공 학 석 사 학 위 논 문

스쿱과 타조의 효율적인 데이터 처리 성능에 관한 연구

지도교수 여 정 모

이 논문을 공학석사 학위논문으로 제출함

2014년 12월

부 경 대 학 교 대 학 원

컴 퓨 터 공 학 과

고 정 현

목차

표 목차

그림목차

스쿱과 타조의 효율적인 데이터 처리 성능에 관한 연구

고정현

부경대학교 일반대학원 컴퓨터공학과

요 약

빅데이터 처리 플랫폼인 하둡의 등장 이후 하둡 기반 기술들을 이용하여 데이터 분석을 할 수 있는 SQL-on-Hadoop 기술이 주목받고 있다. 하둡 관련 기술의 등장으로 DW 시장의 변화가 포착되고 있지만 그 성능에 관한 연구는 미미한 실정이다. 그래서 본 연구에서는 하둡 기반 기술을 이용하여 관계형 데이터베이스와의 데이터 분석성능 비교에 관한 실험을 진행하여 하둡 기반 DW 솔루션들에 대한 선택에 도움이 될 연구를 수행하였다. 분석의 과정의 선행 절차로 데이터를 확보하기 위한 데이터 적재 과정을 포함하여 하둡 기반 기술 중 가장 많이 사용하는 적재 도구인 스쿱(SQOOP)을 이용하여 적재 성능을 비교하였고, 국내 개발자가 주축이 되어 개발하고 2014 년 4 월 아파치 최상위 프로젝트로 선정되어 국내외에서 많은 관심을 받고 있는 SQL-on-Hadoop 기술인 타조(TAJO)를 이용하여 관계형 데이터베이스와의 데이터 분석성능 비교에 관한 실험을 진행하였다. 적절하지 못한 환경 구성을 통해 하둡기반 기술을 사용한다면 적재와 분석 모두 관계형 데이터베이스에 비해 좋은 성능을 얻을 수 없었으나 스쿱과 타조를 성능에 영향을 미치는 여러 요소들에 대해 올바른 사용전략을 세워 활용한다면 관계형 데이터베이스보다 우수한 성능을 보인다는 결과를

얻었다. 또한, 오픈 소스인 하둡 기술들은 개발자들의 많은 참여로 인해 점차 기술의 완성도가 높아져 DW 및 데이터 분석분야에서 중요한 축을 담당할 수 있을 것으로 예상한다.

(A) Study on the Efficient Performances of Data Processing of Sqoop and Tajo

Ko Jung-Hyun

Dept. of Computer Engineering of Pukyong National University

Abstract

Since introduction of Hadoop, big data processing platform, the SQL-on-Hadoop technology available for data analysis using Hadoop-based technologies has attracted attention. With the advent of Hadoop-related technologies, changes of DW market are being captured but there are few studies on their functions and performances. In this study, using Hadoop-based technologies, an experience about comparison of data analysis performance with relational database was conducted for helping to select Hadoop-based DW solutions. For pre-analysis process, using SQOOP, the most used loading instrument of Hadoop-based technologies including data loading process for securing it, loading performances were compared. And using TAJO, SQL-on-Hadoop technology, which was developed mainly by Korean developers and chosen as the highest project of Apache in April 2014, getting attention from overseas and domestic parties, an experiment about

comparison of data analysis performances with relational database was also done. If using Hadoop-based technology through configuring improper circumstance, both loading and analyzing performances were not better than relational database. However, after figuring out experimentally several factors affecting performances of SQOOP and TAJO, the improved experiment showed the results that it has better performance than that of relational database. Also, level of open-source Hadoop technologies has been improved by participation of various developers and it is expected that they will contribute significantly to DW and data analysis.

Ⅰ. 서론

최근 스마트폰, RFID(Radio Frequency Identification), 센서 등의 다양한 IT(Information Technology) 기술의 발전과 더불어 개인 블로그, 페이스북, 트위터 등의 SNS(Social Network Services)의 보급과 활성화로 인해 수많은 데이터가 빠르게 생산되고 있으며 그 데이터를 활용하고자 하는 기업의 상황에 큰 변화가 일어나고 있다. 방대한 양의 데이터를 활용하여 높은 이익을 얻기 위해 데이터를 효과적으로 저장, 분석 처리하는 기술에 한계가 있다는 사실을 느끼기 시작하였고 그에 따라 빅 데이터(Big Data)라 불리는 데이터를 활용하기 위한 새로운 IT기술에 대한 요구가 점점 커지고 있다. 이와 같은 요구에 힘입어 빅 데이터를 본격적으로 활용하기 위해서 다양한 기술들이 개발되어 활용되었고, 그러한 기술 중 하둡(Hadoop)[1]이 대용량 데이터의 분석 및 처리의 한계점을 뛰어넘을 가능성을 보이면서 이제는 빅 데이터 처리의 표준 기술처럼 사용되고 있다. 하둡의 등장으로 인해 이전에는 저장 및 처리할 수 없던 수많은 데이터에 대한 처리 및 분석의 문제를 해결하고 새로운 가치를 창출할 수 있게 된 것이다.

하둡의 등장과 함께 다양한 분산처리 기술들이 등장함으로써 기존 DW(Data Warehouse) 시장의 중심에 있는 업체들의 시장 판세에 커다란 변화가 포착되고 있다. 가트너(Gartner)에서는 2014년 3월 DW 관련 보고서[4]에서 처음으로 빅데이터 및 하둡 전문업체 클라우데라(Cloudera)를 포함했다. 또한, 빅데이터를 지렛대로 활용하는 새로운 경쟁자가 떠오르고 있고 전통적인 DW 벤더도 빅데이터 기술에 투자하고 있다고 발표했다. 그러나 DW 시스템에서 주로 활용되는 관계형 데이터베이스(Relational Database Management System : RDBMS)와 하둡 관련 기술들과의 성능 비

교에 관한 연구는 여전히 부족한 실정이다. 수많은 RDBMS와 하둡 기반 DW 솔루션들은 저마다 벤치마크(Benchmark) 성능테스트 결과가 우수하다며 발표하고 있지만, 그 결과는 각각의 솔루션들에 대해 하드웨어와 시스템 환경을 최적화시켜 발표한 결과이며 특정 유사기술들에 대한 비교결과만을 발표하고 있다. 즉 동일한 하드웨어 환경에서 하둡 기반기술과 RDBMS의 성능비교연구는 미미하다는 것이다. 또한, 오픈 소스인 하둡 기반기술은 기존 DW 시스템 대비 가격경쟁력이 뛰어나고 값싼 하드웨어를 추가하여 유연하게 시스템을 확장해 나갈 수 있는 장점을 가지고 있음에도 불구하고 기존의 DW가 제공하는 모든 이점을 대신해 주지 않기 때문에 DW 사용자들이 어떤 것을 선택해야 할지 고민하고 있다. 그래서 본 논문에서는 하둡 기반기술을 이용한 데이터 처리 성능과 RDBMS의 성능을 비교하여 어느 정도 성능차이를 보이는지에 관한 연구를 수행하여 DW 선택 판단의 도움이 될 근거를 제공하고자 한다.

본 연구에서는 빅데이터라는 단어에서 연상되는 비정형 데이터가 아닌 정형데이터를 대상으로 데이터를 처리하는 것에 초점을 맞춘다. 그 이유는 데이터를 효과적으로 활용하여 최대의 비즈니스 가치를 끌어내고자 할 때 먼저 기존에 쌓아둔 데이터를 이용하여 단기적인 성과를 추구해야 하고 실제로 빅데이터 기술을 활용중인 대부분의 기업에서도 기업의 내부 데이터를 빅데이터 분석의 소스로 활용하고 있기 때문이다.[5] 또한, 데이터 분석을 위한 선행단계로 데이터 적재 단계를 포함한다. 데이터를 분석하기 위해 가장 먼저 선행되어야 할 작업이 기존에 존재하는 원천 데이터를 수집하는 것이기 때문이다. 데이터 적재 단계에서는 하둡 기반기술 중 정형 데이터를 수집하기 위해 가장 많이 사용되는 아파치 스쿱(Apache Sqoop)[6]을 사용하여 실험을 진행하며, 데이터 분석 성능 비교를 위해서는 최근 국내 개발자가 주축이 되어 개발하고 2014년 4월 아파치 최상위 프로젝트(Top Level Project)로 승격되어 많은 주목을 받고

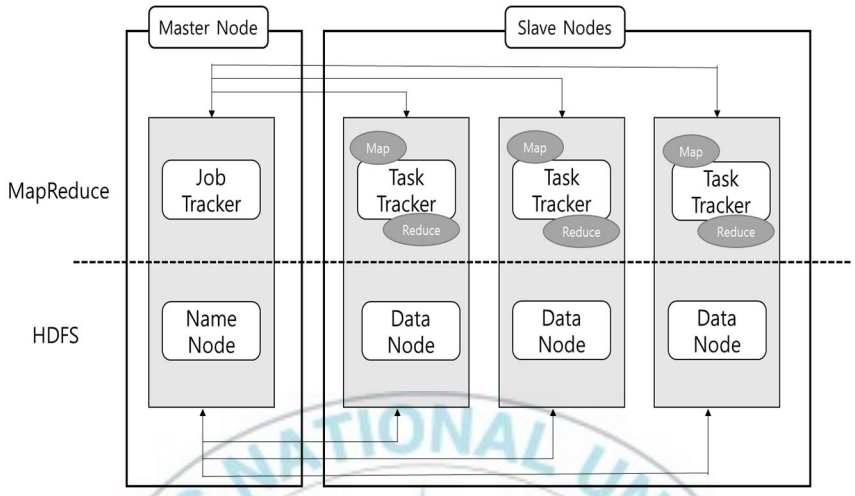
있는 하둡 기반 DW 시스템인 타조(Tajo)[7,8]을 이용하여 RDMBS와의 정형 데이터 처리 성능 비교에 관한 연구를 수행할 것이다. 본 연구는 하둡 기반기술인 스쿱과 타조의 데이터 처리성능이 RDBMS와 비교하여 어느 정도의 성능을 내는지에 대한 실험뿐만 아니라 스쿱과 타조를 효율적으로 사용하여 더 나은 성능을 보일 수 있는지에 대한 요인을 찾아 하둡 기반 기술의 효율적인 데이터 처리 성능에 관한 연구를 수행한다.

II. 관련연구

1. 하둡

하둡은 수많은 컴퓨터를 네트워크로 연결한 후 데이터를 분산시켜 처리하는 오픈 소스 프레임워크이다. 하둡은 오픈 소스 개발자인 더그 커팅(Doug Cutting)과 마이크 카파렐라(Mike Cafarella)에 의해 2006년에 발표되었다. 하둡 개발 이전 커팅과 카파렐라는 너치(Apache Nutch)라는 오픈 소스 검색 엔진 프로젝트를 주도하고 있었는데 너치 프로젝트의 너치 분산 파일 시스템(Nutch Distributed File System; NDFS)과 맵리듀스(MapReduce)가 검색 엔진을 뛰어넘어 많은 분야에 다양하게 사용될 수 있음을 깨닫고 2006년 2월 너치 프로젝트에서 독립시켜 '하둡' 이라 명명하였다. 하둡 프로젝트는 야후(Yahoo)의 지원을 받아 급속히 성장하였고 2008년 2월 하둡이 아파치 재단 내 최고 수준 오픈 소스 프로젝트로 격상되었다. 야후뿐만 아니라 페이스북에서도 100PB의 스토리지로 구성된 하둡 클러스터를 갖추고 있는 것으로 알려졌다. 이처럼 하둡 프로젝트가 성공적으로 안착하자 대용량 데이터 처리와 분석에 어려움을 겪고 있던 많은 기업에서 하둡을 그 해법으로 이용하고자 하는 수요가 폭발적으로 늘어났다. 이에 설치가 쉬우면서도 상업적 소프트웨어 수준의 안정성을 갖춘 하둡 배포 버전을 제작, 판매하는 회사들이 하나둘씩 생겨나기 시작했고, 이들이 하둡 프로젝트에 기여하면서 본격적인 하둡 생태계가 생성되기 시작하였고 점차 빅데이터 처리의 업계 표준으로 자리 잡았다.[9]

하둡은 대용량 비정형 데이터를 효율적으로 저장할 수 있는 HDFS(Hadoop Distributed File System)과, 대용량 데이터 처리를 효과적으로 하기 위한 분산 병렬처리 프레임워크인 맵리듀스로 구성된다.



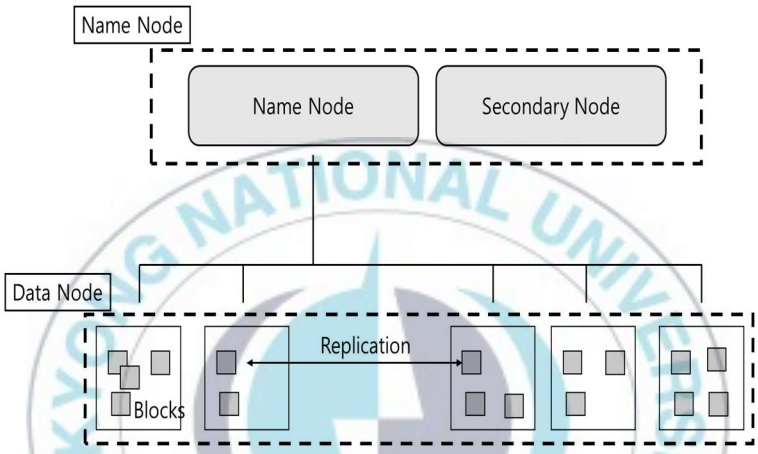
[그림 1] 하둡(Hadoop) 아키텍처

그림 1과 같이 하둡은 메타정보를 저장하고 있는 마스터 노드(Master Node) 한 대 및 데이터를 저장하고 처리하는 슬레이브 노드(Slave Node)로 구성되어 있다. 여러 가지 사유로 슬레이브 노드에 장애가 발생하더라도 나머지 노드를 이용해 기존의 시스템을 유지해주는 안정적 설계로 되어 있다.

1.1 하둡 분산 파일 시스템

HDFS는 신뢰성과 확장성을 갖춘 병렬 컴퓨팅을 위해 개발된 분산 파일 시스템이며, 하둡 프레임워크 상에서 맵리듀스 기반의 병렬 처리를 수행할 때 기반이 되는 파일 시스템이다. 현재 아마존(Amazon), IBM, 야후(Yahoo) 등과 같은 글로벌 IT 기업들의 클라우드 컴퓨팅 플랫폼의 기반이 되는 분산 파일 시스템으로 가장 널리 활용되고 있다. HDFS는 구글

파일 시스템(Google File System:GFS)을 본보기로 삼아 개발된 분산 파일 시스템으로 플랫폼 간의 이식성을 보장하기 위해 자바를 사용하여 구현되었다. 이름만 다를 뿐 구글 파일 시스템과 동일한 구조와 기능을 제공한다.



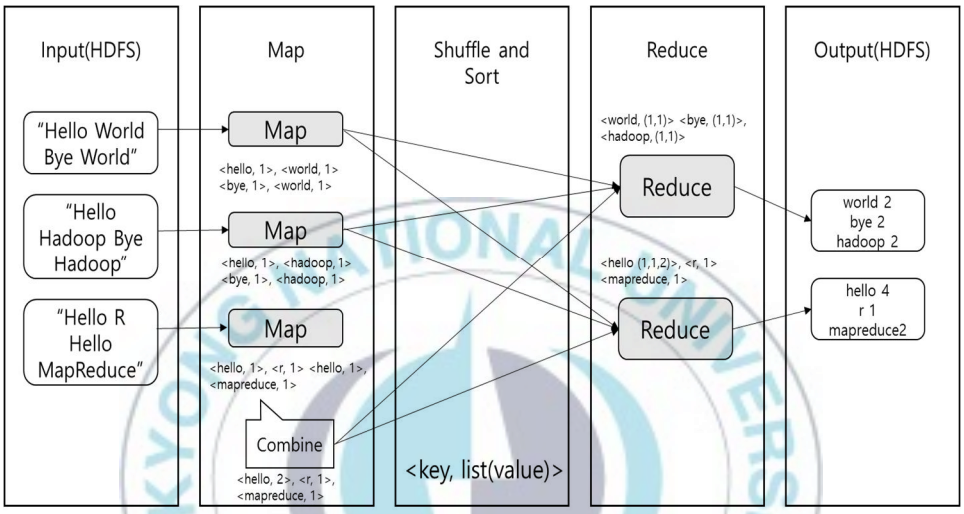
[그림 2] HDFS 아키텍처

HDFS는 그림 2와 같은 마스터-슬레이브(Master-Slave) 구조를 가진다. HDFS 클러스터는 실제 저장하기 위한 데이터들이 아닌 메타(Meta)정보들을 저장하는 하나의 네임 노드(Name Node)와 데이터가 저장되는 데이터 노드(Data Node)가 존재한다. 내부적으로 하나의 파일은 하나 이상의 블록으로 나뉘어 있고, 이 블록들은 데이터 노드들에 저장되어 있다.

1.2 맵 리듀스

맵리듀스는 구글에서 대용량 데이터 처리를 분산 병렬 컴퓨팅에서 처

리하기 위한 목적으로 제작하여 2004년에 발표하였고 분산 환경에 저장된 데이터를 데이터가 저장 된 곳에서 처리할 수 있도록 만든 소프트웨어 프레임워크다.

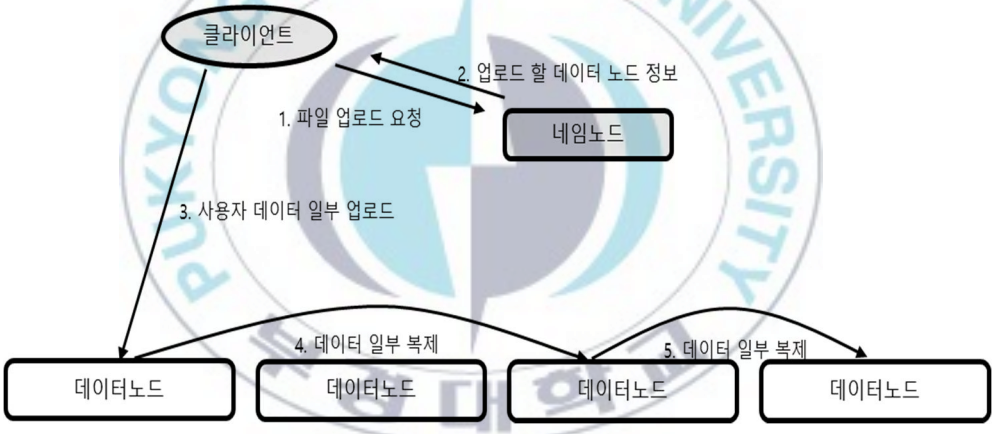


[그림 3] 맵 리듀스 처리과정

맵리듀스는 일반 프로그래밍 방법과는 다른 모형을 제공한다. 그림 3은 맵리듀스의 처리과정을 나타낸다. 일반적인 분산 환경에서의 프로그래밍은 대개의 프로그래머가 익숙한 단일 서버에서의 프로그래밍과 달리 분산된 작업의 스케줄링이나 일부 서버의 고장, 서버 간 네트워크 구성 등 많은 문제를 고려해야 한다. 맵리듀스에서는 이런 복잡한 문제들이 플랫폼 차원에서 단순화 되어 프로그래머는 데이터의 배치 처리를 위한 맵(Mapper)과 리듀스(Reducer)함수만을 작성하면 되도록 구현되어 있다. 병렬 처리에 있어 요구되는 스케줄링 등의 복잡한 사항들을 사용자가 고민하지 않게 함으로써 쉬운 병렬 처리방법을 제공하는 것이 장점이다.[9]

1.3 하둡 복제 정책

분산 시스템은 서로 분산된 여러 서버들을 가지고 파일 시스템을 구성함으로써 높은 확장성과 고 가용성을 지원한다. 원본 데이터를 여러 개 복제본으로 서로 다른 스토리지(Storage)에 저장함으로써 시스템 장애가 발생하더라도 지속적으로 서비스를 제공할 수 있다. 대부분의 분산 파일 시스템에서도 복제를 이용하여 동일한 데이터를 여러 곳에 복제해 놓는데 가장 큰 이유는 고장으로 인한 데이터 손실을 막기 위함이다. HDFS역시 데이터 손실을 방지하기 위해 데이터 복제 정책을 사용한다.



[그림 4] HDFS 데이터 복제 진행과정

HDFS는 사용자 데이터를 업로드와 동시에 여러 데이터 노드에 존재하도록 복제를 수행한다. 사용자가 데이터를 업로드하면 데이터 노드에 데이터가 저장되는데, 이 때 하나의 데이터 노드에만 업로드 한 후 데이터를 나중에 복제한다면 데이터를 잃을 수 있는 확률이 있다. 그래서 HDFS는 사용자가 데이터를 업로드 하는 동시에 여러 개의 데이터 노드에 복

제해 놓는다. 이를 위해 복제 파이프라이닝(Pipelining) 방법을 이용한다.[10] 복제 파이프라이닝은 그림 4와 같이 수행된다. 이와 같이 복제 파이프라이닝을 하면 데이터 손실을 방지할 수 있다. 클라이언트가 업로드하는 동시에 데이터를 다른 데이터 노드에 전달하여 디스크에 보관하기 때문에 업로드가 모두 끝난 후 데이터 노드에 장애가 발생하더라도 데이터를 안전하게 보관할 수 있는 것이다.

하둡 분산 파일 시스템에서는 복제를 위한 환경설정이 존재한다. 하둡 분산 파일 시스템의 기본 설정은 동일한 블록을 3개 복제하는 것이다. 복제본의 개수는 사용자의 설정에 의해 임의로 변경될 수 있다. 분산 파일 시스템 안에서 데이터들을 어떻게 저장하였는지, 복제정책을 어떻게 설정하였는지에 따라서 분산처리성능의 차이가 발생한다. 예를 들어, 처리하고자 하는 데이터들이 모두 하나의 서버에 집중되어 저장되어 있으면 분산 처리의 효과를 전혀 얻을 수 없다. 이러한 측면을 방지하기 위하여 일반적으로 분산 처리 플랫폼 내부적으로 분산 알고리즘에 따라서 데이터를 분산시킨다. 사용자들이 복제정책 설정 옵션을 통해서 데이터의 복제본 수량을 제어할 수도 있다. 복제본 수량이 많아지고 데이터를 여러 서버에 복제하여 분산저장하고 있다면 분산 처리할 때도 좋은 성능을 얻을 수 있다.

분산 시스템에서 복제 정책을 적용하여 많은 이점을 가져올 수 있는 반면에 모든 데이터에 대하여 블록 단위로 여러 개 복제본을 유지하기 때문에 저장 공간이 추가적으로 필요하다. 예를 들어, 복제 인수를 3으로 설정한다면 원본 데이터 추가적으로 복제본 2부를 저장해야 한다. 앞서 언급한 바와 같이 여러 복제본을 만들어 분산하여 저장한다면 분산처리의 성능상 이점을 얻을 수 있고 가용성을 높일 수 있지만 반면 저장공간이 늘어난다. 그렇기 때문에 사용자가 하둡 클러스터 환경을 고려하여 적절한 복제정책을 세워 운영할 필요가 있다.

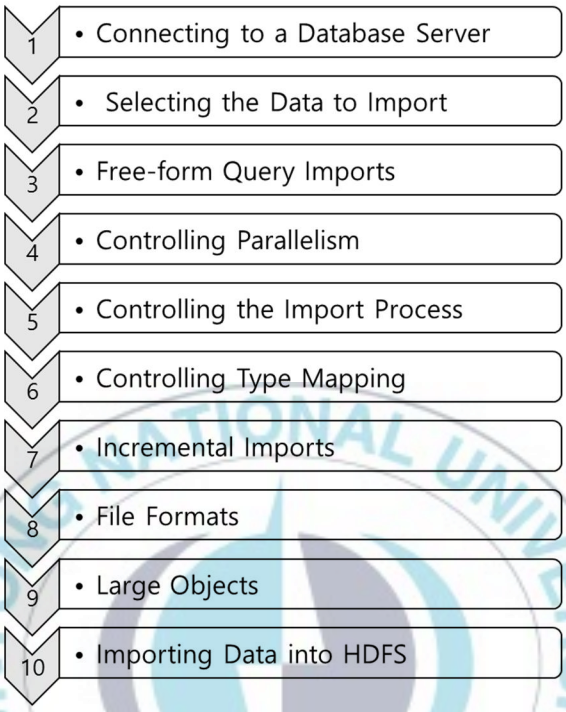
2. 하둡 관련 기술

2.1 스쿱

DW 시스템에서 데이터 분석을 위해서 반드시 선행되어야 하는 과정은 분석하고자 하는 데이터의 수집이다. 수집은 간단해 보이지만 데이터 처리 플랫폼 전체의 아키텍처에 있어서 가장 중요한 요소이며 이후 구축할 여러 가지 판단요소의 기준점이 될 수 있다. 수집은 현재 시스템의 다양한 곳에 존재하는 로그, 데이터 등이 대상이다.

기업 입장에서 하둡 클러스터를 구축하고 실제로 어떤 데이터를 분석할 것인가를 검토해 보면, 먼저 기존에 다양한 RDBMS에 저장되어 있는 데이터를 빼 놓고는 의미 있는 결과를 얻지 못 할 수도 있다. 기존에 RDBMS에 저장된 데이터들은 어느 정도 정제되고 관리되는 데이터 소스이기 때문에 하둡 클러스터에 저장된 비정형 데이터들과 함께 검토함으로써 기업에 진정으로 의미 있는 가치를 얻어 낼 수 있다.

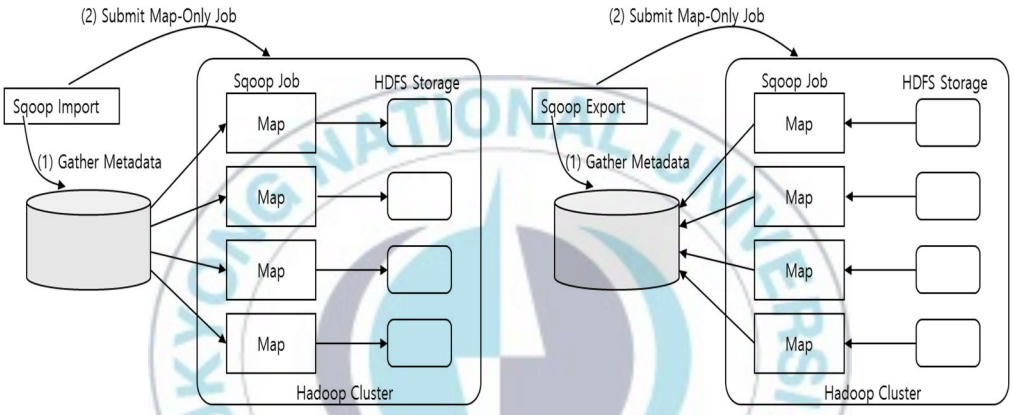
일반적으로 비정형 데이터를 처리하기 위해 하둡을 선택하지만 RDBMS에 저장된 정형 데이터를 하둡에서 처리해야 할 경우도 있다. 예를 들어 별도의 로그 수집 시스템 및 데이터 저장소가 마련되지 않아 RDBMS에 로그를 저장하는 경우에 RDMBS로 계속 누적하면서 데이터를 분석하기에 비용과 시간이 많이 소요되어 하둡과 같은 분산 환경의 저장소로 옮겨 분석할 필요가 있는 경우라던지 로그뿐 아니라 메타성 데이터는 대부분 RDBMS에 저장되어 있는데 이 RDBMS의 메타 데이터를 하둡, 하이브(Hive)로 옮겨야 하는 경우 등이 있다. 이러한 경우 데이터를 손쉽게 옮길 수 있는 방안을 찾게 되는데 스쿱(Sqoop)은 하둡 클러스터와 RDBMS간 데이터 전송을 효과적이고 쉽게 하기 위해 만들어진 도구이다. 스쿱을 사용하면 MySQL, Oracle등의 RDBMS에서 HDFS로 데이터를 손쉽게 가져올 수 있으며 반대로 HDFS에 저장된 데이터들을 RDBMS로 내보낼 수도 있다.



[그림 5] 스쿱(SQOOP) 처리 과정

스쿱은 SQL to hadoop의 의미로서 2009년 첫 버전이 나온 후 2012년에 아파치 톱 레벨 프로젝트(Apache Top Level Project)가 되어 지속 발전하고 있으며 맵리듀스를 기반으로 구현된 데이터 적재프로그램이다. 특히 RDBMS 및 HDFS 사이에 데이터 적재 가능하기 때문에 많은 프로젝트에서 널리 사용하고 있다. 스쿱은 모든 적재 과정을 자동화하며 병렬처리 방식으로 작업한다. 또한 좋은 내고장성(Fault tolerance)을 지원한다. 스쿱은 Row-by-row 방식으로 RDBMS에 저장되어 있는 테이블을 읽고 HDFS에 저장한다. RDBMS에서 읽어온 테이블 하나를 HDFS에서 파일셋으로 저장한다. 병렬 처리 방식으로 적재하기 때문에 적재한 후에 HDFS에서 여

러 개 파일으로 저장하게 된다. 스쿱을 사용하여 HDFS에 저장되어 있는 파일셋을 읽고 RDBMS로 적재하는 Export과정도 가능하다. 파일셋을 병렬 처리 방식으로 읽고 레코드 형태로 변환하여RDBMS의 해당 테이블에 삽입한다. 스쿱이 RDBMS의 데이터를 HDFS로 적재하는 전체적인 절차는 그림 5와 같으며 그림 6은 스쿱을 통해 Import, Export 하는 과정을 보여준다.



[그림 6] 스쿱(SQOOP) Import & Export

2.2 SQL-on-Hadoop

하둡의 분산 병렬 처리 프레임워크인 맵리듀스가 대용량의 데이터를 빠르고 효과적으로 처리할 수 있는 방법을 제공하였지만, 데이터 처리 요청마다 맵리듀스 코드를 작성해야 한다는 단점 때문에 실제 데이터 분석가들이 하둡을 활용하는데 문제점이 존재하였다. 이러한 문제점을 개선하기 위해 아파치 하이브[11] 프로젝트가 등장한다. 하이브는 페이스북에 의해 개발되었고 사용자가 SQL을 통해 질의하면 하이브 엔진에 의해 자동으로 맵리듀스 코드로 변환되어 하둡상에서 실행할 수 있도록 한다. 그래서 사용자가 맵리듀스 코드를 작성하지 않더라도 하이브를 이용

하여 사용자들에게 친숙한 SQL로 질의하여 결과를 볼 수 있는 환경을 제공하였다.

하둡과 하이브가 등장하면서 과거 투자 대비 저렴한 가격으로 대용량 데이터 처리를 할 수 있게 되었다는 사실에 만족하였으나 사용자들은 점차 보다 높은 처리 성능 및 빠른 반응을 요구하였고, 빠른 반응속도로부터 데이터 분석의 생산성을 높여 빠른 의사결정을 가능하게 할 수 있는 시스템이 필요하다는 요구가 증대되었다. 또한, 개발자의 역량에 따라 성능이 좌우되며, 성능 튜닝에 많은 노력이 필요하면서 버그 가능성이 높았다. 맵리듀스와 하이브는 작업 스케줄링, 그리고 맵에서 리듀스로 넘어가는 과정에서 발생하는 데이터 교환 오버헤드가 존재했고 SQL과 유사하지만 많은 부분이 다르고 SQL 표준을 지원하지 않는 불편함이 존재했다. 이 때문에 성능을 보장하며 사람에 의한 오류를 방지하고자 하는 요구가 발생했다. 이러한 사용자들의 요구 사항 변화로 인해 하둡 기반의 차세대 분석 엔진으로 일컬어지는 SQL-on-Hadoop 기술이 등장하게 된다. SQL-on-Hadoop 기술은 맵리듀스 프레임워크가 아닌 새로운 분산 처리 프레임워크를 기반으로 HDFS에 저장된 데이터에 대한 SQL 처리를 제공하는 시스템을 말한다. 또한, SQL 표준을 지원하여 기존 시스템과 통합 또는 대체가 용이하고 맵리듀스의 한계를 극복하는 분산 처리 프레임워크로 보다 높은 처리 성능을 보장한다.[12,13]

먼저 구글에서 2011년 클라우드 기반의 빅데이터 서비스인 빅쿼리(Bigquery)[14] 서비스를 대중에게 공개하였는데 이 서비스는 사용자가 비정형 데이터들을 업로드 한 후, 테이블을 생성하고 SQL 질의를 하는 것과 같은 방식으로 조회 및 분석이 가능한 서비스를 제공한다. 빅쿼리의 내부 플랫폼으로 사용된 핵심 SQL-on-Hadoop 기술이 드레멜[15]이다. 구글은 드레멜 기술에 관련된 논문을 발표한 후, 드레멜과 유사한 시스템을 오픈 소스로 개발하려는 시도들이 많았고 그 사례는 표 1과 같다.

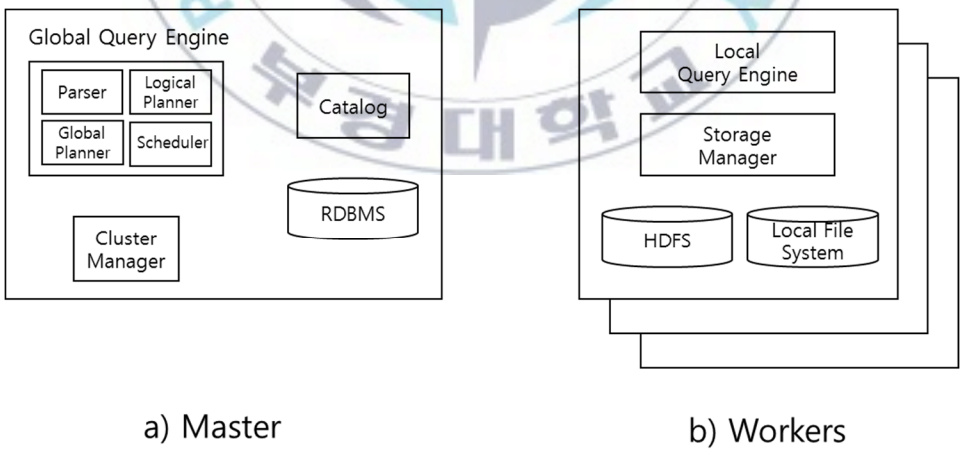
[표 1] SQL-on-Hadoop 기술들의 특징

이름	특징
아파치 드릴 (Drill)	- 하둡 전문 회사 맵알(MapR) 주도로 개발중인 프로젝트. 드레멜의 기능과 유사
아파치 스팅거 (Stinger)	- 하둡 전문 회사 호튼웍스(Hortonworks)에서 개발을 주도. 기존의 하이브 코드를 최대한 이용하여 성능을 개선하는 방식 - Vectorized 엔진 도입으로 기존 튜플 단위 처리 엔진 대체 작업 중
샤크(Shark)	- UC 버클리 대학교에서 개발되어 아파치 top level 프로젝트로 승격 - 인 메모리 기반 시스템 - 하이브와 호환가능
임팔라(Impala)	- 클라우데라(Cloudera)에서 개발 주도 - Low-latency 질의 처리에 특화 - 고성능을 위해 C++ 사용 - 인 메모리 처리에 특화되어 대용량 데이터, 결과값이 큰 데이터 처리에 한계 - 소스는 Open, 참여는 Closed
프레스토(Presto)	- 페이스북에서 자체적으로 개발 - 수 초에서 수분이 걸리는 질의 유형을 타겟으로 설계 - No Fault tolerance - 일부 질의 유형에 대해 approximate query 지원

2.3 타조

다양한 오픈 소스 기반의 SQL 질의 기술들이 대중에게 공개되었는데, 가장 최근에 공개된 기술이 타조(TAJO)이다. 타조는 국내 개발자들이 주도하는 프로젝트로서 2013년 3월 아파치 인큐베이션(Incubation) 프로젝트로 선정된 이후 일 년만인 2014년 4월 최상위 프로젝트로 선정돼 관련 업계의 큰 관심을 받고 있다. 타조는 하둡 기반의 데이터웨어하우징(Data Warehousing) 시스템이며 HDFS 및 다양한 소스의 대용량 데이터에 대한 집계, 연산, 조인, 정렬 등의 분석을 제공한다. 타조 역시 다른 SQL-on-Hadoop 기술들과 유사하게 맵리듀스 프레임워크 대신 자신의 쿼리 실행 엔진을 가진다.

다른 SQL-on-Hadoop기술보다 더욱더 많은 SQL 표준 구문이 호환가능하며 JDBC 및 ODBC, UDF와 호환 가능하다. 또한 유연하고 효율적인 분산 처리 엔진으로 고성능 및 낮은 반응시간에 처리가 가능한 장점이 있다.



[그림 7] 타조 아키텍처

타조는 그림 7[16]과 같이 Master-Worker 모델의 아키텍처로 구성된다.

Tajo Master는 클라이언트 및 애플리케이션의 요청을 처리하며 쿼리 실행 계획과 Worker들의 작업을 조정하는 역할을 한다. Master에서 테이블의 스키마, 물리적인 정보, 각종 통계 등 카탈로그 정보를 가지고 있으며, Query 파서, 플래너, 최적화기, 클러스터 자원 관리, Query Master 등을 관리한다. Query Master는 사용자 질의별 동작을 결정하며 질의 실행 단계(Execution Block)를 제어하고 각 태스크들을 스케줄링하는 역할을 한다. Tajo Worker는 로컬에서 Master에 의해 할당된 작업을 수행하며 Master에게 진행 상태를 주기적으로 보고한다.

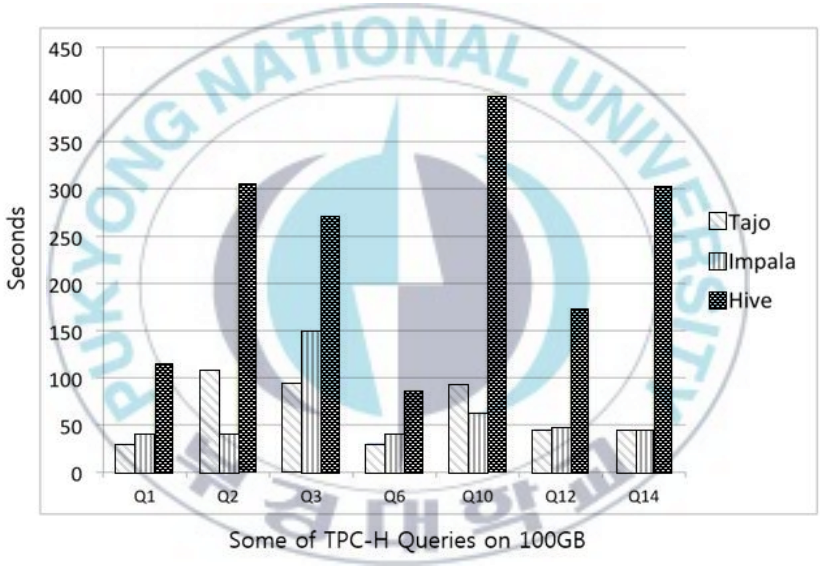
타조의 질의 계획 및 최적화 엔진은 다음과 같은 특징을 가진다. 첫째, 비용기반 최적화(Cost-Based Optimization)방식이다. 다수의 상용 RDBMS와 같이 다양한 정보를 활용하여 시스템이 최적의 조인 순서를 탐색 및 선택하고, 사용자에게 의존하지 않아 효율적인 실행계획을 작성한다. 둘째, 확장 가능한 Rewrite Rule 엔진으로 기존 상용 DB 수준의 다양한 Rewrite Rule로 확장이 가능하다. 셋째, 적응형 최적화(Progressive Query Reoptimization)를 지원한다. 질의 실행 시간에 통계 정보를 기반으로 질의의 남은 부분을 최적화하여 나쁜 질의 계획을 회피한다.[16]

[표 2] 타조와 SQL-on-Hadoop 시스템의 비교

Name	Fault Tolerance	Dynamic Scheduling	Long time 질의 적합	자원 대비 큰 용량 처리
Tajo	O	O	O	O
Impala	X	X	X	X
Stinger (Hive)	O	O	O	O
Presto	X	X	X	X

3. 벤치마크

타조의 연구결과[16]에서는 벤치마크 테스트 성능이 하이브나 임팔라에 비해 더 빠르다고 발표하고 있다. 타조의 100G 데이터 대상 TPC-H 벤치마크 성능 테스트 결과는 그림 8과 같다. 대부분의 질의 문장에서 하이브에 비해 뛰어나며 임팔라와 비교하여도 유사하거나 뛰어난 결과를 보였다.



[그림 8] 타조의 벤치마크(TPC-H) 성능

자동차나 전자제품을 구입하면 에너지 효율에 대한 등급이 적힌 스티커가 붙어있다. 이것이 벤치마크의 대표적인 예이다. 소비자가 제품을 직접 구매하지 않고도 제품의 특징이나 성능을 알 수 있는 객관적인 수치를 제공하고자 하는 것이다. 데이터베이스의 성능을 검증하는 도구로서 역시 벤치마크 시험이 사용되고 있다. 대용량의 실 데이터(Real Data)를 사용하는 것이 효과적이나, 대용량의 실 데이터를 얻기 위해서는 많

은 시간과 비용이 소모된다. 따라서 효과적인 데이터베이스 벤치마크 시험을 위해서는 데이터 생성기를 이용하거나 실 데이터와 유사하게 생성하여 성능 테스트를 한다. 대량의 데이터시험 및 ad-hoc query의 성능 시험에 사용되는 벤치마크는 주로 TPC-H를 사용한다.

그러나 데이터베이스 성능이라는 것이 어떠한 데이터를 가지고, 어떠한 환경에서 어떻게 측정했는지에 따라 천차만별이다. 하드웨어, 운영체제, 네트워크등 많은 요인에 의해 달라질 수 있기 때문에 실제 환경에서 직접 돌려보기 전에는 성능을 미리 알기란 어려운 게 사실이다. 현재 SQL-On-Hadoop 기술들이 제시하는 대부분의 성능 수치는 일반적인 질의나 전체 질의에 대해서 평균 몇 배 빠르다가 아닌 자신들이 유리한 조건에서 테스트한 결과만을 언급하는 경우가 많다. 다른 연구에서 비교한 결과를 보면 대략 평균 3~5배 정도로 하이브에 비해 빠르고 질의의 종류에 따라는 수십배 정도 빠를 수도 있고, 더 느릴 수도 있다고 한다. 따라서 벤치마크 성능 테스트의 수치를 주요 언론, 블로그 등에서 제시한 수치라고 해서 맹신하는 것은 금물이다. 따라서 본 연구에서는 가능한 동일한 스펙의 하드웨어를 이용하여 RDBMS와 하둡 기반 기술의 성능비교 실험을 진행하고자 한다.

III. 스쿱과 타조의 효율적인 데이터 처리 성능

다양한 SQL-on-Hadoop 기술들이 타조와 마찬가지로 하이브와 비교하여 성능 테스트 결과를 발표하고 있지만 동일하거나 유사한 하드웨어 환경에서 RDBMS와의 성능을 비교하는 연구는 극히 드물다. 그래서 본 연구에서는 최근에 가장 주목 받고 있는 SQL-on-Hadoop 기술인 타조를 이용하여 RDBMS와 분석 성능을 비교하고 타조의 효율적인 데이터 처리 성능을 내기 위한 요소를 찾는 연구를 수행하였다. 데이터 분석 성능을 비교하기 전에 선행되어야 할 것이 분석을 위한 데이터를 수집하는 것이다. 일반적으로는 RDBMS 기반 업무시스템에서 수집된 데이터들을 분석하기 위하여 RDBMS로 구축된 DW로 적재하여 진행한다. 전체적인 시각에서 볼 때 데이터를 수집하는 것부터가 데이터 분석의 시작이기 때문에 분석성능을 비교하는 과정의 첫 단계로 원천 데이터가 저장되어있는 RDBMS에서 분석을 위한 RDBMS로 적재하는 성능과 스쿱을 사용하여 하둡 클러스터로 적재하는 성능 비교를 먼저 수행한다. 일반적인 운영환경에서는 분석을 위한 원천데이터를 확보하고 있으나 본 연구에서는 RDBMS에 테스트를 위한 임의의 데이터를 생성하여 성능을 측정한다. 성능 테스트는 표 3,4와 같이 4개의 테이블, 총합 1T 사이즈의 테스트 데이터를 생성하여 데이터 처리 성능을 측정하였다

.

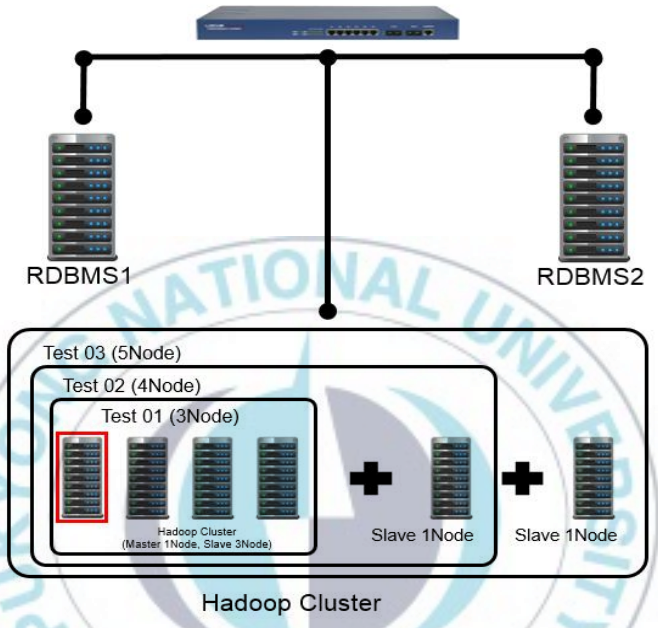
[표 3] 실험 테이블 크기 및 레코드 수

Table Name	Size(GB)	Records
BIG_ORDERS	103.5668	1,677,721,600
BIG_EMPLOYEES	146.359	1,677,721,600
BIG_ORDER_ITEMS	256.5738	11,156,848,640
BIG_CUSTOMERS	569.6716	5,033,164,800

[표 4] 실험 테이블 스키마

Table Name	Column Name	Type	Length
BIG_ORDERS	ORDER_ID	NUMBER	22
	ORDER_DATE	TIMESTAMP	11
	ORDER_MODE	VARCHAR2	8
	CUSTOMER_ID	NUMBER	22
	ORDER_STATUS	NUMBER	22
	ORDER_TOTAL	NUMBER	22
	SALES_REP_ID	NUMBER	22
BIG_EMPLOYEES	EMPLOYEE_ID	NUMBER	22
	FIRST_NAME	VARCHAR2	20
	LAST_NAME	VARCHAR2	25
	EMAIL	VARCHAR2	25
	PHONE_NUMBER	VARCHAR2	20
	HIRE_DATE	DATE	7
	JOB_ID	VARCHAR2	10
	SALARY	NUMBER	22
	COMMISSION_PCT	NUMBER	22
	MANAGER_ID	NUMBER	22
	DEPARTMENT_ID	NUMBER	22
BIG_ORDER_ITEMS	ORDER_ID	NUMBER	22
	LINE_ITEM_ID	NUMBER	22
	PRODUCT_ID	NUMBER	22
	UNIT_PRICE	NUMBER	22
	QUANTITY	NUMBER	22
BIG_CUSTOMERS	CUSTOMER_ID	NUMBER	22
	CUST_FIRST_NAME	VARCHAR2	20
	CUST_LAST_NAME	VARCHAR2	20
	NLS_LANGUAGE	VARCHAR2	3
	NLS_TERRITORY	VARCHAR2	30
	CREDIT_LIMIT	NUMBER	22
	CUST_EMAIL	VARCHAR2	30
	ACCOUNT_MGR_ID	NUMBER	22
	DATE_OF_BIRTH	DATE	7
	MARITAL_STATUS	VARCHAR2	20
	GENDER	VARCHAR2	1
	INCOME_LEVEL	VARCHAR2	20

실험에 사용된 서버의 시스템 구성도와 하드웨어 및 설치된 소프트웨어의 환경은 그림 9, 표 5,6 과 같다.



[그림 9] 실험 시스템 구성도

[표 5] RDBMS 서버 사양

항목	내용
O/S	Window 7
CPU	Intel(R) Core™ i5-2400 CPU @ 3.10GHz
RAM	4G (2G * 2)
HDD	1.5 T (500G * 3)
용도	Database Server
설치 SW	Oracle 11G

[표 6] 하둡 서버 사양

항목	내용
O/S	CentOS 6.4
CPU	Intel(R) Core™ i5-2400 CPU @ 3.10GHz
RAM	2G
HDD	500G
용도	Hadoop Cluster
설치 SW	Hadoop, Sqoop, Tajo

하둡 클러스터의 경우 3대의 슬래이브 노드와 1대의 마스터 노드를 기본으로 동일한 사양의 하드웨어를 1대씩 추가해 가며 5대의 슬래이브 노드가 될 때까지 반복적으로 테스트를 수행하였다. RDBMS에 사용된 제품은 시장에서 가장 많은 점유율을 차지하고 있는 오라클(Oracle) DBMS의 11G 버전을 사용했다.

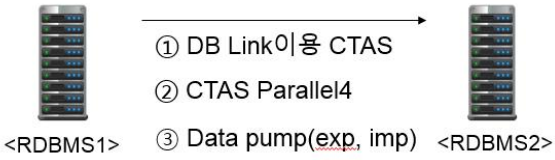
1. RDBMS대비 스쿱과 타조의 데이터 처리 성능

첫 번째 실험은 동일한 용량의 데이터를 RDBMS와 하둡 기반 시스템에서의 데이터 처리 성능을 비교하는 것이다.

1.1 스쿱의 데이터 처리 성능

RDBMS간의 데이터 적재 실험의 절차는 그림10과 같다. RDBMS에서 DW시스템으로 데이터를 전송할 때 사용하는 여러 가지 ETL(Extract, Transform, Load)솔루션이 존재하는데 솔루션을 사용하는 방법 이외에도

기본적으로 RDBMS에서 제공하는 도구도 존재한다. 이러한 도구를 사용하여 사용자가 배치프로그램을 작성하여 데이터를 전송하는 방법을 본 연구에서 사용하고자 한다.



① DB Link이용 CTAS
- 1T 데이터 (4 Tables) 를 DB link를 사용한 CTAS 방식으로 한번에 적재하는 PL/SQL 작성
- 각 테이블을 적재하기 전후 시각을 기록해두는 Log 테이블을 생성하여 Time 측정

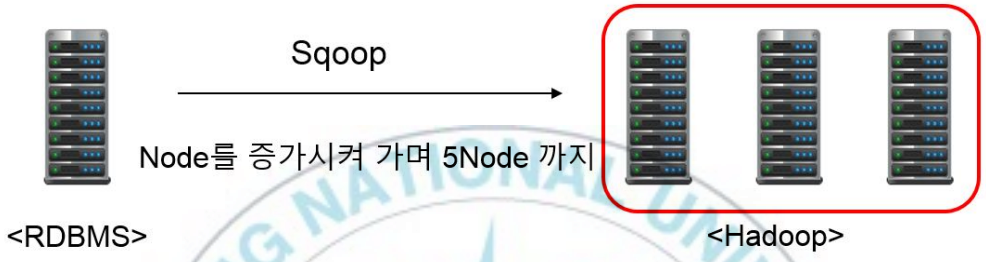
② CTAS parallel 4
- ①번과 같은 방식으로 parallel 옵션을 주고 실행

③ Data pump(exp, imp)
- 1T 데이터 (4 Tables) 를 Data pump를 사용하여 Import 시키는 방식으로 적재
- 각 테이블을 적재하기 전후 Log추적하여 Time 측정

[그림 10] RDBMS간 적재 시나리오

첫 번째 적재 실험은 데이터베이스 링크(DB Link)를 이용하여 전송 받는 2번 서버에서 전송하는 1번 서버의 테이블을 그대로 생성하는 방법이다. CTAS란 SQL 문법 중 Create Table as Select 구문을 의미한다. 데이터베이스 링크란 클라이언트 또는 현재 데이터베이스에서 네트워크상의 다른 데이터베이스에 접속하기 위한 접속 설정을 하는 오라클의 한 객체

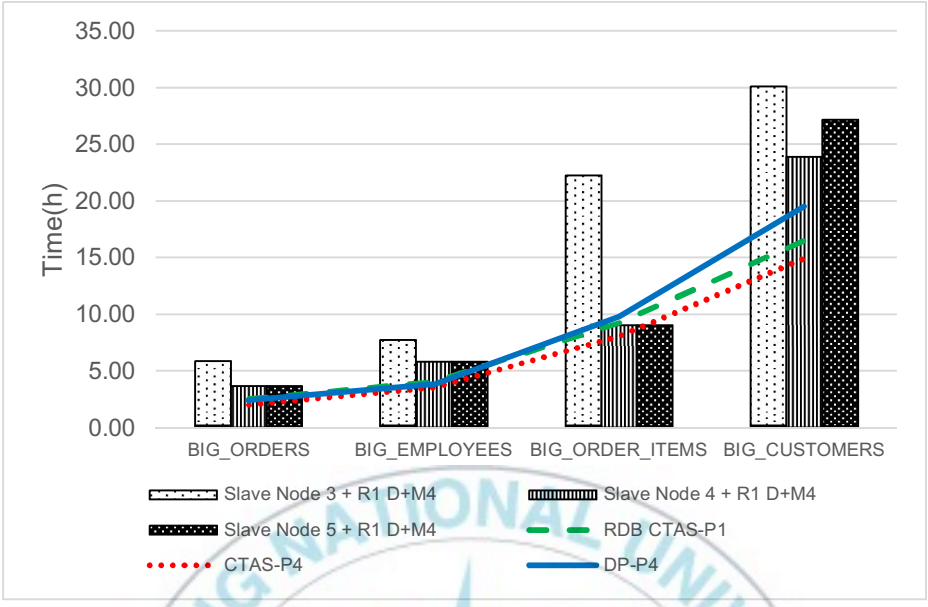
이다. 두 번째 실험은 첫 번째와 마찬가지 방식이나 병렬처리를 이용한 적재를 수행한다. 마지막으로 세 번째 방식은 오라클 RDBMS에서 대용량 데이터 전송을 위해 제공하는 툴인 Data pump를 사용하여 데이터를 적재하는 실험을 진행한다.



① SQOOP을 이용하여 RDBMS에서 Hadoop Cluster로 HDFS형식으로 적재
② Create external table(TAJO)로 Table 생성(HDFS와 연결)
③ 각 테이블 적재 시간 Log 추적하여 Time 측정

[그림 11] RDBMS와 하둡 클러스터간 적재 시나리오

RDBMS에서 하둡 클러스터로 적재하는 절차는 그림 11과 같다. 스쿱을 사용하여 하둡 클러스터의 HDFS 형태로 적재를 하는 방식이다. RDBMS간 적재 실험 시 병렬옵션의 수량을 4로 지정한 것과 마찬가지로 스쿱을 사용한 적재에서도 병렬 처리 역할을 수행하는 Mapper 수량을 4로 지정하여 테스트를 진행한다. 또한 RDMBS간의 적재와 동일한 양의 데이터를 적재하기 위해 하둡의 복제정책은 사용하지 않고 1T의 원천 데이터를 그대로 적재 하였다.



[그림 12] RDBMS와 하둡 클러스터의 적재성능 테스트 결과

RDBMS와 스쿱의 적재 성능 테스트의 전체 결과는 그림 12와 같다. 그림에서 볼 수 있듯이 RDBMS간의 적재 성능은 CTAS 방식으로 병렬 처리를 수행하였을 때 가장 빠른 속도를 보였고, RDBMS간의 적재와 RDBMS에서 하둡 클러스터로의 적재 속도를 비교하여 봤을 때도 RDBMS간의 적재 속도가 모든 방법에서 우수한 성능을 보였다. 조금 더 주목해서 볼 부분은 하둡 클러스터로의 적재실험에서 노드의 수량을 증가시키며 실험을 진행하였지만 더 많은 자원을 사용할 수 있는 5노드로의 적재 성능이 4노드에 비해 더 나은 성능을 보이지 못했고, Big_Customer 테이블의 경우에는 오히려 더 나쁘게 나타났다.

1.2 타조의 데이터 처리 성능

① 분석대상 Table 용량 측정
② 각 Table의 분석대상 Column 선정
③ 분석대상 Column의 Distinct Column Value와 빈도수를 Repository 로 넣는 Query 실행
④ 실행시간 전후 Time 측정하여 기록

[그림 13] RDBMS 분석 시나리오

① RDBMS 실험과 동일한 Query -> Tajo SQL 변환하여 작성
② Query실행
③ 실행시간 전후 Time 측정하여 기록
④ Slave 1Node 추가하여 동일한 과정 반복하여 Test02 실행 및 Time 측정
⑤ Slave 1Node 추가하여 동일한 과정 반복하여 Test03 실행 및 Time 측정

[그림 14] 하둡 클러스터 분석 시나리오

적재된 데이터를 대상으로 RDBMS 및 타조를 이용하여 분석성능을 실험하는 절차는 그림 13, 14와 같다. DW에서 주로 사용되는 질의문의 형태는 전체 로우(Row)를 대상으로 그룹별 집계결과를 출력하는 형태이기 때문에 그림15, 16과 같이 집계 결과를 Result_table 테이블에 삽입하는 질의문을 테스트 대상 질의문으로 선정했다. 타조에서 오라클의 SQL 문법을 완벽하게 지원하지 않기 때문에 두 테스트 질의문이 상이한 면이 존재하나, 전체범위를 처리하여 집계한 후 테이블에 결과를 삽입하는 처리방식은 동일하다

```
INSERT INTO mdq_rslt_data (rslt_id, st_date, col_value, frequency)
SELECT MAX(r.rslt_id), MAX(r.st_date), DATE_OF_BIRTH, COUNT(0)
  FROM BIG_CUSTOMERS,
       (SELECT RAWTOHEX(SYS_GUID()) rslt_id
             , TO_CHAR(SYSDATE, 'yyyymmddhh24miss') st_date
          FROM dual
         WHERE rownum = 1) r
 GROUP BY DATE_OF_BIRTH
```

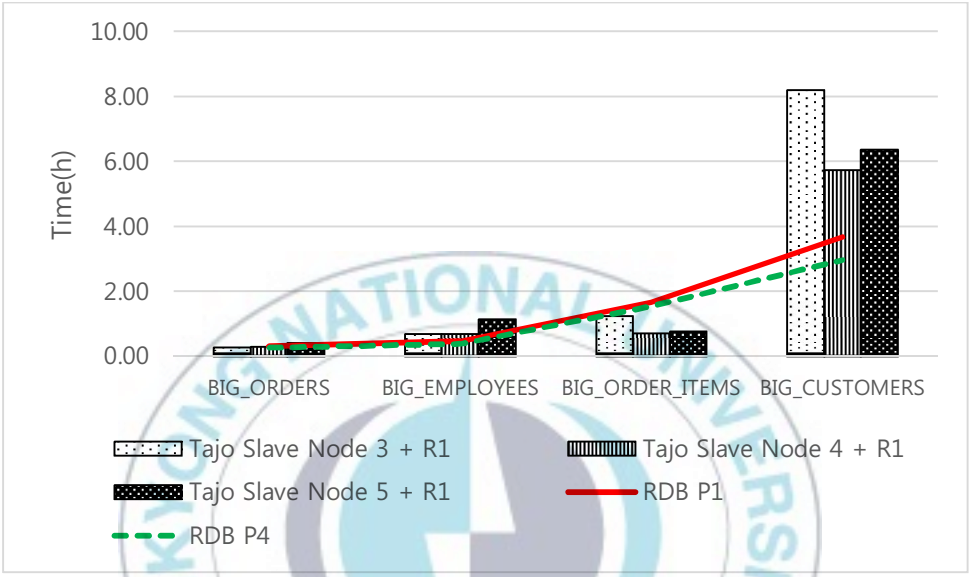
[그림 15] RDBMS 테스트 질의문

```
Create Table BIG_CUSTOMERS_rslt(col text, cnt int8);
INSERT OVERWRITE INTO BIG_CUSTOMERS_rslt (col, cnt)
SELECT DATE_OF_BIRTH, COUNT(*)
  FROM BIG_CUSTOMERS
 GROUP BY DATE_OF_BIRTH;
```

[그림 16] 타조 테스트 질의문

테스트 대상 테이블에서 임의의 컬럼을 선정한 후 그림 15의 RDBMS 테스트 질의문장에 대해서 병렬처리 옵션의 여부에 따라 두 번 수행하였다. PL/SQL을 이용하여 질의 문장을 수행하기 전후의 시간을 기록하였고 분석 성능 결과는 그림 17의 꺾은선 그래프와 같다. 병렬처리 옵션을 사용하지 않은 경우가 사용한 경우보다 다소 많은 시간이 걸렸으며 테이블의

전체 레코드 수와 상관없이 사이즈가 클수록 분석을 수행하는데 많은 시간이 소요됐다.



[그림 17] RDBMS와 하둡 클러스터의 분석성능 테스트 결과

타조의 로그를 추적하여 분석 성능을 측정한 결과는 그림 17의 막대 그래프와 같다. 일반적으로 노드를 추가하면 하드웨어 자원이 늘어나기 때문에 분석 성능이 선형적으로 향상되어야 함에도 불구하고 그림 17의 결과에서는 적재 실험의 결과와 마찬가지로 오히려 성능이 나빠진 경우가 발생했다. Big_Order_Items 테이블의 결과만 비교적 양호한 결과가 도출되었고 Big_Orders 테이블과 Big_Employees 테이블의 경우는 노드수가 늘어남에 따라 오히려 시간이 증가하였다. 또한 Big_Customers테이블은 크기에 비해 과도한 시간이 걸렸으며 대부분의 경우 좋은 결과를 얻지 못하였다.

테이블이 가장 작은 Big_Orders 테이블의 경우 RDBMS와 타조 모두 0.27시간으로 동일한 성능을 보였다. Big_Employees 테이블의 경우는 RDBMS에서 0.39시간, 타조 3노드에서 0.68시간으로 RDBMS의 성능이 더 뛰어났으며 Big_Customers 테이블의 경우 RDBMS에서 2.97시간 타조 4노드에서 6.37시간으로 타조에서 비교적 많은 시간이 소요되었다. Big_Order_Items 테이블의 경우는 RDBMS에서 1.55시간 타조 4노드에서 0.71시간으로 타조가 RDBMS보다 분석성능이 더 빠르게 나타났다. 전반적으로 RDBMS의 성능이 타조보다 좋은 결과를 보였다.

2. 효율적인 데이터 처리를 위한 스쿱과 타조의 영향요소

그림12의 적재성능 비교 결과와 그림17의 분석성능 비교결과 모두에서 자원을 많이 가용할 수 있는 하둡 클러스터 환경에서 더 나은 성능을 보일 것으로 예상했지만 반대의 결과가 나타난 이유에 대해 실험을 통해 분석한다.

2.1. 스쿱의 영향 요소

그림 12의 결과에서와 같이 SQOOP을 이용한 적재성능 실험에서 동일한 양의 데이터를 적재함에도 불구하고 더 많은 자원을 가용할 수 있는 하둡 환경에서 노드 수 에 상관없이 대부분 RDBMS 보다 낮은 성능을 보였다. 스쿱에서 사용할 수 있는 여러 가지 옵션들을 변경해 가며 실험 결과의 요인을 분석하고자 몇 가지 실험을 진행한다.

1) Direct 인자

스쿱은 기본적으로 가져오기 프로세스에서 합리적이고 벤더 중립적인 가져오기를 제공하고 자바 프로그램 내에서 데이터베이스를 연결해주는 응용프로그램 인터페이스인 JDBC를 사용한다. 그러나 스쿱은 몇몇 데이터베이스에 대해서 Direct 인자를 통해 데이터베이스에서 제공하는 데이터 이전 도구를 사용하여 빠른 속도의 데이터 전송 성능을 제공는 옵션을 제공한다. 그림 12의 실험에서는 Direct 인자를 사용하지 않았기 때문에 더 나은 성능을 위해 Direct 인자를 이용한 실험을 추가적으로 진행하였다.

[그림 18] Direct 옵션 테스트 결과

그러나 그림 18의 결과와 같이 성능 향상의 효과를 볼 수 없었는데, 이는 Direct 옵션을 제공하는 데이터베이스가 제한적이기 때문이다. 본 실

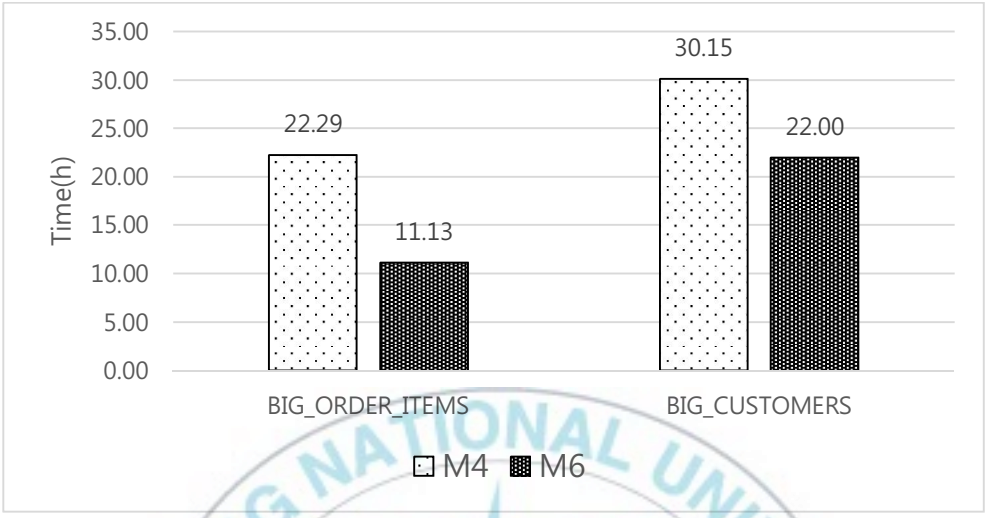
험에 사용한 오라클 DBMS는 Direct 옵션을 제공하지 않기 때문에 성능 향상의 효과를 볼 수 없었지만, MySQL이나 PostgreSQL처럼 Direct 옵션을 제공하는 RDBMS를 이용하여 데이터를 전송한다면 보다 나은 성능을 얻을 수 있을 것으로 예상한다.

2) Mapper 수량

스쿱을 사용하여 RDBMS에 있는 데이터를 하둡으로 전송할 때 HDFS의 모든 저장 현황을 파악하고 있는 하둡의 네임노드가 내부적인 알고리즘에 의해서 데이터 노드에 있는 Mapper에게 적재 작업을 할당해 준다. Mapper가 네임노드의 지시에 따라서 데이터를 지정된 위치로 적재하는 작업을 수행한다. 하나의 데이터 노드에서 여러 개의 Mapper를 생성하여 동시에 작업할 수 있기 때문에 RDBMS의 Parallel Degree 옵션과 비슷하다고 생각할 수 있다. Parallel Degree는 CPU의 개수 및 데이터 량에 따라서 적절하게 지정해 주어야 하는 것처럼 Mapper의 수 또한 노드의 개수 및 각 서버의 하드웨어 성능에 따라서 적절하게 지정해 주어야 한다. 즉 Mapper의 수량을 지정하는 옵션은 데이터 노드의 수량과 밀접한 관계가 있다는 것이다.

Mapper의 수량에 따른 성능 변화를 측정하기 위해 두 가지 테스트를 진행하고자 한다. 먼저 데이터 노드의 수량을 고정시킨 뒤 Mapper의 수량을 늘려가면서 적재시간을 측정하여 성능의 변화를 알아보고, 반대로 Mapper의 수량을 고정시킨 뒤 데이터 노드의 수를 늘려가면서 Mapper와 데이터 노드 수량간의 관계가 성능에 미치는 영향을 알아보고자 한다.

1) 동일 수량 데이터 노드에서 Mapper수량 변경에 따른 성능



[그림 19] 3노드 에서 Mapper 수량에 따른 성능비교 결과

3개의 데이터 노드로 구성된 환경에서 Mapper의 수량을 4와 6으로 변경하며 테스트를 진행하였다.

그림 19에서와 같이 Mapper의 개수를 4에서 6으로 늘린 후의 적재 성능은 평균 39% 정도 향상되었다. 결과를 통해서 Mapper의 개수를 늘린다면 적재 성능을 크게 향상시킬 수 있다는 사실을 확인할 수 있지만 Mapper는 병렬처리와 유사하게 동작하기 때문에 하드웨어 사양을 고려하지 않고 높게 설정한다면 오히려 더 낮은 성능이 나타날 수 있다.

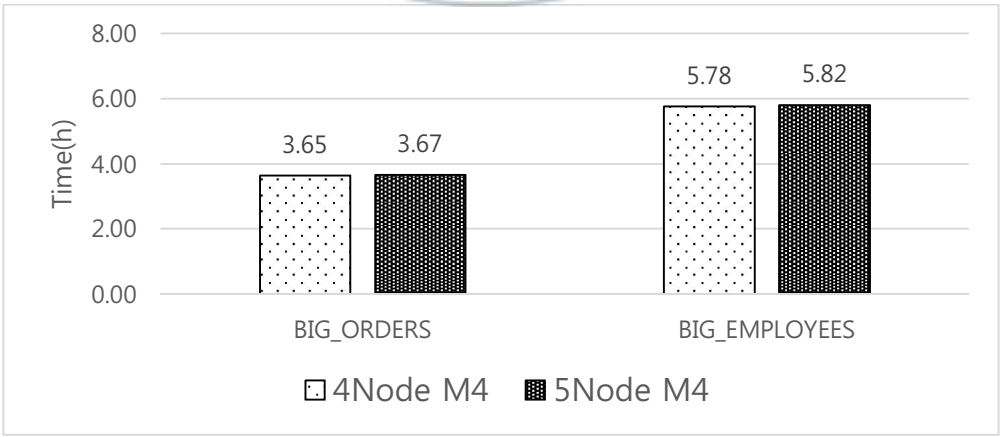
5대의 데이터 노드에서 Mapper 수량을 4, 8, 20으로 변경한 실험의 결과인 그림 20에서 확인할 수 있듯이 Mapper의 수를 한계치 이상으로 설정한다면 작업이 먼저 끝난 Mapper가 더 이상 작업을 하지 않고 하드웨어 자원만 차지하고 있기 때문에 Mapper 수량이 20개 일 때 오히려 가장 나쁜 성능을 발휘한 것을 볼 수 있다.

[그림 20] 5노드 Big_Orders 테이블의 Mapper수량에 따른 성능비교 결과

2) 동일 수량 Mapper에서 데이터 노드 수량 변경에 따른 성능

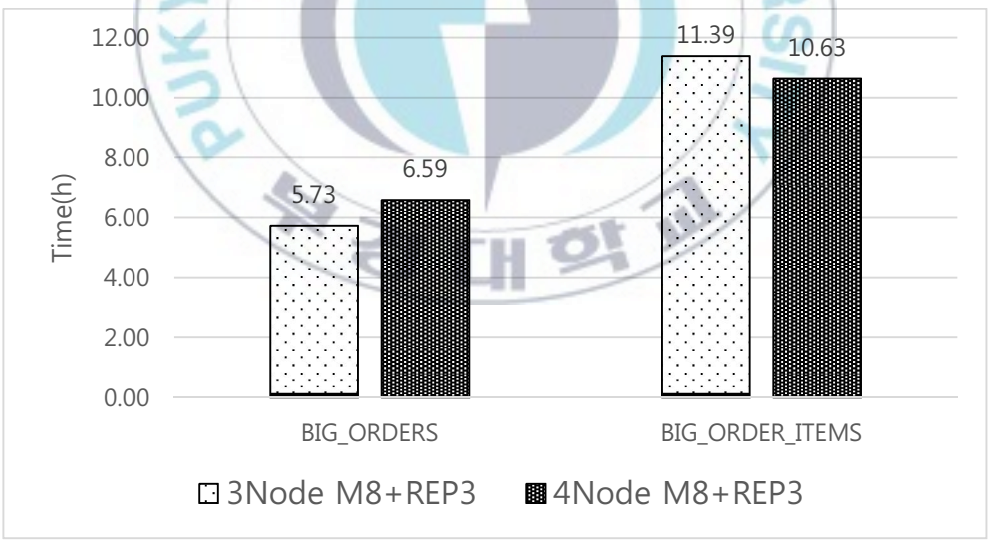
하둡은 분산 처리 환경에서 데이터 노드를 적은 비용으로 비교적 자유롭게 추가할 수 있다는 장점이 있다. 그렇기 때문에 Node의 개수는 적재의 성능을 향상시킬 수 있는 요소가 될 수 있다.

이 단계에서 Mapper 수를 동일하게 4로 설정해 둔 상태에 슬레이브 노드 개수를 4로부터 5개까지 추가해가면서 적재 시간을 측정하였다.



[그림 21] Mapper4에서 노드수 변화에 따른 성능비교 결과

4개의 Mapper로 설정한 4, 5 데이터 노드에서의 실험 결과인 그림 21에서 사용가능 한 하드웨어 자원이 증가하였음에도 불구하고 성능 향상의 효과를 볼 수 없었다. 그림 22의 결과는 3개의 노드에서 Mapper 수량을 8로 설정하고 하둡의 기본 복제정책을 사용한 것과, 4개의 노드에서 나머지 환경을 고정시키고 실험한 결과 이다. 앞선 실험과 비교하여 볼 때 복제 정책을 사용한 것이 다른 점이지만 본 결과에서는 노드 수량에 따른 변화를 알아보고자 한다. 하둡의 복제정책이 적재 성능에 미치는 영향에 대해서는 4.3절에서 실험한다. 그림 22의 결과와 같이 4개의 데이터 노드를 사용한 Big_Orders테이블에서 3개의 노드를 사용한 것 보다 더 나쁜 성능이 측정되기도 하였고 Big_Order_Items 역시 큰 성능향상효과를 볼 수 없었다.

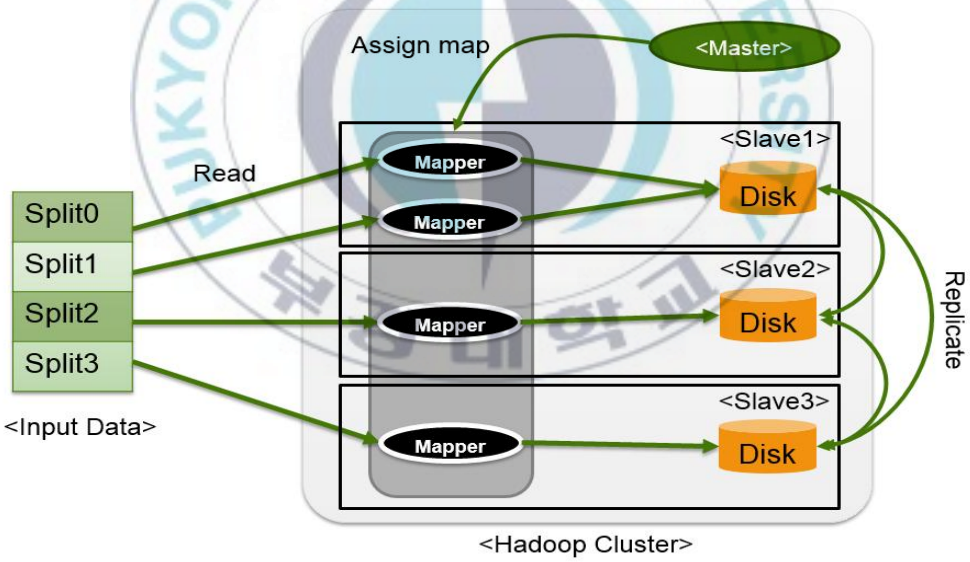


[그림 22] Mapper8 + Rep3에서 노드수 변화에 따른 성능비교 결과

앞서 그림 19, 20의 결과에서 동일 수량 노드에서 Mapper 수량을 증가

시킬 때 성능이 향상되지만 가용자원을 넘어가는 Mapper 수량을 지정할 경우에는 오히려 나쁜 성능을 얻을 수 있다는 결론을 얻었고, 그림21, 22의 결과에서 노드수를 증가하더라도 적절한 Mapper 수량을 설정하지 않는다면 성능향상효과를 볼 수 없다는 결론을 얻을 수 있다. 결론적으로 스쿱을 이용한 하둡으로의 데이터적재에서 중요한 요인은 노드의 수만 무작정 증가시키는 것이 아니라 데이터 노드에 비례하여 적절하게 Mapper의 수량을 설정하는 것이 성능 향상을 위한 중요한 요인이라 할 수 있다.

3) Split-by인자에 의한 구분 컬럼



[그림 23] 분할컬럼을 이용한 스쿱의 부하 분산

고속 전송을 하기 위한 Direct 옵션과 작업을 수행하는 Mapper의 수량을 적절하게 설정하는 것도 중요하지만, Mapper에게 어떠한 기준으로 작

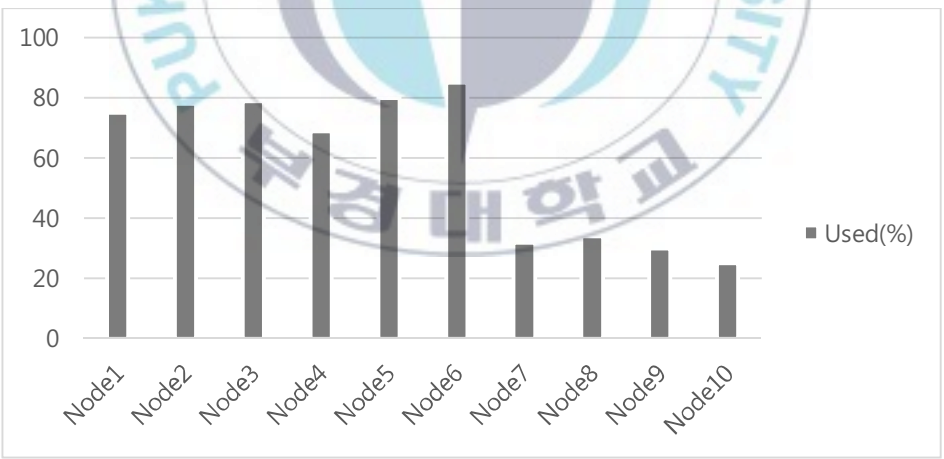
업을 분할 할 것인지 명시해 주는 것 또한 중요한 사항이다. 스쿱이 병렬 가져오기를 수행할 때, 부하를 분할할 수 있는 기준이 필요하다. 스쿱은 부하를 분할하기 위해 분할 컬럼을 사용한다. 그림 23은 분할 컬럼을 이용한 스쿱이 부하를 분산시키는 과정을 보여준다. 스쿱은 기본적으로 제공되는 주 키(Primary Key) 컬럼을 테이블에서 인식하고 그것을 분할 컬럼으로서 사용한다. 분할 컬럼의 가장 큰 값과 가장 작은 값들을 데이터베이스로부터 가져오고 이를 기준으로 맵 태스크들이 수행할 작업의 범위를 전체 범위에서 균등하게 크기가 나누어 작동시킨다. 예를 들면, 주 키 컬럼인 id가 최소값이 0이고 최대값이 1000인 테이블이 있을 때, 스쿱은 4개의 태스크를 정해서 4개의 프로세스가 동작되도록 한다고 했을 때, 각각의 프로세스는 SELECT * FROM sometable WHERE id >= low AND id < high 형태의 SQL문의 (low, high)를 (0, 250), (250, 500), (500, 750), (750, 1001)로 다르게 설정하여 실행하게 된다.

대부분의 RDBMS에 생성된 테이블의 주 키에 기본키 제약조건이 생성되어 키 값을 중복을 허용하지 않지만 운영의 편의성 및 사용자의 조작 미숙으로 인해 제약조건을 설정하지 않는 경우도 존재한다. 이 경우는 키 값의 중복이 허용되기 때문에 특정값을 가진 로우들이 많이 존재 할 가능성이 있으며 실제로 일정하게 분산되어 저장되어있지 않다면 불균형한 태스크를 초래하게 된다. 키 값의 분포도가 좋지 않거나 기본키가 존재하지 않는 테이블의 데이터를 적재하는 경우에는 --split-by인자로 다른 컬럼을 명시적으로 선택해주어야 한다. 스쿱은 또한 현재 다중 컬럼 인덱스에 대하여 분할할 수 없다. 테이블이 다중 컬럼 키를 가지고 있다면, 분할 컬럼을 명시적으로 직접 선택해야만 한다. 본 실험에서는 가상으로 테스트를 위한 데이터를 생성하였기 때문에 기본키 제약조건을 생성하지 않았고 명시적으로 분할 컬럼을 선택하여 실행하였다. 본 연구에서는 분포도가 나쁜 분할컬럼을 선정하였을 때 적재성능을 테스트 하지 못하였

으나, 스쿱의 적재 프로세스에 관련된 연구에서 이와 같은 사항을 언급하고 있으며, 실제 테스트는 추후 연구과제로 미룬다.

2.2. 타조의 영향 요소

타조에서 RDBMS와 분석 성능을 비교하였을 때 보다 좋은 성능을 보이지 못한 이유는 하둡 클러스터의 모든 하드웨어 자원을 효율적으로 사용하지 못한데 이유가 있다. 앞선 분석 성능 결과는 스쿱을 통해 RDBMS의 데이터를 하둡 클러스터상으로 적재를 할 때 적재시간을 단축하기 위해서 하둡의 복제 정책을 사용하지 않고 원본만 저장하는 전략을 사용한 결과다. 스쿱은 RDBMS에 저장된 데이터를 하둡 파일시스템으로 쉽게 옮길 수 있도록 도와주는 오픈 소스 도구이다.



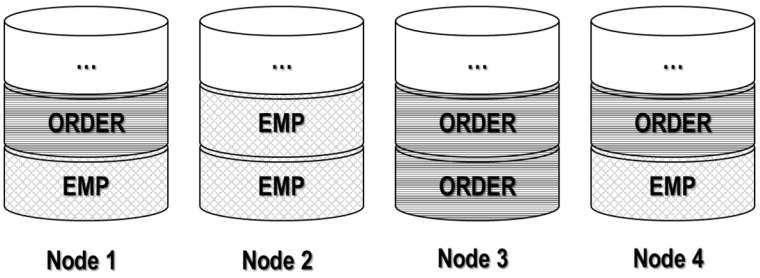
[그림 24] 데이터 노드 사용률

스쿱은 적절한 내부 알고리즘에 의해 작업을 분산하여 하둡 클러스터상으로 데이터를 옮기는데 데이터가 모든 노드에 균등하게 분배된다는

보장은 없다. 원본 데이터가 각 노드에 균등하게 적재되어있지 않은 상태에서 데이터 분석을 수행한다면 올바른 분석 성능 효과를 이끌어 낼 수 없다. 그림 24에서와 같이 특정 노드(Node1~Node6)에만 데이터들이 더 많이 쌓여있다면 그 노드에서만 많은 작업을 수행하게 된다. 앞선 그림 17의 결과는 작업량의 불균형으로 인해 하둡 클러스터 내의 모든 자원을 효과적으로 사용하지 못하였기 때문에 나타난 결과로 볼 수 있다. 그렇기 때문에 클러스터의 모든 데이터 노드에서 작업을 균등하게 처리할 수 있게 사용자가 적절한 전략을 설정할 필요가 있다.

작업을 균등하게 처리하게 하도록 부하를 분산시키는 방법(Load Balancing)에는 두 가지가 있다. 첫째 하둡에서 제공하는 Balancer를 통해 데이터를 균등하게 분배시켜주는 방법이다. Balancer는 각 노드들에 저장된 데이터들을 균등하게 분배시켜주는 역할을 한다.

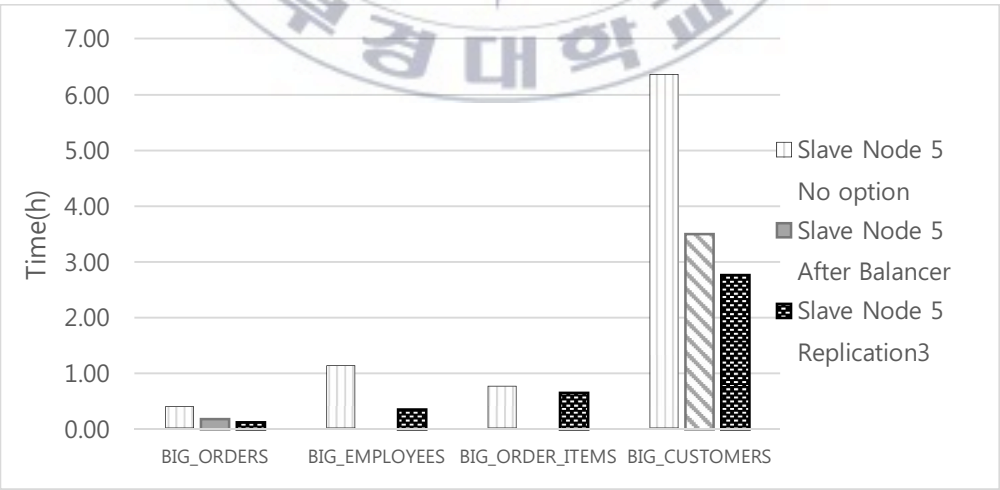
그러나 Balancer를 통하여 균등하게 분배하였다고 하더라도 처리하고자 하는 데이터는 특정 노드에 몰려 있을 수 있다. 그림 25에서와 같이 데이터가 균등하게 분배되어있다 하더라도 EMP 테이블의 데이터들은 노드 1, 2, 4에 저장되어 분석을 수행하는데 클러스터의 모든 자원을 모두 사용할 수 없게 된다.



[그림 25] 동일 데이터 량 분배 상태

2.3. 하둡 복제 정책

작업을 균등하게 분배시키는 두 번째 방법은 적절한 하둡 복제 정책을 설정하는 것이다. 하둡은 내 고장성(Fault tolerance)과 고가용성(High Availability)을 위해 효율적인 데이터 복제 알고리즘을 가지고 있다. 이 복제 정책을 이용하여 분석하고자 하는 데이터가 여러 노드에 복제되어 분산되어있다면 하둡은 각각의 노드들이 가지고 있는 데이터들의 메타정보를 이용하여 적절하게 작업을 분배할 수 있게 된다. 사용자가 적절한 복제 정책 전략을 설정하여 알맞은 복제 레벨을 지정해 주어야 한다. 복제 레벨은 실제 사용되는 데이터 노드의 수 보다 작은 값을 설정해야 한다. 레벨이 높을수록 많은 복제본을 생성하기 때문에 저장공간을 많이 차지하지만 높은 고가용성을 지원한다. 사용자의 시스템 사양이 충분하다면 성능을 위해서 데이터 노드수에 가까운 레벨을 설정하는 것이 성능 및 안정성에 유리하다. 하둡의 복제 정책 레벨의 기본값은 3이다. 원본 데이터에 대해 추가 2개의 데이터를 복제하여 분산시킨다.

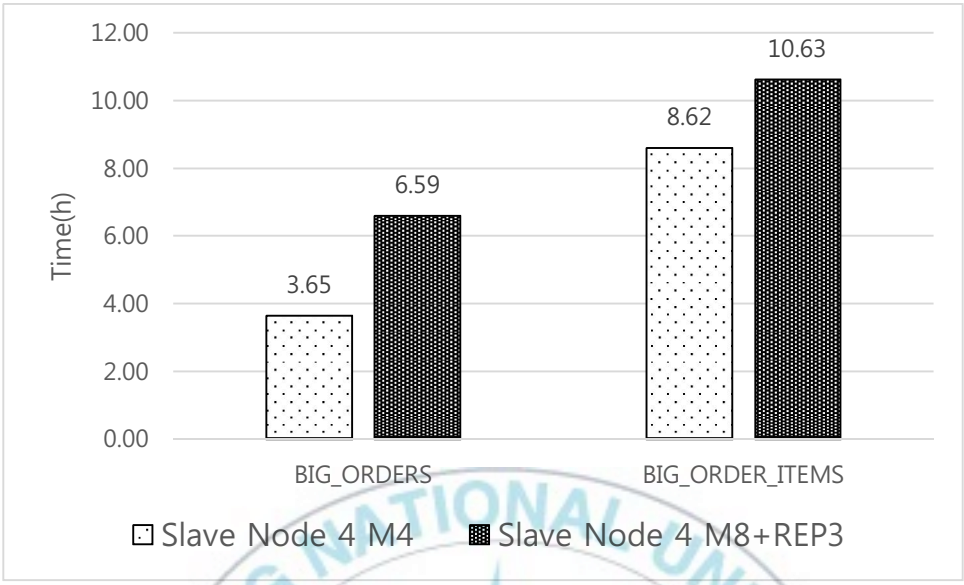


[그림 26] 균등분배 후 데이터 분석성능 비교 결과

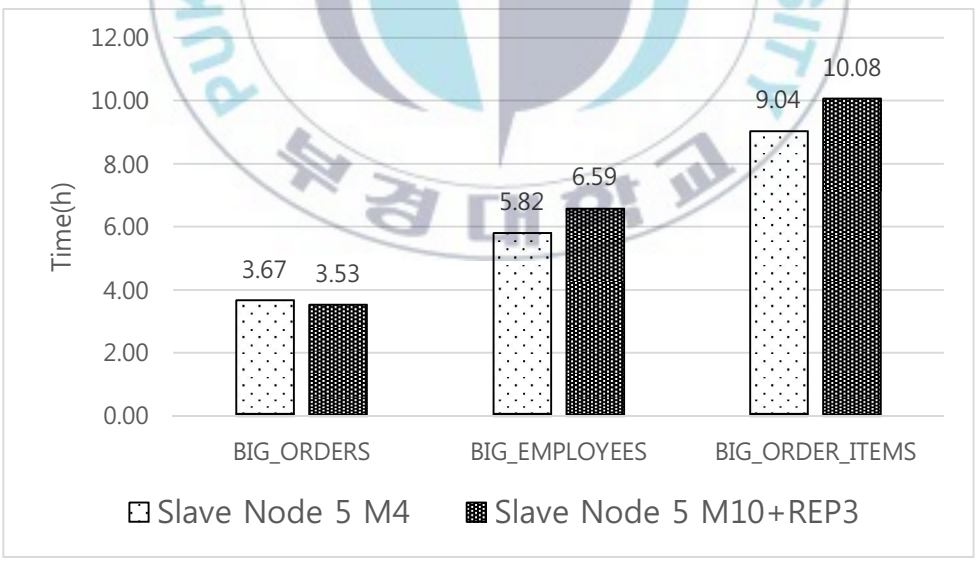
본 연구의 실험에서는 복제 정책 레벨을 기본값으로 설정하였고, Balancer를 통한 데이터 분배 후 데이터 분석, 복제 정책을 사용한 후 데이터 분석 결과 비교는 그림 26과 같다.

복제 정책을 사용하지 않은 환경에서의 분석 성능에 비해 Balancer를 이용한 데이터 밸런싱 후의 분석 성능이 보다 빠르며, 복제 정책을 사용한 후의 분석 성능이 밸런싱 후의 분석 성능 보다 더 빨랐다. Big_Order_Items 테이블에서는 비교적 적은 성능 향상을 보였는데 이 테이블의 경우는 초기 분석시에도 RDBMS보다 좋은 성능을 보였다. 이는 스쿱을 통해 적재를 수행할 때 분석할 데이터들이 균등한 비율로 분산되어 있어 좋은 성능을 보였다는 사실을 알 수 있다. 복제 정책을 사용할 때, 각 노드에 데이터가 분산되어 저장되어있어 분석 성능을 향상 할 수 있으나, 복제정책으로 인해 더 많은 데이터를 저장하기 때문에 데이터 적재성능에도 영향을 미치게 된다. Replication값을 3으로 설정하였을 때 1T의 데이터가 실제 디스크에 저장된다면 데이터의 용량이 3TB가 된다. 복제정책을 사용하게 되면 더 많은 데이터를 저장해야 하기 때문에 당연히 더 많은 시간이 소요되기 마련이다.

다음의 그림 27의 결과에서, 동일하게 4노드 환경에서 Mapper의 수량을 변경한 실험 결과를 보면, Mapper의 수량이 적절하게 늘어나서 적재 성능이 빨라져야 함에도 복제 정책을 사용함으로써 더 많은 데이터가 적재되기 때문에 Mapper가 8인 실험에서 더 느린 성능이 나타났다. 마찬가지로 5노드의 실험 결과인 그림 28에서도 Mapper수가 늘어났음에도 불구하고 더 나쁜 성능이 나타난 사실을 확인할 수 있다.



[그림 27] 4노드에서의 복제정책에 따른 성능 비교



[그림 28] 5노드에서의 복제정책에 따른 성능 비교

앞서 그림 12의 적재 성능 테스트를 수행하였을 때는 RDBMS에 저장되어 있는 데이터를 동일하게 옮기기 위해서 복제본의 수를 1로 변경하여 적재 실험을 진행하였다. 그러나 실제 운영 환경에서는 하둡의 복제본의 수를 1로 설정해 사용하는 경우는 테스트 환경을 제외하곤 극히 드물다. 하둡 클러스터의 노드 개수 및 데이터 분석할 때의 올바른 성능을 고려하면 최소한 기본값 이상으로 설정하여야 한다.

IV. 스쿱과 타조의 데이터 처리 성능 실험 결과

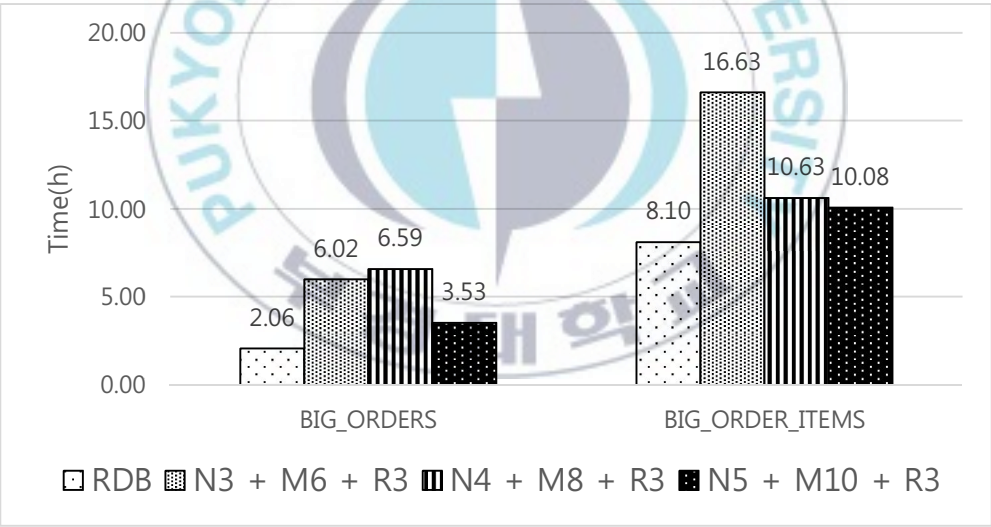
하둡 클러스터에서의 적재 및 분석성능 실험결과를 개선하기 위해 여러 가지 테스트를 통해 성능영향 요인을 알아보았고 실험 환경에 적합하도록 설정을 변경 한 후 적재 및 성능테스트를 진행하여 RDBMS와 비교한다.

1. 스쿱의 데이터 처리 성능 실험 결과

앞선 스쿱의 적재성능의 결과를 분석한 실험결과를 토대로 도출한 스쿱을 이용한 데이터 적재를 수행할 때의 올바른 사용 전략은 다음과 같다. 첫째, Direct 옵션을 지원하는 DBMS를 사용할 경우 Direct를 사용한다. 둘째, 시스템 사양, 적재 데이터 크기, 노드 수를 고려하여 적절한 Mapper 개수를 지정한다. Mapper 수량을 증가시켜야 한다는 것을 실험적으로 보였으나 시스템 사양, 데이터 크기 및 노드 수 대비 최적의 Mapper수를 찾는 문제는 여전히 존재한다. 이 문제에 대한 해결방안은 추후 연구과제로 미룬다. 셋째, Primary key가 존재하지 않거나 다중 컬럼으로 구성된 경우 -split by에 명시적으로 분포도가 좋은 컬럼을 선택한다. 넷째, 스쿱의 처리 속도를 향상시킬 수 있는 JDBC 드라이버(Progress Data Direct Type 5 JDBC Driver)를 사용한다. 다섯째, 모든 Node에 균등하게 적재시킬 수 있다면 복제정책을 사용할 필요가 없다. 복제정책은 분석 성능에 영향을 미치기 때문에 상황에 맞게 복제정책 사용해야 한다.

이와 같은 전략으로 적재성능을 개선하기 위한 실험을 진행 하였다. RDBMS로의 적재 방식은 가장 빠른 성능을 보였던 CTAS방식으로 병렬처리

로 수행하였고, 하둡 클러스터로의 적재는 각각 노드 수량에 비례하도록 Mapper 수를 늘려가며 복제정책을 사용하여 실험을 진행하였다. 실험의 결과는 그림 29와 같다. 복제정책을 사용한 이유는 분석성능의 향상을 위함이라는 것을 앞서 언급하였다. 결과에서와 같이 RDBMS에서 병렬 처리한 결과가 가장 좋은 결과를 보였다. 또한 이는 5개 노드로 구성된 하둡 환경에서 Mapper 개수가 10일때의 적재 성능보다 평균 30% 정도 빠른 것도 확인할 수 있다. 복제정책으로 인해 더 많은 데이터를 적재하는 것이기 때문에 당연한 결과이다. 한편, 최초의 실험과 달리 하둡 클러스터의 데이터 노드 개수 및 Mapper 개수가 늘어나면서 성능이 점차 향상되는 결과를 볼 수 있다.



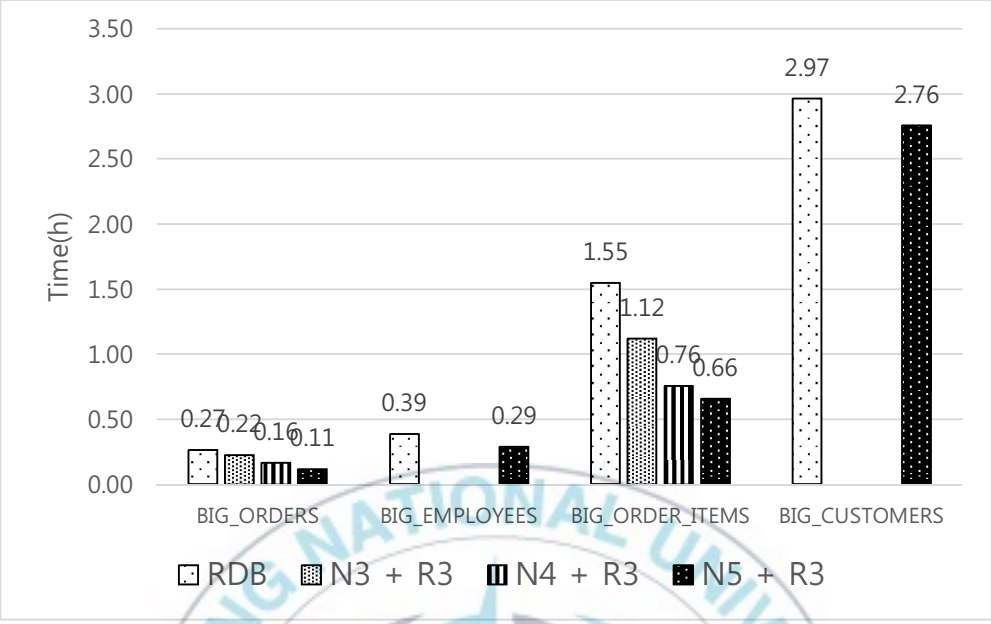
[그림 29] 5노드에서의 복제정책에 따른 성능 비교

앞서 분석한 요인들을 고려하여 종합적으로 비교한 결과를 볼 때 하둡 클러스터에서 운영환경에서 복제 정책을 사용하는 것을 고려하여 더 많

은 데이터가 적재되기 때문에 너무 당연하게도 RDBMS의 적재성능이 더 빠르다고 볼 수 있다. 그렇지만 데이터 노드의 수량을 증가시키고 그에 맞게 Mapper 수량을 일정하게 늘림에 따라서 선형적으로 스쿱을 사용한 하둡으로의 적재성능이 향상되는 것을 볼 수 있기 때문에 적은 비용으로 하둡의 하드웨어 장비를 추가시킬 수 있다는 장점을 이용하여 더 많은 노드로 구성된 하둡 클러스터를 이용한다면 스쿱의 적재 성능이 많이 향상되어 더 많은 데이터를 적재함에도 불구하고 RDBMS와 비슷한 성능을 보이거나 그보다 나은 성능을 보일 것으로 예상한다.

2. 타조의 데이터 처리 성능 실험 결과

타조의 경우 스쿱과 달리 질의문을 처리하는 과정에 사용자가 관여할 수 있는 환경설정이 존재하지 않는다. 그러나 처리해야 할 데이터가 하둡 클러스터의 데이터 노드에 균등하게 분배된 상태라면 최초의 실험보다 나은 결과를 보일 것이라는 것을 실험적으로 보였다. 균등하게 데이터를 분배하는 방법에는 하둡에서 제공하는 Balancer 도구를 이용할 수도 있지만, 근본적인 해결책은 HDFS의 복제정책을 사용하는 것이다. 복제정책을 사용할 때의 이슈는 스쿱을 통한 적재에서와 같이 시스템사양, 총 데이터 크기, 하둡 클러스터의 노드 수에 따라 적절한 복제수량을 지정해야 한다는 것이다. 상황에 맞는 복제정책의 복제수량을 결정하는 문제에 대한 해결 또한 추후 연구과제로 미루고, 본 연구에서는 하둡의 기본정책인 복제수량 3으로 설정한 뒤 분석을 수행하였다.



[그림 30] 성능 개선 후 분석성능 비교

앞의 실험 결과에 근거하여 복제 정책을 사용하여 부하를 분산 시킨 뒤 올바른 분석 성능을 내도록 수정환경을 만든 후 분석 결과는 그림 30과 같다. 개선 후의 성능은 5대의 노드에서 최대 2배 이상의 성능 향상을 보였고, RDBMS와 비교해 보더라도 3노드에서부터 더 나은 성능을 보인다. 또한, 노드 수의 증가에 따라 선형적으로 성능이 빨라짐을 알 수 있다.

또한 개선을 시킨 환경에서 그림 31과 같이 추가적인 SQL 질의문을 대상으로 각 테이블을 대상으로 테스트를 진행하였는데 그림 30과 유사한 결과를 보였고 추가적인 3개의 질의문을 수행하는데 걸린 시간에 대한 평균적인 시간을 나타낸 그림은 그림 32과 같다.

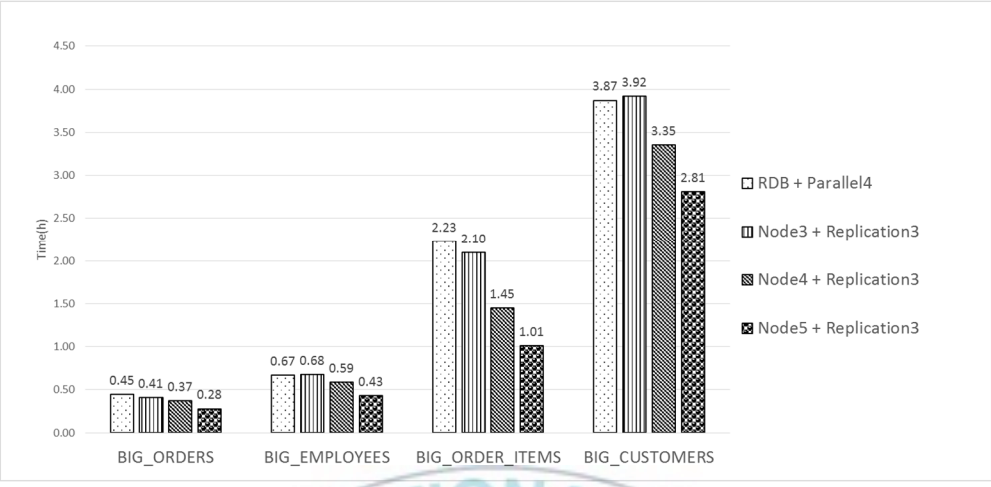
```
SELECT col_name1, AVG(col_name2), SUM(col_name3*0.7)
  FROM tab_name
 GROUP BY col_name1

SELECT col_name1, COUNT(*),
       MAX(CASE WHEN col_name2 >= 'value' THEN col_name2 + value
           ELSE 0 END),
       MIN(CASE WHEN col_name2 <= 'value' THEN col_name2 + value
           ELSE 9999 END),
       SUM(col_name3 * (1 - col_name4))
  FROM tab_name
 GROUP BY col_name1

SELECT col_name1, COUNT(*),
       MAX(CASE WHEN col_name2 >= 'value' THEN col_name2 + value
           ELSE 0 END)
  FROM tab_name
 GROUP BY col_name1
```

[그림 31] 추가적인 타조 테스트 질의문

그림 32의 결과에서도 볼 수 있듯이 RDBMS에서의 질의수행시간 보다 타조의 노드수 증가에 따른 수행시간이 더 빠른 것을 볼 수 있다.



[그림 32] 추가적인 질의문 테스트 결과

V. 결론

하둡의 등장 이후 다양한 형태의 대용량 데이터를 효율적으로 처리하고 분석하는 표준 기술로 채택되고 있지만 아직도 다양한 분야에서 관계형 데이터베이스 시스템을 이용하여 데이터 분석을 수행하고 있다. 이는 하둡 기반 솔루션들의 성능이 관계형 데이터베이스에 비해 얼마나 나은 결과를 보이는지에 관한 연구가 미미하기 때문이다. 본 연구에서는 하둡 기반 기술을 이용한 데이터 분석성능이 RDBMS대비 어느 정도의 성능을 보이는지를 동일한 하드웨어를 사용하여 비교하는 실험을 진행하였다. 데이터 분석의 첫 단계로 데이터를 적재하는 과정이 필요하기 때문에 데이터 적재 또한 분석의 한 과정으로 보고 RDBMS에서 확보한 데이터를 RDBMS서버와 하둡 클러스터로 각각 적재하는 실험을 하였다. 적재성능 비교는 하둡에서 가장 많이 사용하는 스쿱을 이용하였고, 분석성능 비교에서는 최근 주목받고 있는 SQL-on-Hadoop 기술의 일종인 타조를 이용하여 관계형 데이터베이스와의 성능 비교를 수행하였다.

최초의 적재성능 비교에서는 RDBMS에서 더 나은 성능을 보였고 하둡 클러스터의 노드 수 증가에 따른 성능향상결과를 볼 수 없었다. 분석성능을 비교한 결과 역시 RDBMS에서 타조보다 나은 성능을 보였다. 또한, 하둡 노드수 증가에 따른 성능향상결과도 볼 수 없었는데, 이는 모든 하드웨어 자원을 가용하지 못하거나 부하를 분산시키지 못했기 때문이다. 스쿱을 사용하여 적재할 때 노드 수에 따라 적절하게 Mapper의 수량을 증가시켜 자원을 사용할 수 있도록 하였고, 타조를 통해 분석할 경우에는 복제정책을 사용하여 부하를 분산시킨 후 향상된 성능 결과를 얻도록 하였다. 결론 적으로 본 연구의 실험에서는 적재시 RDBMS에서 더 나은 성능을 보였으나 하둡의 노드 증가에 따라 성능이 선형적으로 증가함을

알 수 있었고 이는 하둡의 클러스터가 확장됨에 따라 복제정책을 사용하여 RDBMS보다 더 많은 데이터를 적재 하더라도 더 나은 성능을 보일 수 있다는 것을 의미한다. 데이터 분석 성능은 하둡 3노드에서부터 RDBMS의 성능보다 뛰어난 결과를 나타냈다.

실제 분석을 하는 환경을 고려해 볼 때, 원천 데이터를 분석 시스템으로 적재하는 한번의 과정을 거쳐, 분석 시스템에서 반복적으로 다양하게 분석 활동을 진행 할 것이다. 물론, 하둡 클러스터가 확장 됨에 따라 적재 성능 또한 RDBMS로의 적재 성능보다 좋아질 수 있음을 보였지만, 더 나쁘다고 하더라도 반복적으로 분석을 수행 할 때 얻을 수 있는 이익이 많다면 하둡 클러스터에서의 분석을 충분히 고려해 볼 수 있다는 것을 의미한다.

하둡 기반 솔루션인 스쿱과 타조를 본 연구의 5장에서 언급한 바와 같은 전략을 세워 분석을 수행한다면 관계형 데이터베이스 보다 나은 성능을 얻을 수 있어 RDBMS와 유사하거나 보다 나은 성능을 얻을 수 있다는 것을 본 연구에서 실험적으로 보인 것이다. 그러나 하둡 기반 솔루션 들은 운영 시스템에 적용하기 위해서는 사용자 함수 지원 및 여러 가지로 개선해야 할 사항들이 아직 많음은 분명하다. 하지만 적은 비용으로 투자대비 큰 효과를 낼 수 있는 장점도 존재한다. 그리고 하둡 기반 DW 시스템인 타조는 2014년 4월 아파치의 최상위 프로젝트로 승격되면서 그 기술의 성숙도가 높다는 점을 인정받았고 이로 인해 국내외 다양한 업체의 더 많은 개발자들이 개발에 참여하여 다양한 기능이 추가되고 안정성이 개선될 것으로 보인다. 타조를 비롯한 하둡 기술들은 이처럼 점차 기술의 완성도가 높아져 향후 빅데이터 분석에 중요한 축을 담당할 것이며 관계형 데이터베이스가 주도하는 DW 시장에서도 주목 받고 활용 될 수 있을 것으로 예상된다.

참고문헌

[1] Apache Hadoop [Internet]. http://hadoop. apache.org

[2] Shvachko, K., Hairong Kuang, Radia, S., Chansler, R., "The Hadoop Distributed File System" in Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, 2010. 3.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data procession on large clusters," in Usenix OSDI, vol. 51, no. 1. ACM, 2004.

[4] Mark A.Beyer and Roxane Edjlali, "Magic Quadrant for Data Warehouse Database Management Systems" Gartner, 2014.

[5] IBM Institute for Business Value, "Analytics: The real world use of big data" IBM, 2012.

[6] Apache Sqoop [Internet]. http://sqoop. apache.org

[7] Apache Tajo [Internet]. http://tajo.incu bator.apache.org

[8] Hyunsik Choi, Jihoon Son, Haemi Yang, Hyoseok Ryu, Byungnam Lim, Soohyung Kim, Yon Dohn Chung, "Tajo: A Distributed Data Warehouse System on Large Clusters" in IEEE ICDE Conference. 2013.

[9] Junghuk Park, Sangyeol Lee, Dahyun Kang, Joongho Won, "Hadoop and MapReduce", Journal of the Korean Data & Information Science Society. 2013.

[10] Sangwon Seo, Jaehong Kim, Yoonsung Park, "Hadoop & NoSQL". Gilbut. 2012

[11] A. Thusoo, J. Sarma, N.Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, “Hive - a petabyte scale data warehouse using Hadoop,” in Data Engine-ering(ICDE), 2010 IEEE 26thInternationalConferenceon,march2010,pp.996-1005.

[12] Takgil Sim, “Trend of SQL-on-hadoop tech-nology based on Open-source,” The Korea Society of Computer & Information, Vol. 21, No.1, 2013. 6.

[13] Jaehwa Jung, “SQL-on-Hadoop with Apache Tajo, and application case of SK Telecom”, in Deview, 2013.

[14] Googole BigQuery [Internet], http://developers.google.com /bigquery/

[15] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis “Dremel: Interactive Analysis of Web-Scale Datasets”, Proc. of the 36th Int'l Conf on Very Large Data Bases, pp330-339, 2010.

[16] Hyunsik Choi “SQL-on-Hadoop and Tajo”, Tech Planet, 2013.

[17] Transaction Processing Performance Coouncill [Internet]. http://www.tpc.org/tpch/

[18] Lee Hyunjong, "Use of Big Data Hadoop Platform" in Journal of Communications and Networks, Vol.29 No.11, 2012.

[19] Ankit Jain, "Instant Apache Sqoop", Packt Publishing Ltd, 2013.

[20] HooYoung Ahn, KyongHa Lee, SooHo Lee, YoonJoon Lee, SangMin Lee, YoungKyun Kim, "An Efficient Method for Enhancing the Storage

Efficiency in Hadoop DFS" in Journal of KISS : computing practices, Vol.19 No.3, 2013.

[21] Dae Soon Choi, Jeehong Kim, Young Ik Eom, "Analyses of Replica Placement Schemes in Distributed File Systems", in Journal of Computing Science and Engineering, Vol.39 No.1A, 2012.

[22] Nodar MOMTSELIDZE, Alex KUKSIN, "Hadoop Integrating with Oracle Data Warehouse and Data Mining" in Journal of Technical Science and Technologies, Vol.2 No.1, 2013.

[23] Ognjen V. Joldzic, Dijana R. Vukovic, "The Impact of Cluster Characteristics on HiveQL Query Optimization" in Telecommunications Forum (TELFOR), 2013 21st.

[24] Rinusha Irudeen, Sanjeeva Samaraweera, "Big data solution for Sri Lankan development: A case study from travel and tourism" in Advances in ICT for Emerging Regions, 2013 International Conference on.

[25] Tom White, "Hadoop: The Definitive Guide, Third Edition", O'Reilly/Yahoo Press, 2012.

[26] Kathleen Ting, Jarek Jarcec Cecho, "Apache Sqoop Cookbook", O'Reilly, 2013.