으며, 각각 10GB의 데이터로부터 시작한다. 데이터가 증가함에 따라 MySQL과 HBase의 성능을 비교하며 오토매핑 모듈의 성능을 확인한다.

(시나리오 1) 하나의 테이블 데이터가 증가할 경우

초기 데이터 생성에 있어 별도의 시간 측정이 없어도 MySQL의 탐색 시간이 더 빠르다. 데이터 양이 많아짐에 따라 HBase의 탐색 시간이 MySQL보다 빠르기 때문이다. 따라서 초기 테이블 객체의 데이터베이스 선택은 MySQL로 되어 있다. 실험은 IP 별로 나뉜 세 개의 테이블을 설정하고 그 중 하나의 테이블만 용량을 증가시켜 MySQL과 HBase, 오토매핑 모듈을 통한 데이터베이스 선택으로 평가하였다. 테이블의 데이터 용량 증가 정도는 표 2와 같다.

그림 7에서와 같이 각 테이블이 모두 10GB인 경우 HBase보다 MySQL에서 조회하는 탐색 시간이 더 짧기 때문에 오토매핑 모듈은 데이터베이스 선택에 있어 MySQL을 선택한다. 하지만 테이블 1이 20GB가 초과하게 되면 오토매핑 모듈은 MySQL의 조회시간 보다 HBase의 조회시간이 빠르다고 탐지하게 되고, 테이블 1에 대한 데이터베이스 선택을 HBase로 변환한다. 그리고 조회 절차에도 있어 테이블 1에 한해서는 HBase로 접근을 하며, 나머지 테이블 2와 테이블 3은 MySQL이 빠르다고 판단되어 MySQL로 접근을 한다. 그 결과 MySQL만 사용하거나 HBase만 사용한 데이터 조회 시간보다 오토매핑을 통해 데이터베이스를 연동한 조회 시간이 더 빠르게 된다.

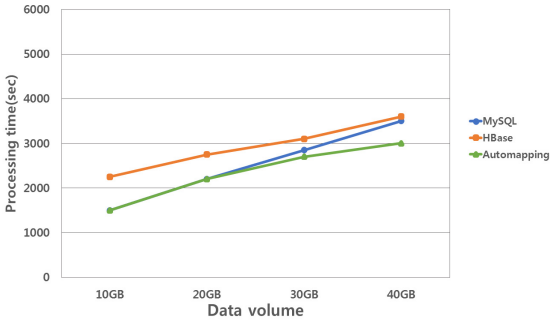

Fig. 7 Data retrieval time (scenario 1)

(시나리오 2) 테이블 하나의 데이터가 40GB이고, 나머지 중 하나의 테이블의 데이터가 증가할 경우

두 번째 시나리오로 테이블 1은 이미 MySQL 보다 HBase의 조회 시간이 빠르다고 판단된 경우, 테이블 2의 데이터 용량을 10GB에서 순차적으로 증가시켜 MySQL만 사용하는 경우, HBase만 사용하는 경우와 오토매핑을 통해 두 가지 데이터베이스를 병행하는 경우의 조회 시간을 평가한다. 각 테이블의 데이터 용량 증가 정도는 표 3과 같다.

Table. 2 Data volume of each table for scenario 1

	T1	T2	T3	T4
Table 1	10GB	20GB	30GB	40GB
Table 2	10GB	10GB	10GB	10GB
Table 3	10GB	10GB	10GB	10GB

Table. 3 Data volume of each table for scenario 2

	T1	T2	T3	T4
Table 1	40GB	40GB	40GB	40GB
Table 2	10GB	20GB	30GB	40GB
Table 3	10GB	10GB	10GB	10GB

시나리오 1에서와 달리 초기 지정된 테이블 1의 데이터베이스 조회 시간은 MySQL보다 HBase가 효율적이라고 판단되었다. 그렇기 때문에 용량이 증가되는 테이블 2가 10GB인 순간부터 데이터베이스 조회 시간은 단일 데이터베이스를 사용할 때 보다 오토매핑을 통해 데이터베이스를 병행하여 접근 할 경우의 데이터 조회 시간이 더 빠르게 측정 되었다(그림 8). 테이블 2의 데이터가 증가함에 따라 오토매핑을 통한 데이터베이스 선택에 있어 세 개의 테이블 중 두 개의 테이블이 HBase를 사용하기 때문에 효율적인 데이터베이스 접근이 가능하다. 시나리오 1과 시나리오 2를 통해 저용량 데이터에 있어서 오토매핑을 통한 데이터베이스의 연동은 MySQL의 성능과 근접하고, 대용량 데이터에 있어서는 HBase와 근접해지기 때문에 유동적이고 효율적인 데이터베이스의 접근이 가능하다.

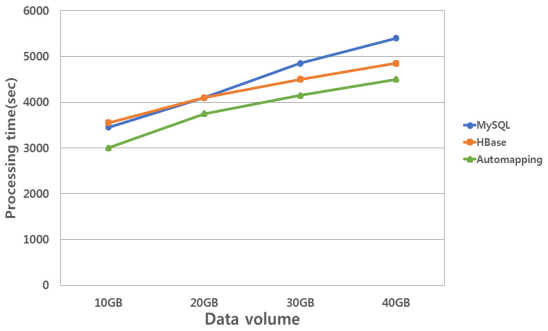

Fig. 8 Data retrieval time (scenario 2)

실험 결과 데이터 양에 따라 효율적인 데이터베이스로 접근하기 때문에 10GB의 저용량 데이터인 경우 NoSQL보다 약 35%, 40GB의 대용량 데이터에서는 RDBMS 보다 약 17% 빠른 검색 시간을 나타내었고, 단일 데이터베이스 사용에 비교하여 검색시간이 항상 더 빠르게 나타났다.

Ⅴ. 결 론

분산병렬처리시스템에서 데이터가 증가할 때마다 효율적인 데이터를 관리하기 위해 데이터베이스 시스템을 변경하는 것은 현실적으로 매우 힘들다. 그렇다고 계속하여 기존의 데이터베이스를 사용하게 되면 데이터가 증가할수록 데이터베이스의 성능은 큰 폭으로 줄게 되고, 막상 대용량 데이터에 적합한 데이터베이스 시스템을 도입하려 하면 현재 데이터의 양이 많지 않아 당장의 성능을 기대하기가 어렵다.

본 논문에서는 이러한 문제를 해결하기 위해 기존의 RDMS와 NoSQL의 두 개의 데이터베이스시스템을 설치하여 처리할 데이터 양에 따라 오토매핑 기법을 이용하여 데이터베이스시템을 선택적으로 사용할 수 있는 방안을 제시하였다. 저용량 데이터의 경우 RDBMS를 사용하여 검색시간을 줄일 수 있고 대용량 데이터의 경우 NoSQL로 접근하여 검색시간을 상당히 줄일 수 있음을 확인하였다.

ACKNOWLEDGMENTS

This work was supported in part by the human resource training program for regional innovation and creativity through the Ministry of Education and National Research Foundation of Korea (2015H1C1A1035859), and the leading human resource training program of regional neo industry through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and future Planning (grant number) (NRF-2016H1D5A1909154).

REFERENCES

[1] Y. G. Kim, S. H. Kim, M. H. Jo, and W. J. Kim, “The Bigdata Processing Environment Building for the Learning System,” *Journal of the Korea Institute of Electronic Communication Sciences*, vol. 9, no. 7, pp.791-797, July 2014.

[2] S. R. Kim, G. S. Jang, and C. W. Cho, “Case Study of Design and Implementation for Hadoop-Based Integrated Facility Monitoring System,” *Journal of the Korea Institute of Industrial Engineers*, vol. 40, no.1, pp.34-42, Jan. 2014.

[3] K. S. Kim, S. J. Ham, J. Y. Ha, and T. S. Kim, “Performance Analysis of HDFS based on Heterogeneous

Storages," *in Proceedings of Korea Computer Congress*, Pukyong University, pp.1475-1477, April 2014.

[4] H. Y. Ahn, K. H. Lee, S. H. Lee, Y. H. Lee, S. M. Lee, and Y. K. Kim, "An Efficient Method for Enhancing the Storage Efficiency in Hadoop DFS," *KIISE Transactions on Computing Practices*, vol.19, no.3, pp.144-148, Mar. 2013.

[5] H. W. Kim, S. E. Park, and S. Y. Euh, "The Distributed Encryption Processing System for Large Capacity Personal Information based on MapReduce," *Journal of the Korea Instituted of Information and Communication Engineering*, vol.18, no.3, pp.576-585, Mar. 2014.

[6] K. S. Noh and D. S. Lee, "Bigdata Platform Implementation Model," *Indian Journal of Science and Technology*, vol.8, no.18, Aug. 2015.

[7] A. Abouzeid, K. B. Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, "HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads," *In Proceedings of the VLDB*, Aug. 2009.

[8] J. K. Bae, "A Study on Technical Issues and Institutional Issues of BigData Analysis Market: Focusing on the In-depth Interview Method," *Asia-pacific Journal of Multimedia Services Convergent with Art, Humanities, and Sociology*, vol.7, no.5, pp. 885-894, May 2017.

[9] K. A. Yang, D. W. Lee, K. H. Kim, and H. J. Yoon, "Analysis of Security Threat and Security Requirements of the Bigdata System," *Journal of Security Engineering*, vol.13, no.6, pp. 501-514, June 2016.

[10] S. T. Hong, M. Yun, D. H. Choe, H. S. Jo, and J. U Jang, "HadoopX : Hadoop MapReduce-based Iterative Data Processing System," *Korea Information Processing Society Review*, vol.21, no.3, pp.8-16, Mar. 2014.

[11] S. H. Lee and D. W. Lee, "Big Data Processing and Utilization," *Journal of Digital Convergence*, vol.11, no.4, pp.267-271, April 2013.

[12] K. H. Han et al, "A Study on implementation model for security log analysis system using Big Data platform," *Journal of Digital Convergence*, vol.12, no.8, pp.351-359, Aug. 2014.

김희성(Hee-Sung Kim)

2013년 대전대학교 정보통신공학과 졸업(학사)
2015년 대전대학교 정보통신공학과 졸업(석사)
현재 지식시스템(주) 연구원
※ 관심분야 : 클라우드 컴퓨팅, 네트워크보안 등

이봉환(Bong-Hwan Lee)

1985년 서강대학교 전자공학과졸업(학사)
1987년 연세대학교 대학원 전자공학과 졸업(석사)
1993년 Texas A&M 대학교 대학원 전기 및 컴퓨터공학과 졸업(박사)
현재 대전대학교 전자정보통신공학과 교수
※ 관심분야 : 클라우드 컴퓨팅, 사물인터넷, 네트워크보안 등