碩士學位論文

Hadoop 기반 데이터 웨어하우스 시스템을 위한 효율적인 데이터베이스 색인 기법

嘉泉大學校 大學院
모바일소프트웨어學科
모바일소프트웨어학 專攻
李　　廷　　彬

碩士學位論文

Hadoop 기반 데이터 웨어하우스 시스템을 위한 효율적인 데이터베이스 색인 기법

Efficient Database Indexing Technipue for Data Warehouse System Based on Hadoop

嘉泉大學校 大學院

모바일소프트웨어學科

모바일소프트웨어학 專攻

李　　廷　　彬

碩 士 學 位 論 文
指導敎授 朴石千

Hadoop 기반 데이터 웨어하우스 시스템을 위한 효율적인 데이터베이스 색인 기법

Efficient Database Indexing Technipue for Data Warehouse System Based on Hadoop

위 論文을 모바일소프트웨어학 碩士學位 論文으로 제출함.

2016年 月 日

嘉泉大學校 大學院
모바일소프트웨어 學科
모바일소프트웨어학 專攻
李 廷 彬

이 論文을 李廷彬의

工學碩士 學位論文으로 認准함

2016年 月 日

審査委員長 ㊞

審 査 委 員 ㊞

審 査 委 員 ㊞

요 약

기존 기업들은 원활한 의사 결정을 위하여 데이터 웨어하우스 시스템에서 기업 활동에서 생성되는 데이터를 이용하여 비즈니스 분석을 시도하였으나, 최근 빅데이터 시대가 도래 하면서 기존 데이터 웨어하우스 환경인 RDBMS로는 처리에 한계가 있다. 따라서 이를 Hadoop 기반으로 플랫폼 화하여 대체하였으며 탐색 성능 향상을 위해 색인 기법에 대한 연구가 진행되고 있다.

그러나 기존 색인 기법들은 Update가 불가하거나 디스크 기반에서 데이터를 처리하기 때문에 색인의 생성, 탐색 속도가 증가하는 문제가 있다. 따라서 본 논문에서는 위와 같은 문제점을 개선하기 위하여 색인의 Update가 가능하고 색인 데이터의 탐색 시간을 감소시키는 효율적인 DB 인덱싱 기법을 제안하였다.

본 논문에서 제안하는 색인 기법을 설계하기 위하여 Update를 가능하게 하기 위해 데이터 B+-tree를 이용하여 추가된 데이터에 대해서만 일정 크기로 색인을 생성하였다. 또한 데이터의 탐색 속도를 감소시키기 위해 메모리 기반 색인 B+-tree에 색인 데이터를 저장하는 색인 기법을 설계하였다.

본 논문에서 설계한 효율적인 DB 인덱싱 기법을 활용한 테스트 시스템을 구현하기 위하여 1개의 마스터 노드와 2개의 슬레이브 노드 환경에서 Jdk1.7, Java언어를 사용하였다. 또한 HDFS에서 데이터를 찾기 위한 색인 데이터를 생성하고 생성한 색인을 사용한 데이터로 HDFS에 적재된 데이터를 탐색하는 Hadoop 기반 색인 기법을 구현하였다.

마지막으로 제안하는 효율적인 DB 인덱싱 기법의 성능 평가는 TPC에서 제공하는 데이터 웨어하우스의 성능을 측정하는 TPC-H 벤치마크의 lineitem을 데이터를 사용하였다. 기존 시스템과 제안 시스템의 색인 생성과 탐색 시간을 평가한 결과 제안 알고리즘이 기존 알고리즘보다 색인 생성 시간은 60%, 데이터 탐색 시간은 10배 향상되었음을 확인하였다.

약 어 표

DB-Indexing	Double Btree-Indexing
CI	Create Index
DS	Data Search
HDFS	Hadoop Distributed File System
IDC	International Data Corporation
GFS	Google File System

목 차

표 목 차

그 림 목 차

I. 서 론

최근 수 년 간 소셜 미디어의 성장과 인터넷 서비스 등의 급격한 발전과 확산으로 대량의 데이터가 생성되고 있다. 이러한 데이터의 축적량이 기존의 데이터베이스 시스템으로 처리하기에는 어려울 만큼 크고 빠르게 축적되면서도 다양한 곳에서 유래하는 데이터를 빅 데이터라고 정의한다[1, 2]. 이러한 빅데이터를 축적하고 분석하려는 노력이 활발하게 이어지고 있는데 이는 축적된 다양한 데이터의 분석을 통해 기존에 알지 못했던 새로운 가치들을 도출할 수 있기 때문이다. 기존에 기업은 경쟁력 강화를 위해 보유하고 있는 정보나 서비스를 운영하면서 축적하여 온 로그 등의 데이터와 기업에서 제공하는 서비스나 소비자(혹은 사용자)들의 활동을 통하여 취합할 수 있는 데이터들을 결합하여 분석을 진행하고자 데이터 웨어하우스 환경을 구축하고, 이의 분석 결과를 이용하여 효율적인 의사 결정을 하고자 하였다.

하지만 기존 데이터 웨어하우스 환경인 관계형 DBMS는 소수 서버의 스케일-업(Scale-up)구조로 높은 비용과 확장성이 제한되어 최근의 빅데이터의 처리에는 한계가 있다.

Hadoop[12]은 스케일-아웃(Scale-out) 구조로 Google의 맵리듀스[8, 9]와 하둡 분산 파일 시스템인 HDFS[11]로 이루어져 있으며 대용량 데이터를 저렴한 비용으로 처리하는 프레임 워크로 널리 사용되고 있다. 데이터 웨어하우스 시스템의 핵심인 읽기 위주의 대규모 데이터에 대해 빠른 질의 응답 성능을 가진 Hadoop으로 기능을 대체하여 플랫폼 화하였고 탐색 성능 향상을 위해 색인을 이용한 연구들이 진행되고 있다.

Hadoop 기반의 색인 기법들은 성능 향상을 위해 색인에 데이터 파일의 각 키 별 데이터 레코드의 오프셋을 이용하여 탐색 시 불필요한 디스크 I/O

를 감소시키는 기법을 제안하였다. Batch mode creation of the Index using MapReduce에서는 데이터 파일들을 배치 적으로 맵리듀스를 통해 B+-tree[16] 구조에 색인을 생성하고 요청에 따라 각 리프 노드에서 오프셋을 구하여 해당 데이터를 탐색한다[14]. Split Index의 경우에는 RDBMS에서 데이터 파일을 관리하다가 일정 크기가 지나면 색인을 생성하고 데이터 파일을 Hadoop에 저장한다[14]. 이러한 방법으로 데이터 요청 시 탐색 시간을 최소화 하였다.

그러나 기존의 색인을 이용한 기법들은 색인 데이터의 효율성은 고려하지 못하였다. 주기적으로 데이터 파일 단위로 색인을 생성하여 기존에 생성하였던 데이터에 대해서도 다시 생성하여 생성 Update 시간이 지연되거나 디스크에 색인 파일을 저장함으로써 색인 탐색 시의 메모리보다 I/O 시간이 증가하여 탐색의 효율성이 떨어진다.

따라서 본 논문에서는 데이터 B+-tree를 이용하여 추가된 데이터에 대해서만 일정 크기로 색인을 생성하여 Update가 가능하게 하고 메모리 기반 B- Tree에 색인 데이터를 저장하여 색인 데이터 탐색 시간을 감소하는 방식의 기법을 제안하였다. 이를 통해 Hadoop 기반 데이터 웨어하우스 환경에서 정형 데이터의 효율적인 색인을 설계 및 구현하였다.

본 논문에서 제안하는 색인을 설계하기 위하여 일정 크기로 모은 사용자의 입력 데이터를 (Key, FileName, Offset, Length)형태의 색인 데이터를 만들고 HDFS에 적재하고, 이후 색인 데이터를 통해 사용자가 요청하면 HDFS에서 데이터를 탐색했다.

또한 테스트 분석 결과 Hadoop 기반에서 데이터 웨어하우스 시스템에 효율적인 데이터베이스 색인 기법을 통하여 색인의 생성과 데이터의 탐색 시

간이 기존 Hadoop 기반 색인 기법보다 빠른 데이터 처리 시간을 보장하는 것을 확인하였다.

본 논문은 모두 5장으로 구성하였다. 1장 서론에서는 본 논문의 배경 및 필요성, 2장에서는 데이터 웨어하우스, Hadoop, Hadoop 기반 색인, 3장에서는 2장에서 분석한 기술을 토대로 정형 데이터의 효율적인 색인 기법을 설계하였고, 4장에서는 설계한 색인 기법을 테스트하여 제안 기법에 대한 성능을 평가 및 검증하였으며, 마지막으로 5장 결론에서는 본 연구의 결론과 향후 연구방향을 제시하였다.

II. 관련연구

본 장에서는 Hadoop 기반 데이터 웨어하우스 시스템을 위한 효율적인 데이터베이스 색인 기법을 설계하기 위하여 빅데이터의 전반적인 개요와 데이터웨어하우스, 색인에 대한 특성을 알아본다. 또한 기존의 Hadoop 기반 색인 기법에 대하여 설명하고 본 논문에서 제시하는 효율적인 DB 인덱싱 기법을 구현하기 위한 기술인 Hadoop과 세부 기술인 맵리듀스와 HDFS에 대하여 알아본다.

2.1 빅데이터

2.1.1 정의

빅데이터[1]는 기존의 데이터 수집, 저장, 관리, 분석 역량을 넘어서는 대용량의 데어터 세트를 의미하며 기존 관계 데이터와 비교하여서 양, 속도, 다양성, 복잡성에서 차이가 있다. 데이터는 정형화된 데이터와 비정형화된 데이터가 있고 최근에 논의되고 있는 빅데이터의 경우에는 정형화된 것이든 비정형화된 것이든 관계없이 엄청난 양의 데이터를 말한다. 빅데이터에 정의는 다양하지만, 기업인 관점에서 빅데이터는 기업의 효과적인 전략 도출에 필요한 상세하고도 높은 빈도의 수로 생성되는 다양한 종류의 데이터로 정의한다. 다음은 기관에서 내린 빅데이터의 정의이다. 빅데이터에 대한 정의는 보는 과점에 따라서 다양하지만, 결론적으로는 빅데이터란 현재 시스템의 처리 범위를 넘는 데이터로써 기존의 방식과 다른 새로운 처리와 분석

방법을 필요로 한다는 것이다.

2.1.2 출연 배경

그림 2.1과 같이 전 세계적으로 몇 년간 생산, 유통되는 디지털 기반 정보량이 2011년을 기준으로 1.8 제타바이트, 즉 약 2조 기가 바이트에 달하고 있다. 이는 전 세계적으로 사람 한 명당 평균 280 기가바이트, 약 DVD 영와 45편의 (1편당 6기가바이트 기준)에 해당이 되는 엄청난 정보량이다. 빅데이터 현상은 기업이의 고객 데이터 수집 활동 및 멀티미디어 콘텐츠가 폭발적으로 증가 뿐만 아니라 스마트폰 보급으로 인하여 SNS의 활성 및 사물통신망의 확대로 빠르게 확산되고 있다. 데이터 처리는 기존에도 논의되어 왔었지만 최근 다시 주 관심사로 떠오르게 된것은 다양한 산업 분야에서 IT를 활용한 전자화나 자동화가 크게 발전됨에 따라서 처리 및 분석에 활용할 수 있는 대량의 데이터가 축척이 되고 있기 때문이다.

세계적인 컨설팅 기관인 Mckinsey and company에서는 'big data : next frontier forinnovation, comperirion and productivity'에서는 빅데이터 부상하게된 배경을 다음과 같이 정리하였다.

첫째, 기업의 고객 데이터 수집 및 트래킹 행위의 증가하였다. 고객 데이터들이 인터넷, 스마트폰 등 다양한 매체들을 통해서 데이터가 트래킹이 되며, 온라인을 넘어서 오프라인 상에서도 사용자 정보, 소비자 형태에 관련된 정보 수집이 가능해 지게 되었다. 영국의 유통업체인 테스코의 경우에는 매달 15억 건 이상의 고객 데이터를 수집하고 있다.

둘째, 저장매체와 카메라 모듈, 디스플레이 가격의 인하로 멀티미디어 콘

텐츠 사용 확산과 이에 관련한 정보의 증가를 가져왔다. 고화질 동영상의 경우에는 이미 전체 인터넷 트래픽의 50% 이상을 차지하고 있으며 2013년 기준, 70% 에 달하고 있다.

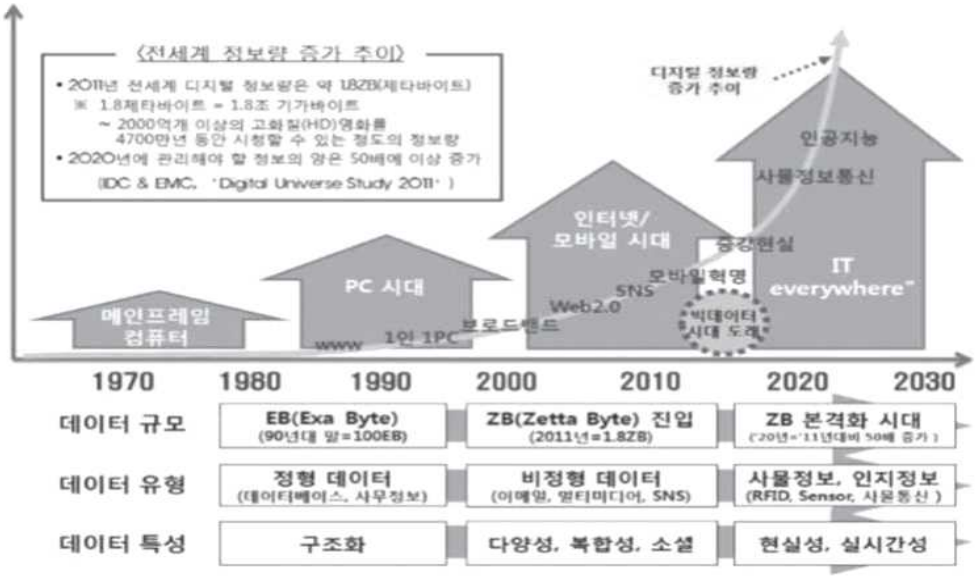

그림 2.1 빅데이터 증가 추이

셋째, 트위터, 페이스북 등 SNS의 급격한 확산으로 텍스트 등 비정형 데이터의 폭증이다. 페이스북 에서만 한 이용자당 매월 평균 90개 이상의 콘텐츠가 업로드가 되고 있으며, 유튜브의 경우에는 1분마다 24시간 분량의 비디오가 업로드 된다. SNS에서 이용되는 정보는 비정형 데이터로서, 이 f 를 처리하기 위해서는 추가적인 데이터 처리가 필요하며 데이터의 복잡성을 증가 시키게 된다.

넷째, M2M (Machin to machine), IoT (internet of things) 등의 통신 기술의 발전에 따른 사물 간 통신 네트워크에서 발생하는 데이터의 증가 이다. M2M, IoT 등 의 활성화되며 사용자가 데이터를 생성하지 않고 인프라가 다량의 데이터를 생성하게 되었다.

2.1.3 구성요소

빅데이터[2]의 구성요소는 일반적으로 3V(Volume, Velocity, Variety)를 기본으로 하고 1V(Value)나 1C(Complexity)가 추가되어 설명된다. IBM은 빅데이터를 3V 특징을 가지고 있는 새로운 타입의 데이터로 과거에는 찾을 수 없었던 인사이트(Insight)를 얻는 기회로 정의하였고 SAS의 경우에는 3V에 새로운 가치(Value)를 더한 4V를 빅데이터의 기본모습으로 제시하였다.

(1) 규모 (Volume)

데이터의 발생량으로서 페타바이트, 제타바이트 이상의 정보를 기준으로 물리적 크기뿐만이 아니라 데이터의 속성에도 연관되며 그것을 처리하는데 어려움이 있는지 혹은 없는지를 의미하게 된다.

(2) 속도 (Velocity)

데이터를 처리는 속도로서 배치 방식의 분석만을 의미하는 것이 아닌 필요에 따라서 수만은 사용자들의 요청을 실시간 처리한 후 처리 결과를 반환하는 기능에 대한 시간적 의미와도 연관된다. 다시 말해서, 데이터의 생성 및 처리가 진행이 되는 속도를 말하며, 그 의미는 세 가지로 구분이 된다. 데이터가 발생 후에 기업 내의 스토리지에 저장되기까지의 속도와 발생한 데이터의 필요하지 않은 부분과 의미 없는 부분을 처리하여서 사용하게 되는 수준까지의 속도, 그리고 정제가 된 데이터를 분석하고 의미를 추출하여서 최종의 목표를 달성 하는 속도까지를 의미한다. 이러한 속도는 데이터의 접근성 및 사용 가능성을 높이는 데에 많은 영향을 주게 된다.

(3) 다양성 (Variety)

기존의 전통적인 기업 분석은 데이터베이스에서 처리하는 구조적인 데이터로서 수치 및 텍스트가 대부분을 이루었다. 하지만 빅데이터 분석에는 통일된 구조로는 처리하기 어려운 비정형 데이터 분석이 전체 분석 중 90% 이상을 차지하며 사진 및 동영상 등의 구조화된 데이터가 아니라 다양한 형태의 데이터를 포함한다.

(4) 복잡성 (Complexity)

데이터의 복잡성은 데이터 구조, 규칙, 도메인, 저장 타입 등 데이터의 발생, 정제, 처리, 등의 과정에 포함된 모든 요소가 복잡해지는 것을 의미하게 된다. 예를 들어서 미디어 관리 시스템의 경우 영상 데이터는 다양한 인코딩 포맷을 갖고 있기 때문에 메타데이터를 기준으로 데이터를 분류한 후 파싱(parsing)하는 작업이 필요하다. 이와 같이 데이터 유형에 따라서 분석을 위한 전처리(preprocessing)가 필요하다. 이와 같이, 빅데이터는 기본적으로 대용량의 데이터를 갖고 있으며 실시간 빠른 속도로 데이터가 생성되는 실시간성을 포함하며 구조화된 데이터가 아니라 다양한 형태의 정보로 구성이 되기 때문에 기존의 데이터 보다 복잡하고 심화된 데이터 처리 및 관리가 필요한 새로운 패러다임이다.

2.1.4 빅데이터의 특성

앞에서 언급한 것처럼 비정형성, 다양성 및 복잡성을 가진 빅데이터에 대

해서 신속, 정확 비즈니스의 가치를 실현하기 위해서 분석하기 위한 IT기반 구조가 반드시 필요하다. 펄크럼 리서치에서는 현재 기업이 보고하고 있는 전체 정보 중에서 약 20% 정도만이 데이터베이스의 테이블이나 컬럼에서 정의 가능한 정형(Structured) 정보로 이루어져 있고, 나머지 80%의 정보의 경우에는 데이터베이스에 정의할 수 없는 비정형(Unstructured) 정보로 이루어져 있다고 한다. gartner는 2011년에 발간된 'Big Data Analytics'에서 기존 데이터 처리와 빅데이터 처리에 대해서 그 차이점을 설명한다.

(1) 상대적 의사결정 (Speed of decision making)

기존 데이터 처리와 다르게 신속한 의사결정은 상대적으로 덜 요구되어지며 대용량의 데이터에 기반을 두고 있는 분석 위주로 장기적, 전략적인 접근이 필요하다. 즉 달리 말하면 기존 데이터 처리에 요구되는 즉각적인 처리 속도와는 다르게 즉각 적인 의사결정이 상대적으로 덜 요구되게 된다.

(2) 처리 복잡성 (Processing complexity)

단순한 처리 모델이 아닌 다양한 데이터 소스, 복잡한 알고리즘 처리, 대용량 데이터 처리 등으로 인하여 처리의 복잡도가 매우 높으며 이것을 해결하기 위해 일반적으로 분산처리 기술이 필요하다.

(3) 방대한 처리 데이터양 (Transactional data volumes)

클릭 스트림데이터 (click stream data)의 경우는, 고객 정보수집 및 분석을 장기간에 걸쳐 수행이 되어야하므로 기존 방법과 비교해서 처리해야 할

데이터양이 방대하다. 비정형 데이터 구조(Data Structure)인 소셜 미디어 데이터, 클릭스트림 데이터, 로그 파일 및 콜 센터 로그 등의 비정형 데이터 파일의 비중이 높다. 처리 복잡성을 증대시키는 요인이기도 한다.

(4) 처리 및 분석의 유연성 (Flexibility of processing/Analytics)

주어진 데이터 모델이나 파악된 데이터의 상관 관계 및 주어진 절차 등에 상관없이 기존 데이터 처리 방법과 비교하게 되면 처리 및 분석의 유연성이 높다. 새롭고 다양한 처리 방법의 수용을 위해서 유연성이 기본적으로 보장되어야 한다.

(5) 통시처리량 (Throughput)

대용량 및 복잡한 처리의 특징이 있어서 동시에 처리가 필요한 데이터의 양은 낮은 편으로 실시간 처리가 보장되어야 하는 데이터 분석에는 적합하지 않다.

2.2 데이터 웨어하우스

2.2.1 개요

기업은 분석하고자 하는 대단위의 데이터가 분산 저장되어 있고, 이들 각각도 대부분 대량의 데이터로 이루어진다. 예를 들어 신용 카드 회사는 점포와 관련된 데이터와 점포에서 결제된 내용이 담긴 데이터가 다른 스키마로 존재하게 되거나 각각 다른 시스템에서 운영 중일 수 있다. 의사 결정자

는 여러 종류의 데이터에 모두 접근하여 질의해야 하지만, 데이터에 대하여 각각 질의를 수행하는 것은 비효율적이며 상황에 따라서 다른 추가적인 정보의 필요성이 있다. 데이터 웨어하우스 시스템은 이런 상황에서 여러 저장소 혹은 근원지로부터 모은 데이터나 정보를 일관된 스키마로 한 곳에 모아 놓은 저장소를 의미하며 이후 데이터를 다차원 분석을 하여 의사결정에 도움을 주는 시스템이다[3]. 기존의 데이터 웨어하우스 시스템은 다른 장소의 관계형 데이터베이스로부터 가져오는 축적된 데이터의 규모가 최고 테라바이트(TB) 정도였기 때문에 워크스테이션 규모의 장치에 대용량 저장 매체를 필요한 만큼 장착하여서 사용하였다[4]. 그리고 각 데이터의 다양한 스키마는 일정한 형식을 가진 정형 데이터로 관계형 데이터베이스로 처리하였다. 그러나 최근 데이터 생산량의 폭발적인 증가로 인하여 빅 데이터의 중요성이 인식되면서 수요자들에 의해서 다루어지는 빅 데이터는 기존의 데이터와 비교하였을 때 양과 형식, 근원지가 크게 달라졌다. SNS 회사인 페이스 북(Facebook)의 경우에는 2010년 당시 축적해 온 데이터의 규모가 750TB에 이를 만큼, 사용자가 생산하는 데이터의 양이 이미 테라바이트(TB) 규모를 넘어서 페타바이트(PB)에 이르고 있다[4, 6]. 과거에는 데이터가 생산되는 근원지가 설문 조사 또는 현실상의 매장이나 유선 전화 등에 한정되어 있었다. 그러나 모바일과 임베디드 기기의 발달로 인하여 근원지가 사람들이 자주 사용하는 인터넷, 모바일 디바이스의 애플리케이션, 실시간 결제 시스템, SNS와 같은 다양한 기기나 서비스로 확장되었다. 이와 같이 다양한 근원지로부터 생성되는 데이터들은 스키마와 내용이 모두 달라서, 이를 그대로 관계형 데이터베이스에 저장하기에는 별도의 처리가 필요하게 되거나 모든 데이터를 포괄하는 스키마를 필요로 한다. 따라서 이를

저장하고 활용하기 위해서 비관계형 데이터베이스(NoSQL Database)[7]가 제시되었다. 비관계형 데이터베이스의 종류로는 컬럼 모델, 키-값 모델, 그래프 모델, 문서 모델 등이 있으며, 현재 다양하게 활용 되고 있다[5]. 이와 같이 빅 데이터의 다양한 특징들 때문에 기존의 데이터 웨어하우스 시스템에서는 데이터를 그대로 활용하기에는 어려움이 있으므로 빅 데이터 분석 프레임워크 및 관련 소프트웨어가 요구되게 된다.

2.2.2 데이터 웨어하우스 구성요소

지식 집합체로서의 데이터 웨어하우스[3]는 효율적인 정보의 통합, 가공 처리를 위해서 데이터를 추출하고 정재, 적재하기 위한 ETL 기술과 ODS (Operational Data Store), 데이터마트(DM, DataMart)등의 핵심적인 구성요소와 이것들을 보다 더 효율적으로 관리하기 위한 데이터 통합(Data Integration), 데이터 전환(Data Migration) 및 마스터 데이터 관리(MDM, Master Data Management)기술 등의 보조적인 구성요소들로 이루어진다.

데이터의 통합 관리를 위하여 데이터 웨어하우스 시스템을 구축하며 효과적인 활용과 관리를 위한 체계를 수립하고 이것들을 운영하는 일은 정보기반의 의사결정을 효율화하는데 중요한 역할을 하고 있다. 구성 요소별로 속성을 정의하면 첫 번째는 이기종(Oracle, MYSQL, DB2, MSSQL 등) 데이터베이스가 연계되어 운영이 되고 있는 운영계 시스템(대부분은 일괄적 관점이 아니라 순차 혹은 일부 병행의 관점에서 도입, 구축함)의 지속적인 데이터 증가와 테이블, 컬럼 등의 객체 속성 변화는 지속적인 동기화 및 변환, 검증 등의 요구사항을 지원하는 ETL의 기술이 필요된다. 이와 더불어

서 기업 정보 시스템을 연계하고 통합하는 EAI(Enterprise Application Integration)프레임워크와 데이터베이스 내에서 변경이 되는 데이터를 식별하여서 후속적인 처리를 자동화하는 기술인 CDC(Change Data Capture) 등의 다양한 기술들과 연계해 더욱 강력한 기능을 제공한다. 두 번째 ODS는 운영시스템(Online Transaction Processing)의 경우는 데이터베이스를 정의한 시간에서 객체와 데이터의 변경된 사항을 동기화하는 저장소로서 데이터 웨어하우스 및 데이터 마트에 가공된 데이터를 적재하기 위한 임시 저장소이다. ODS는 지속적인 데이터 동기화를 실행함으로 데이터 웨어하우스, 데이터 마트의 새로운 분석 요구로부터 운영 시스템의 중복 및 부하를 발생 시키는 접근을 최소화 시킨단. 또한 데이터 무결성 및 일괄 업데이트와 정재 작업도 지원하며 목적에 따라 기능이 변경이 되거나 생략되는 경우도 있다.

세 번째로 데이터 마트는 최종 사용자 관점으로 가공이 된 비즈니스 영역 데이터이거나 이를 다차원으로 분석하기 위해 요약, 저장하는 장소로서 활용이 된다. 최종 사용자 관점은 기업의 마케팅, 영업, 인사, 재무 등의 업무 영역별 혹은 부서별 속성을 의미하고 발생한 데이터 객체의 통합 및 가공을 통해서 상당한 부분의 요약 및 집계된 데이터를 포함하게 되다. 그 외에도 데이터 통합, 데이터 전환 및 MDM 기술의 경우는 운영 데이터베이스 시스템과 데이터웨어하우스 시스템을 보다 효율적으로 운영, 개선하기 위한 방법론적인 도구로서 사용된다. 이상적인 데이터 웨어하우스[4] 구조는 BI(경영정보학, Business Intelligence)라는 큰 틀에서 의사 결정 지원을 위한 대화형 정보 분석을 지원하며 다차원의 정보의 제공이 목적인 온라인 분석처리(OLAP, Online Analytical Processing), 대규모 저장 데이터 내의 자

동적이고 체계적인 통계적인 규칙이나 패턴을 찾아내는 데이터마이닝(DM, Data Mining)등과 함께 통합 관리되어야 한다[3, 4].

2.3 Hadoop

구글에서 대량의 데이터를 분석하는 것에 사용할 수 있는 맵리듀스 프레임워크[8, 9]를 제안하였고, 이미 대량의 데이터를 분석하기 위해 구글 파일 시스템(GFS)[10]과 함께 사용 중이라고 밝혔다. 그리고 이를 복제하여 구글의 분석 시스템과 동일하게 구성하도록 맵리듀스 프레임워크와 HDFS(하둡 분산 파일 시스템)[11]를 함께 구현한 오픈소스 분석 소프트웨어가 아파치 하둡(Apache Hadoop, 이하 하둡)[12]이다. 현재 하둡은 HDFS와 맵리듀스 프레임워크, 클러스터 자원 관리 프레임워크인 YARN(얀)[13]으로 이루어져 있다.

HDFS 파일 시스템은 순수 자바 파일 시스템으로 여러 노드에 대용량의 파일을 저장할 수 있다(이상적 파일 크기는 64MB). HDFS는 단일 호스트에서 RAID 스토리지를 걸치지 않고서 안정성을 제공하며, 연결된 수많은 노드에 데이터를 복사하는데, 기본 복사본의 숫자는 세 개이다. 이는 HDFS 파일 시스템이 세 개의 노드(2개는 같은 랙, 1개는 다른 랙)에 데이터를 저장한다. 파일 시스템은 데이터 노드들의 클러스터를 통해 구축이 되며, 각각의 서버들은 데이터 블럭을 네트워크를 통해서 전달한다. 또한, HDFS는 웹 브라우저나 다른 클라이언트를 통해 모든 컨텐츠에 대해 접근할 수 있도록 해서 HTTP 프로토콜을 통헤서 데이터를 제공하기도 한다. 데이터 노드들은 데이터의 재 분배 작업(복사본을 옮기거나 데이터의 복제를 위해)을 위

해서 서로 통신한다.

HDFS는 마스터, 슬레이브(Master, Slave)구조를 가진다. HDFS 클러스터는 하나의 네임 노드가 파일 시스템을 관리하며 클라이언트의 접근을 통제는 마스터 서버로 구성된다. 또한 클러스터의 각 노드에는 데이터 노드가 한 개씩 존재하며 이 데이터 노드는 실행될 때마다 노드에 추가되는 스토리지를 관리한다. HDFS는 네임스페이스를 공개해 유저 데이어가 파일에 저장되는 것을 허락해 준다. 내부적으로 하나의 파일은 하나 이상의 하나의 파일은 하나 이상의 블록으로 나누어지고, 이 블록들은 데이터노드들에 저장된다. 네임 노드는 파일과 디렉터리의 읽기(open), 닫기(close), 이름 변경(rename) 등 파일 시스템의 네임스페이스 등의 여러 기능을 수행한다. 또한, 데이터 노드와 블록들의 맵핑도 결정한다. 데이터 노드는 파일시스템의 클라이언트가 요구하는 읽기(read), 쓰기(write)기능들을 담당한다.

또한 데이터 노드는 네임노드에서의 생성, 복제, 삭제 등의 기능도 수행한다. 네임 노드와 데이터 노드는 GNU, Linux OS를 기반의 상용 머신에서 실행하기 위해 설계된 소프트웨어의 일부이다. HDFS는 Java 언어를 사용하기 때문데 Java가 동작하는 어떠한 컴퓨터에서나 네임 노드나 데이터 노드 소프트웨어를 실행할 수 있다.

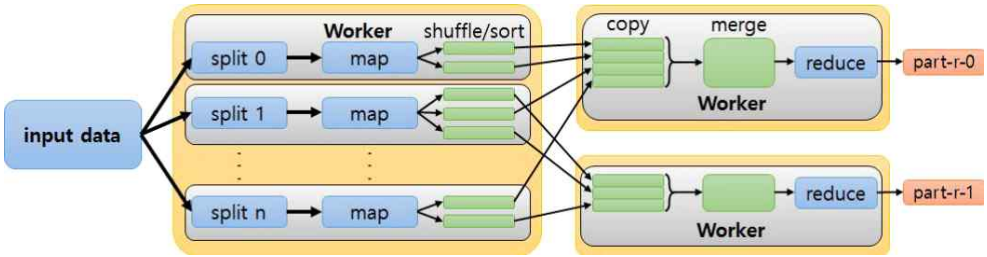

그림 2.2 맵리듀스 구조

그림 2.2는 하둡의 맵리듀스 프레임워크를 기반으로 하여 작성된 맵리듀스 애플리케이션의 작업 수행 과정을 표현한 것이다. 맵리듀스 애플리케이션의 작업 수행은 크게 맵(Map)과 셔플 및 정렬(Shuffle/Sort), 리듀스(Reduce)의 3단계 과정으로 이루어진다. 맵은 함수로 작성된 동작에 따라 입력받은 데이터를 정제하고 변형하는 역할을 수행. 정렬 및 셔플은 리듀스 과정을 수행하는 각 워커(Worker)에 맵으로부터 전달받은 데이터를 용이하게 전달하기 위하여 데이터를 섞거나 정렬한다. 리듀스(Reduce)는 함수로 작성된 동작에 따라서 전달받은 데이터를 하나로 결합하고 파일로 작성하는 동작을 수행하게 된다. 맵리듀스 애플리케이션의 보다 세부 적인 작업의 수행 내용은 다음과 같다. 다수의 서버에서 각각 실행 중인 워커가 데이터를 HDFS로부터 읽어서 맵에 전달하면, 맵에 작성된 일련의 동작으로 각각의 데이터를 처리한다. 그리고 처리된 데이터를 셔플 및 정렬을 거친 후에 각 데이터를 다시 처리하기에 적합한 워커로 보낸다. 마지막으로 맵을 통하여 처리되어 전달받은 데이터들을 하나로 결합하는 리듀스를 수행한 후 각 결과를 모아 결과물을 HDFS에 저장한다. 이 과정에서 YARN은 각 서버의 워커에 필요한 자원의 할당과 해제를 담당하며, 작업에 대한 내역(진행 상황, 로그 등)을 마스터 서버로 전달한다. 또한 받은 정보를 바탕으로 할당된 작업을 수행할 수 있는 워커를 판단하여 명령을 보내는 보조적인 역할을 담당한다. 맵리듀스는 근본적으로 맵, 셔플 및 정렬, 리듀스의 각 과정에 해당하는 일들을 하나의 작업으로 묶어서 처리한다. 따라서 더 복잡하거나 다양한 처리를 수행한다면 기존의 알고리즘을 맵리듀스 프레임워크의 수행 과정에 맞게 일련의 일괄 처리의 형태로 계획하여 실행하거나 혹은 전처리 작업을 거친 후의 데이터를 이용하여 맵리듀스 작업을 수행해야만 하는 단점이 있다.

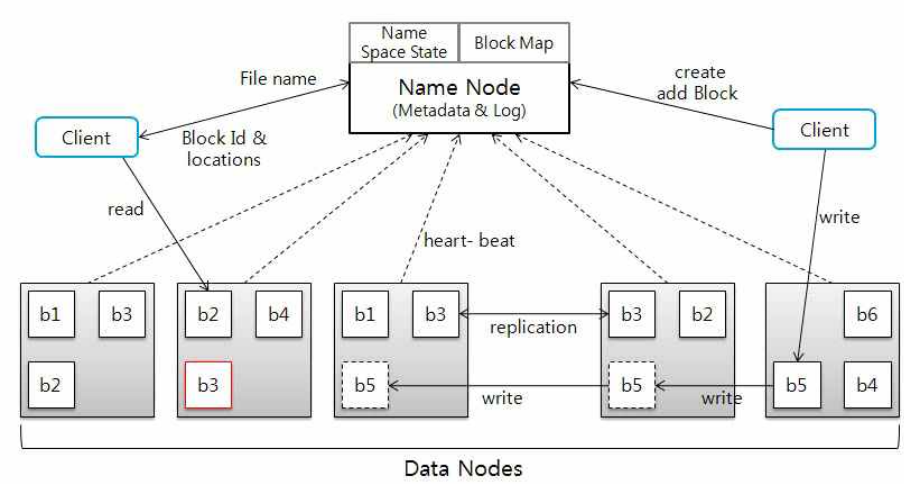

그림 2.3 HDFS 구조

그림 2.3은 HDFS[11] 안에서 네임 노드와 데이터 노드의 전체적인 통신 구조를 나타낸다. 데이터 노드는 클라이언트로부터 출력된 데이터 블록을 클러스터 내의 다른 데이터 노드에 복사시키고, 자신의 상태를 일정한 주기의 하트 비트로 네임 노드에 전송하며, 네임 노드는 전송된 데이터를 기반으로 노드를 관리하고 클라이언트 요청에 응답한다. 클라이언트는 데이터를 읽을 경우 네임 노드에서는 먼저 접속하여 데이터 블록의 Id와 위치를 알아내고 데이터 노드에 직접 접근한다[18].

2.4 색인 기법

2.4.1 데이터베이스 색인

데이터베이스 색인(Indexing)이란 캐쉬 기법의 일종으로써 데이터베이스의

효율적인 접근을 수행하는데 사용되는 색인을 만드는 작업이다. 색인이란 원래 정보를 미리 생성해서 원하는 데이터를 빨르게 찾을 수 있게 해주는 기법으로써, 데이터베이스 색인은 데이터베이스 안의 테이블에 원하는 정보를 좀 더 빠르게 탐색해 줄 수 있게 원하는 데이터의 위치 정보를 모아 놓은 객체(Object)이다. 데이터베이스는 시스템 내부에서 하나 이상의 저장 파일로 구성되고 파일은 같은 형식의 레코드들로 구성된다. 그리고 이런 레코드들은 일정한 규칙에 따라서 색인된다. 데이터베이스에서 색인에 의한 데이터베이스 레코드 접근은 먼저 레코드에 해당하는 색인을 찾은 후 색인 안에 저장되어 있는 해당 레코드의 물리적 주소를 이용해 사전 검색 없이 한 번에 데이터에 접근 가능하도록 한다. 헌 색인 작업은 한 개의 레코드당 최초에 한 번씩 수행이 되며 테이블의 데이터가 Insert, Update등의 쿼리로 인해 변경이 될 경우에 다시 색인을 하게 되는 특징을 가지고 있다. 한 개의 테이블에 대해서 레코드별로 색인을 수행할 경우 Query를 수행 시에 빠른 응답을 할 수 있다는 장점도 있지만 데이터의 양이 많아지면 성능이 느려지거나 많은 메모리를 사용한다는 단점도 있다[18].

2.4.2 C-ISAM(Indexed Sequential Access)File

C-ISAM은 C 언어로 만들어진 색인 순차 파일이다. 많은 양의 정보에서 특정 레코드를 빠르게 탐색하기 위해 두 개의 파일로 구성한다. 한 파일은 데이터를 저장하고 다른 파일은 색인을 저장한다. 이 두 개의 파일은 항상 같이 사용되며 논리적으로는 하나의 C-ISAM 파일을 구성한다.

C-ISAM은 순차처리(sequential access)어 직접 처리(directa ccess)가

모두가 가능하다. 그리고 또한 파일 안의 트랙에는 레코드들이 정렬이되 있어야 한다. 데이터를 기록하는 부분 이외에 데이터 레코드 중에서 키 항목만을 모아서 색인 표를 생성하고 이를 이용해 순차 처리와 직접 처리 모두를 가능하게 한다. 색인은 특정 레코드에 대한 빠른 접근을 처리하는데 이용한다.

2.4.2.1 색인구성방법

색인의 순차 파일에서 색인과 순차 자료 파일을 구성하는 방법으로는 기본적으로 정적 색인(static index)방법과 동적 색인(dynamic index)방법으로 나뉘고 두 방법의 구분은 새로운 레코드를 삽입하거나 기존의 레코드를 삭제를 할 경우 순차 자료 파일에서 레코드의 순서를 유지하고 색인을 갱신하는 방법에 그 차이가 있다.

가) 정적색인(static index) 방법

데이터 파일에 레코드가 삽입이 되거나 삭제가 됨에 따라서 색인의 내용은 변경되지만 파일 구조 자체는 변하지 않는다. 데이터 파일에서 새로 삽입될 레코드가 저장될 공간이 없으면 오버플로 구역(overflow area)을 사용 한다. 색인은 보조 기억 장치의 실린더 및 트랙 등과 같은 물리적 특성에 맞게끔 하드웨어와 긴밀한 관계를 유지하며 설계한다.

나) 동적색인(dynamic index) 방법

색인이나 데이터 파일은 블록으로 구성이 되고, 각 블록에는 나중에 레코드가 삽입이 될 것을 감안하여서 빈 공간을 미리 준비해 두는 방법이다. 색인은 트리 형식으로 구성하고 하드웨어와는 관계기 없이 독립적으로 설계될 수 있기 때문에 최근에 들어 많이 사용 된다. 두 가지 방법에서는 모두 색인, 자료 부분을 별도의 파일로 분리하고 구성하는 이유는 이 두 요소의 내부 구조가 서로 다르고, 한 개의 파일에는 같은 종류의 데이터를 한 가지 방법으로만 구성해야하기 때문이다.

2.4.3 인덱스

인덱스는 일반적으로 키(Key)라는 개념이기 보다는 옵티마이자(Optimizer) 최적의 처리 경로를 결정하기 위해 사용하는 도구이다. 일반적인 통계에서 보면 애플리케이션 프로그램의 수정 없이도 적절한 인덱스의 지정만으로도 60% 이상의 접근 효율성을 향상 시킬 수 있다.

인덱스란 특정 테이블 안에서 컬럼의 값을 가지고 대응이 되는 레코드의 논리적인 주소를 연관시켜 주는 별도의 저장 구조로서 성능의 향상에 대한 요구를 충족시키기 위해 테이블 안에 추가를 시킬 수 있다. 인덱스는 다른 액세스 방법에 비해서 생성 및 해체, 재구성이 손쉽고 융통성이 있는 접근 기법일 뿐만 아니라 적용의 용이성으로 근래에 가장 많이 활용하고 있는 튜닝 기법이다. 그러나 인덱스를 유지 관리하기 위해서는 자원이 필요하다. 다시 말해서 인덱스를 가지고 있기 위한 디스크 공간이 필요하다. 모든 인덱스는 인덱스 데이터의 값만을 가지고 있는 것이 아니라 인덱스의 값과 각 인덱스들이 연결되는 데이터 구조 또는 인식 번호(Identification Number)

를 포함하고 있다. 결론적으로 인덱스는 적절히 사용해야 불필요한 자원의 낭비를 최소화 한다. 관계형 데이터베이스에서 사용하는 인덱스의 종류는 크게 3 가지로 분류 할 수 있다.

가. 주 식별자 인덱스(PrimaryKey Index) : 각 테이블의 한 경우가 그 테이블의 모든 경우와 구분 할 수 있는 유일한 것이며 절대로 NULL값을 가질 수 없다.

나. 부 식별자 인덱스(AlternateKey Index) : 데이터 질의 시에 성능 향상을 위해 지정하는 인덱스로써 조건 절(IF, WHERE)에 자주 나타나고 순차적 정렬(Sort)을 자주 수행하는 속성들이 대상이다.

다. 외부 식별자 인덱스(ForeignKey Index) : 이는 두 테이블 간에 관계를 결정해주는 키로서 주종 관계의 테이블에서 종속 실체 쪽에 위치하는 주 실체의 주 식별자와 동일한 인덱스를 말한다.

색인은 데이터를 찾을 때 빠르게 탐색하기 위해서 사용 한다. 색인이 없는 경우 (SELECT)때 특정한 값을 탐색하기 위해 모든 데이터 페이지를 전부 찾아야 한다. 이것을 table scan이라고 한다. 색인이 찾고자 하는 컬럼이나 표현식이 존재하고, 색인은 사용하는 것이 더 효율적이라면 SQL서버는 모든 페이지를 찾지 않고 색인 페이지를 찾아서 빠르게 데이터를 가져온다. 이것을 인덱스 SCAN이라고 한다. 색인이 있으면 특정한 작업, 데이터를 정렬하거나, 그룹 단위로 계산 함수를 사용할 때에도 색인은 많은 도움이 된

다. 뿐만 아니라 색인을 사용하면 유일성 검사가 가능하다. 즉, 키 값 이 같은 행이 두 번 입력되지 않도록 하기 위해서도 색인을 사용한다. 이러한 장점에도 불구하고, 색인으로 인한 손해는 있다. 색인 생성 많은 시간과 공간이 필요하고, 만들고 난 후에도 추가적인 공간을 필요로 한다. 또한, 인덱스는 SELECT시는 많은 성능 향상을 주지만, INSERT, UPDATE, DELETE시 에는 오히려 성능이 저하된다. 그 중 INSERT 작업의 경우는 오히려 더 많이 걸린다.

어떠한 컬럼에 대해서 색인을 생성하는가에 대한 기준은 거의 검색되지 않는 컬럼, 전체 중 상당수의 부분을 가져오는 질의에 사용되는 컬럼, 유일성, 혹은 같은 값이 많은 컬럼들, 그리고 SELECT 속도 보다 데이터의 변경 속도가 중요할 경우 되도록 색인의 수를 최소화 한다.

생성된 인덱스는 경우에 따라서 하나 이상 사용하기도 하고 사용하지 않을 수도 있다. 옵티마이자(Optimizer)에 의해 판단되는 접근 경로는 주어진 조건, 인덱스의 구성, 옵티마이자, 클러스터링, 통계정보 모드 등에 따라 다양하게 나타나며 결정된 액세스 경로에 따라서 수행 속도가 달라진다. 그러므로 양호한 접근 경로를 보장 받기 위해서는 옵티마이자의 액세스 생성 원리에 대한 이해를 바탕으로 적절한 접근 경로가 생성될 수 있도록 테이블을 설계하고, 종합적이고 적절한 경우를 대비한 인덱스의 지정, 넓은 범위의 처리나 조인의 효율성을 향상시키기 위한 클러스터링, 인덱스의 적용 원칙에 알맞는 SQL코딩, 효율적인 SQL구성 등을 활용하여 옵티마이자를 제어하는 등의 조치가 필요하다. 관계형 데이터베이스로 작성된 애플리케이션은 과거와 같이 알고리즘을 수정하여 수행속도를 향상시키는 경우보다는 주로 옵티마이자의 접근 경로 판단 기준인 인덱스, 클러스터 등의 조정을 통해서 이

루어진다. 이것은 옵티마이자가 최적의 접근 경로를 판단 할 수 있게 적절한 인덱스를 구성하지 않고서는 아무리 애플리케이션의 로직을 최적으로 구성하더라도 이미 수행속도의 향상을 기대할 수 없다는 것이다. 또한 다른 의미로는 인덱스 등의 적절한 지정만으로 애플리케이션의 수정 없이도 많은 속도 향상을 기대할 수 있다는 것이다. 인덱스를 설계하는 것은 해당 테이블의 접근 형태를 수집하는 것에서부터 시작한다. 인덱스는 반복적으로 수행되는 접근 형태를 찾고 분포도가 양호한 컬럼들을 발췌하여 액세스 유형을 조사해서 입력한다. 그리고 자주 넓은 범위의 조건이 부여되는 경우를 찾아서 조건에서 자주 사용되는 주요 컬럼들을 추출해 접근 유형을 조합하며 자주 결합되어 사용되는 컬럼들의 조합 형태 및 정렬 순서를 조사하여 입력한다. 또한 역순으로 정렬해서 추출하는 경우와 일련 번호를 부여하는 경우에 그리고 통계자료 추출을 위한 액세스 유형을 조사하여 입력한다.

2.4.3.1 인덱스 구조 (Index Structure)

인덱스의 구조는 크게 B-Tree, HashTable, ISAM 구조로 구성되고, 이들은 모두트리 구조를 사용한다. 인덱스 노드의 일부들은 메모리에 상주하여 작업을 수행하게 되는데 일반적으로는 루트 노드(RootNode)가 메모리에 있게 된다. 데이터의 탐색 요구에 대해 정보시스템의 성능은 루트(Root)에서 리프노드(LeafNode)의 평균 깊이(Depth)에 의존하며 인덱스 구조에서 깊이 차이를 적게하는 것이 관계형 데이터베이스 관리 시스템에서의 기술력이다. 데이터 구조에서 입력될 때 노드가 더 이상 저장할 공간이 없을 경우 이를 오버플로우(Overflow)라 하며 이를 처리하는 알고리즘(Algorithm)이 인덱

스 성능을 결정하므로 중요하게 인식되어야 한다.

(1) B-Tree구조

B-Tree구조는 거의 대부분의 관계형 데이터베이스 관리시스템에서 사용하는 인덱스 구조로 균형이 잡힌 트리(BalancedTree)이며 각 노드들이 키 포인터(Key-Pointer)쌍의 순차적 구조를 가진게 된다. 키는 키 값으로 정렬되어 키의 재 정렬이 수월하고 여러 종류의 질의 형태에 최적화 된다. 그리고 B-tree구조는 특히 RangeQuery에 유리힌 강점을 가진다.

(2) ISAM구조

B-Tree구조와 유사하지만 이따금씩 오버플로우가 발생되면 체인 데이터(ChainData)를 발생시킨다. ISAM 구조도 B-Tree와 마찬가지로 RangeQuery에 유리 하다. 입력이 많이 발생하는 데이터 구조에서는 병목현상이 발생할 수 있다. ISAM구조는 고정적이며 각 노드들에는 오버플로우가 발생하게 되고 체인으로 연결된다. 따라서 자주 구조를 재구축 해 주어야 한다.

(3) Hash Table구조

해쉬 기능(Hash function)이라는 특별한 방법으로 키 값을 생성하여 저장한다. 주어진 키를 디스크 상에 주어진 주소 또는 디렉토리 (Address

Directory)위치 번지의 값으로 변환, 저장하며 PointQuery에 적절하다[19].

2.4.4 Hadoop 기반 색인 기법

Mingyuan An은 네트워크 보안 모니터 어플리케이션에 사용되는 하둡 기반 색인을 제안하였다. 그림 2.4는 B+-트리와 선택된 리프 데이터 노드에서 맵 작업을 나타낸다. 소스 데이터부터 배치 적으로 적재되고, 한번 저장되면 update 작업을 할 수 없다. 후에 지속 배치 방식으로 색인을 생성한다. B+-tree를 생성할 때 먼저 (속성 값, 오프 셋)쌍으로 정렬한 모든 레코드들을 색인 파일에 출력하고 색인 트리의 리프 노드에 구성한다. 리프 노드를 찾을 때 루트까지 상향식 방법으로 색인 파일을 합친다. 전통적인 B+-tree와 달리 연속적인 범위의 색인 파일 공간을 보장하여 쿼리의 병렬적인 접근을 보장한다[14].

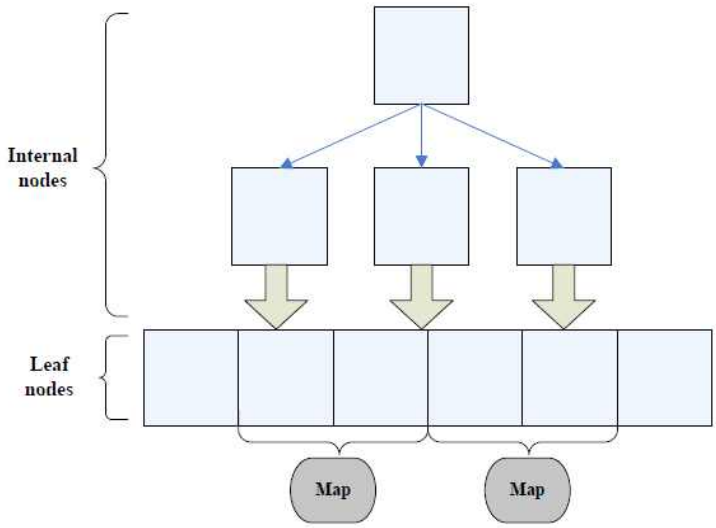

그림2.4 B+-트리 기반 색인

Vinitha Reddy Gankidi은 기존 RDBMS와 HDFS의 동시 사용 시스템에서 HDFS의 데이터 탐색을 빠르게 처리하기 위해 Polybase split index를 제안하였다. 그림 2.5는 Polybase의 구조를 나타낸다. Split Index는 RDBMS인 SQL Server에 색인인 B+-트리에 외부 테이블로 형태로 HDFS의 색인이 저장된다. 색인의 구조는 key, filename, offset, length, blockNumber로 구성되어있다. HDFS의 색인의 생성은 점진적으로 재생성하며 색인이 다시 만들어지기 전의 데이터를 위해 하이브리드 탐색을 한다. 하이브리드 탐색은 RDBMS의 색인을 통해서 찾고 남은 데이터들은 맵리듀스 잡을 이용하여 찾는다[15].

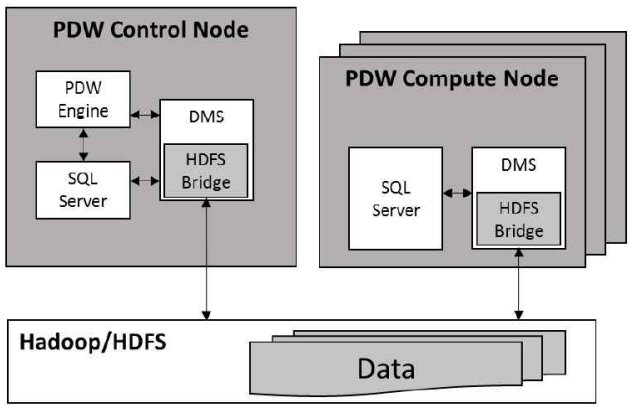

그림 2.5 Polybase 구조

2.5 효율적인 DB 인덱싱 기법 제안

기존 데이터 웨어하우스는 시스템을 Hadoop으로 기능을 대체하여 플랫폼화하고 색인을 이용하여 탐색 성능 향상 시키는 연구[14, 15, 20]가 진행되고 있다. 하지만 기존 Hadoop 기반 색인 기법들은 색인 데이터의 효율성의 대한 연구가 부족하고 문제점을 다음과 같이 지적한다.

(1) Update를 고려하지 않은 색인 : 그림 2.6에서 (a)와 같이 전체 파일 단위의 단일 색인을 생성하게 되면 이후 지속적으로 추가되는 데이터 파일에 대한 색인을 생성하기 위해서 기존의 색인이 되었던 데이터에 대해서도 색인을 재생성하여 색인 생성 시간이 증가한다[14].

(2) 색인이 디스크에 저장 : 그림 2.6에서 (b)와 같이 색인이 디스크에 저장되게 되면 색인 탐색 시에 메모리에 저장하는 것보다 상대적으로 I/O가 증가하고 파일을 전체 탐색하기 때문에 탐색 시간이 증가하여 데이터 탐색 효율이 저하된다[15, 20].

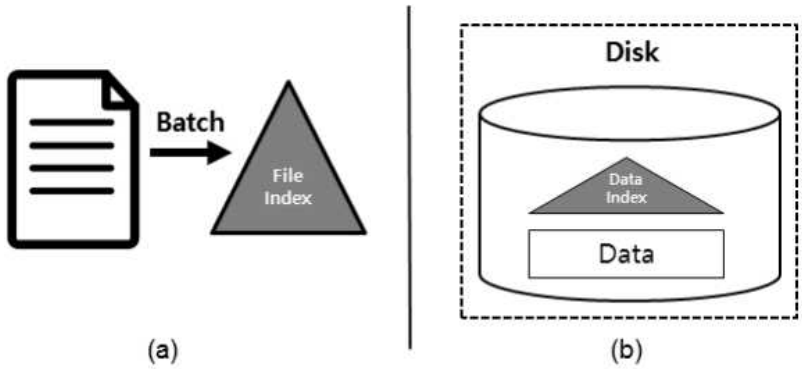

그림 2.6 기존 Hadoop 기반 색인 기법

따라서 본 논문에서는 그림 2.7과 같이 Update가 가능하게 하고 메모리 기반에 색인을 관리하여 색인 데이터 탐색 시간이 감소하는 효율적인 DB 인덱싱 기법을 제안하였다.

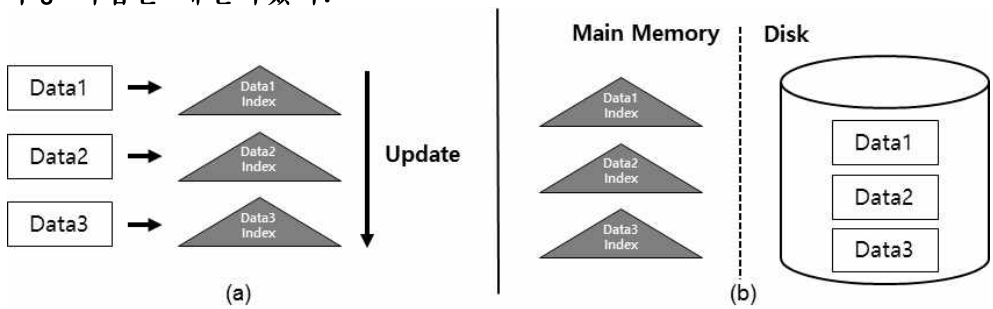

그림 2.7 제안하는 효율적인 DB 인덱싱 기법

III. 효율적인 DB 인덱싱 기법 설계

본 장에서는 Hadoop 기반 데이터웨어하우스 시스템에서 데이터를 색인하기 위한 효율적인 데이터베이스 색인 기법을 설계하였다. 설계를 통하여 기능적인 색인 기법의 구조를 파악하고 색인 기법의 단계별 적용 기술을 분석한다.

또한 2장 관련 연구에서 설명하였던 Hadoop의 HDFS를 활용하여 기존의 Hadoop 기반 색인 기법의 문제점을 개선하기 위한 효율적인 DB 인덱싱 기법을 제안한다.

3.1 효율적인 DB 인덱싱 기법 개요

본 장에서는 효율적인 DB 인덱싱 기법을 제안한다. 그림 3.1와 같이 DB 효율적인 DB 인덱싱 기법은 두 개 B+-tree와 HDFS로 구성되어 있다. 데이터와 색인 B+-tree가 있다. 데이터 B+-tree에서는 지속적으로 삽입 되는 데이터를 일정 크기까지 저장 후 가지고 있는 데이터는 HDFS에 저장하고 그 데이터에 대한 색인은 만들어서 색인 B+-tree에 저장한다. 색인 B+-tree는 데이터 B+-tree에서 생성되어 HDFS에 저장된 데이터 파일의 색인 데이터를 저장한다. 또한 데이터 탐색을 할 때에 해당 Key 값에 대한 색인을 제공하고 연속적인 범위의 Key 탐색에 대해서도 효율적인 탐색을 지원한다. HDFS에는 데이터 B+-tree의 데이터가 축적된 대용량의 데이터가 저장되며 색인 B+-tree에 데이터 파일의 색인 되어있다.

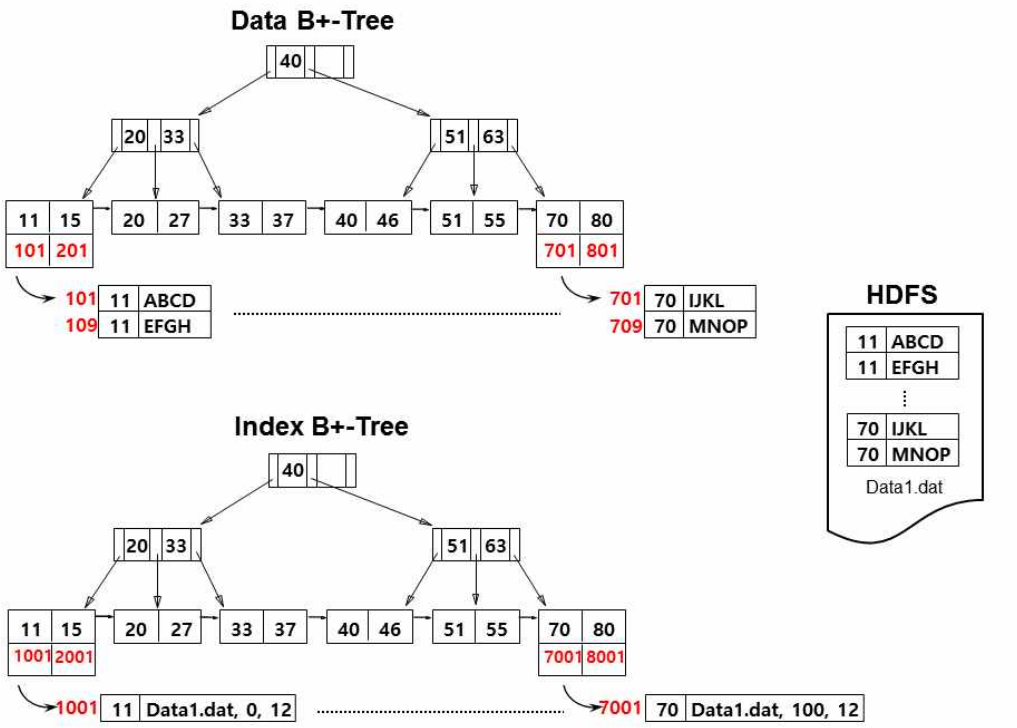

그림 3.1 효율적인 DB 인덱싱 기법

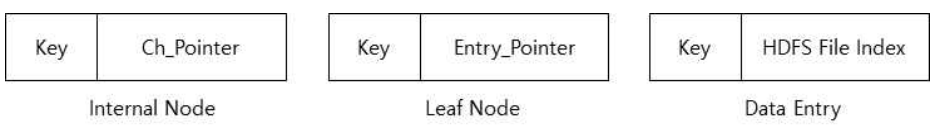

그림 3.2 색인 B+-Tree의 노드와 엔트리 구조

데이터와 색인 B+-tree의 구조는 계층 트리 형태의 색인 구조를 생성한다. 계층 트리는 내부 노드, 말단 노드, 엔트리로 구성된다. 데이터와 색인 B+-tree의 내부 노드는 검색 키 값의 범위와 자식 노드의 포인터로 구성된다. 데이터 B+-tree의 말단 노드에는 데이터 B+-tree에는 색인을 생성하기 전의 데이터의 키 값과 데이터 엔트리 리스트의 포인터가 들어있고, 색인 B+-tree에는 데이터 키와 색인 데이터 엔트리 리스트의 포인터가 들어있다. 색인 데이터 엔트리는 HDFS 파일의 데이터를 탐색 할 때 HDFS 파

일 단위별로 분할 처리하기 때문에 HDFS 파일에서 데이터의 키 별로 파일의 위치를 구분한 HDFS File 정보를 사용한다. 색인 데이터 엔트리의 데이터의 HDFS File 정보에는 HDFS에서의 데이터 파일 이름과, 해당 검색 키 값의 시작 Offset과 데이터의 길이가 저장된다.

3.2 효율적인 DB 인덱싱 기법 요구사항

본 논문에서 제안하는 Hadoop 기반 데이터 웨어하우스 시스템을 위한 효율적인 데이터베이스 색인 기법을 설계하기 위한 요구 사항을 분석하면 다음과 같다.

- **추가적인 데이터에 대한 Update 지원 :** 색인이 생성된 데이터가 아닌 추가된 데이터에 대해서만 색인을 생성하여 생성 시간을 감소시킨다.

- **메인 메모리에 색인 저장 :** 색인 탐색 속도를 증가시키기 위해 디스크의 I/O시간이 감소하는 메인 메모리에 색인을 저장하여 탐색한다.

3.3 효율적인 DB 인덱싱 기법 기능 블록 다이어그램

제안하는 효율적인 DB 인덱싱 기법의 시스템은 그림 3.3과 같이 Server-Client 구조로 이루어져 있다. Sever의 CI와 DS가 색인을 구성하며 HDFS Client는 DB와의 인터페이스 역할을 한다. DB에는 HDFS로 데이터를 저장한다. CI는 Client의 데이터의 삽입 요청이 발생하면 데이터

B+-tree에 저장하고 HDFS의 블록 크기가 되면 HDFS에 색인 생성 모듈을 이용하여 색인을 생성한다. 색인 생성되면 색인 B+-tree에 저장하고 HDFS에는 데이터를 HDFS의 데이터 파일로 저장한다. DS는 Client가 데이터의 탐색을 요청하면 색인 B+-tree에서 요청 Key에 대한 색인 데이터를 찾아서 데이터 탐색 모듈에 전달한다. 데이터 탐색 모듈에서는 HDFS Client를 이용해 색인 데이터의 정보(HDFS 파일 이름, Offset, 데이터 길이)를 이용하여 HDFS에 저장된 데이터에서 탐색하여 Client에게 제공한다.

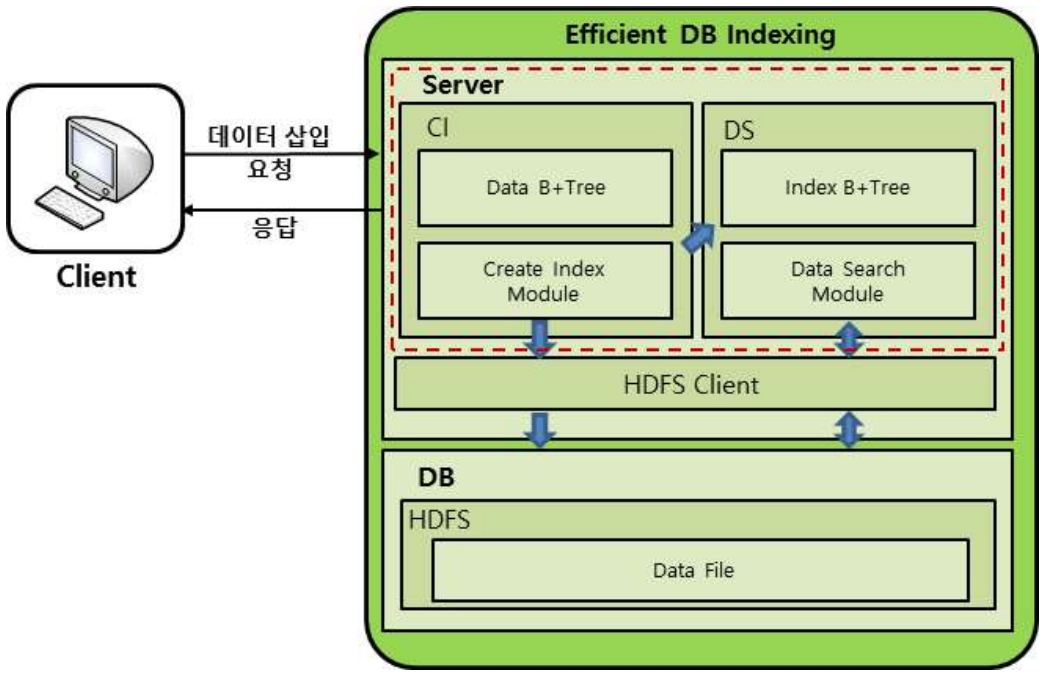

그림 3.3 색인 시스템 기능 블록 다이어그램

3.4 효율적인 DB 인덱싱 기법 데이터 흐름도

제안한 시스템은 사용자로부터 입력 받은 데이터를 색인하여 HDFS 저장 후 다시 사용자의 데이터 요청 시 해당 데이터를 전송해 주는 구조를 가진

다. 그림 3.4는 효율적인 DB 인덱싱 기법의 데이터 흐름을 나타내는 데이터 흐름도이다.

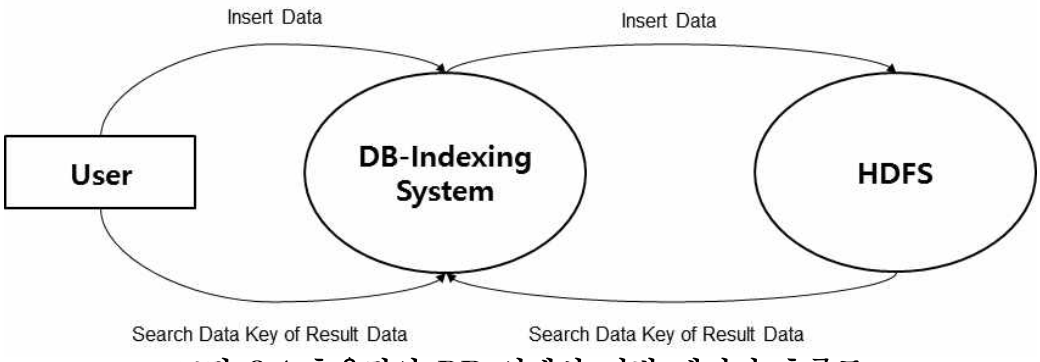

그림 3.4 효율적인 DB 인덱싱 기법 데이터 흐름도

사용자는 데이터를 삽입하여 HDFS에 저장할 수 있고 삽입한 데이터를 데이터의 키를 통하여 해당 키의 데이터를 탐색할 수 있다.

효율적인 DB 인덱싱 기법은 사용자에게 입력 받은 데이터를 받아서 HDFS에 적재, 색인을 생성하고 사용자가 요청하는 데이터를 색인을 통해 HDFS에서 탐색한다.

3.5 효율적인 DB 인덱싱 기법 설계

본 절에서는 Hadoop 기반 데이터 웨어하우스에 효율적인 색인 기법과 알고리즘과 그 세부절차에 대하여 설계한다.

3.4.1 색인 생성 알고리즘

색인 생성은 개요도는 그림 3.5와 같다. 데이터 B+Tree에서 Client가 삽

입한 데이터가 B+-tree에서 수용 가능한 데이터 크기가 넘어가게 되면 색인 생성을 하고 알고리즘은 그림 3.6과 같이 설계하였다.

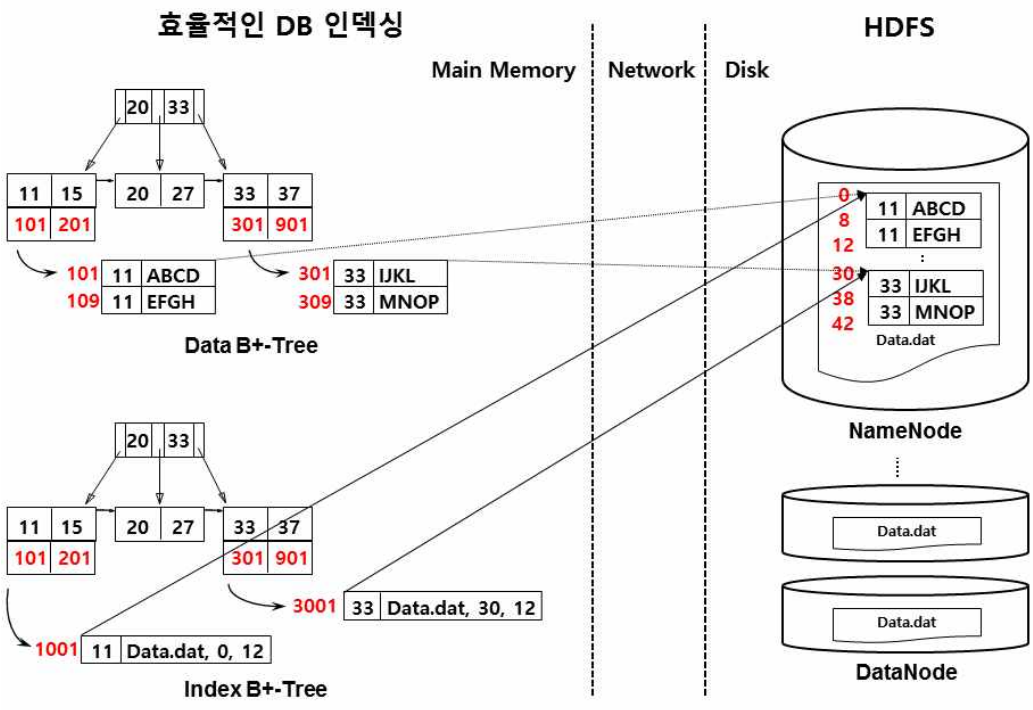

그림 3.5 색인 생성 개요도

HDFS 데이터를 저장할 파일 이름을 초기화한 후에 데이터가 있는 데이터 B+-tree의 모든 말단 노드에서 색인을 생성하게 된다. 각 말단 노드의 데이터 노드에서 각키 별 모든 데이터의 길이를 계산하고 마친 데이터는 데이터를 파일에 기록한다. 새로운 키 전까지 현재 키에 대한 전체 길이를 구하고 색인 데이터(Key, HDFS_FileName, Offset, Key_Length)를 색인 B+-tree에 삽입한다. 이후 전체 길이를 Offset에 더하여 다음 키에 Offset을 생성한다. 모든 키에 색인 생성을 마친 로컬 시스템에 있는 데이터 파일을 HDFS에 적재한다.

알고리즘 1. 색인 생성(Create Index)

```
1:   set HDFS_FileName // HDFS에 저장할 File 이름 입력
2:   do
3:      set Offset Initialize  // Offset 초기화
4:      for each LeafNode of Entry // 말단 노드의 엔트리가 있을 때 까지
5:         set Key_Length Initialize // 키의 길이 초기화
6:         for each Entry of Data // 엔트리의 데이터가 있을 때 까지
7:            Key_Length += Data of Length  // 데이터의 길이를 키 길이에 더하기
8:            write data to Temp_File  // 데이터를 Temp 파일에 쓰기
9:         end for
10         InsertIndex(Key, HDFS_FileName, Offset, Key_Length) // 색인 삽입
11:        Offset += Key_Length // 키의 길이를 오프셋에 더하기
12:     end for
13:  until LeafNode is Not Null // 말단 노드가 없을 때 까지
14:  store into HDFS from Local Temp_File // 로컬의 Temp 파일을 HDFS에 저장
```

그림 3.6 색인 생성 알고리즘

3.4.2 데이터 탐색 알고리즘

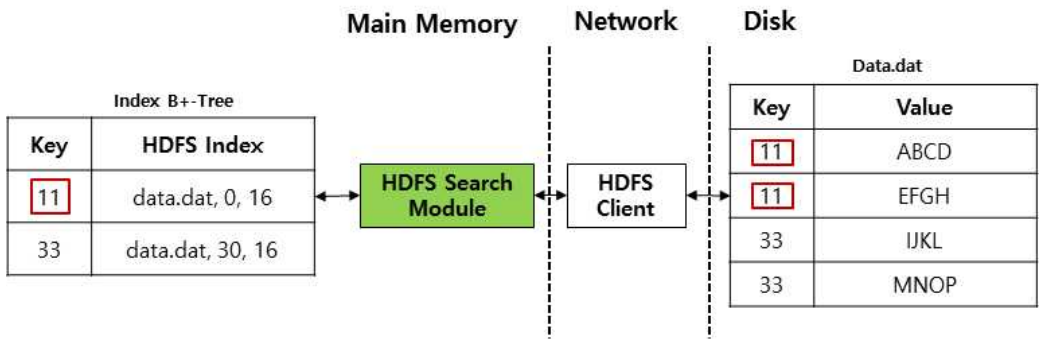

그림 3.7 데이터 탐색 개요도

색인 탐색의 개요도는 그림 3.7와 같다. Client가 요청하는 키 값들의 데이

터 리스트를 색인 B+-tree에서 내부 노드와 말단 노드를 돌면서 탐색하는 키 값을 가지고 있는 모든 엔트리를 찾게 되며 이의 데이터 탐색의 알고리즘은 그림 3.8과 같이 설계하였다.

알고리즘 2. 데이터 탐색(Search Data)

```
1:  set Search_Entry_List // 색인 엔트리 리스트 입력
2:  set Search_Data_List Initialize // 색인 데이터 리스트 초기화
3:  for each Search_Entry_List of Entry // 탐색 키 엔트리 리스트가 있을 때 까지
4:      for each Entry of Data // 엔트리에 색인 데이터가 있을 때 까지
5:          add Data to Search_Data_List  // 색인 데이터 리스트에 색인 데이터 추가
6:      end for
7:  end for
8:  for each Search_Data_List of Data // 색인 데이터 리스트에 데이터가 있을 때 까지
9:      HadoopRead(FileName, Offset, Length) // Hadoop에서 탐색 데이터 읽기
10: end for
```

그림 3.8 데이터 탐색 알고리즘

탐색 엔트리 리스트를 입력 받은 후 리스트에 있는 각 엔트리의 HDFS 색인 데이터를 색인 데이터 리스트에 저장한다. 색인 데이터 리스트에 있는 모든 색인 데이터의 정보를 이용하여 Hadoop의 HDFS의 저장된 데이터를 전부 탐색한다. HadoopRead에서는 HDFS의 파일 이름으로 파일을 열고 파일의 Offset으로 처음 탐색 키 데이터의 레코드에서부터 Length로 마지막 레코드까지 읽어 HDFS에서 로컬 시스템으로 탐색키의 데이터를 적재한다.

IV. 효율적인 DB 인덱싱 기법 구현 및 평가

본 장에서는 3장에서 설계한 Hadoop기반 데이터 웨어하우스 시스템을 위한 색인의 생성과 데이터 탐색 알고리즘을 사용하여 효율적인 DB 인덱싱 기법 시스템을 구현하고, 해당 시스템을 테스트 및 평가하여 제안 기법의 성능을 평가한다.

4.1 구현 환경

본 논문에서 제안한 Hadoop 기반 데이터 웨어하우스 시스템을 위한 효율적인 데이터베이스 색인 기법은 Windows 10기반 운영체제를 사용하였다. 구현을 위한 언어는 Jdk 1.8과 Hadoop 1.2.1 가상 머신은 VirtualBox에서 Fedora22 운영 체제를 사용하였다. 효율적인 DB 인덱싱 기법의 구현 환경은 표 4.1과 같다.

표 4.1 구현 환경

실험 환경	값
OS	Window 10
RAM	8G
CPU	Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
NETWORK	100Mbps
HYPERVISOR	VirtualBox 5.1.8
GUEST OS	Fedora 19
GUEST RAM	4G
LANGUAGE	Jdk 1.8 / Hadoop 1.2.1

4.2 테스트 및 평가

4.2.1 테스트 환경

테스트 툴은 OLAP의 성능을 측정하는 TPC-H BenchMark Tool[17]을 사용하였다. HDFS는 2.7.1 버전, Java는 1.8 버전이 사용되었다.

표 4.2 하둡 Master 실험 환경

실험 환경	값
OS	OS X Yosemite(10.10.3)
RAM	8G
CPU	Intel(R) Iris(TM) i5-4570R CPU @ 2.70GHz
NETWORK	100Mbps
HYPERVISOR	VMware Fusion 7.1.2
GUEST OS	Fedora 22
GUEST RAM	6G
LANGUAGE	Jdk 1.8 / Hadoop 2.7.1

표 4.3 하둡 Slave 실험 환경

실험 환경	값	
NODE	Slave1 (SecondaryNameNode)	Slave2
OS	OS X Yosemite(10.10.3)	OS X Yosemite(10.10.3)
RAM	4G	4G
CPU	2.5 GHz Intel Core i5	3.06 GHz Intel Core i3
NETWORK	100Mbps	100Mps
HYPERVISOR	VMware Fusion 7.1.2	VMware Fusion 7.1.2
GUEST OS	Fedora 22	Fedora 22
GUEST RAM	4G	4G
LANGUAGE	Jdk 1.8 / Hadoop 2.7.1	Jdk 1.8 / Hadoop 2.7.1

테스트는 표 4.2와 표 4.3과 같이 1대의 마스터 노드와 2대의 슬레이브

노드에서 수행되었고 테스트에 사용된 HDFS 블록의 크기는 기본 설정 값인 64MB이며 복제 본 개수는 총 세 개다.

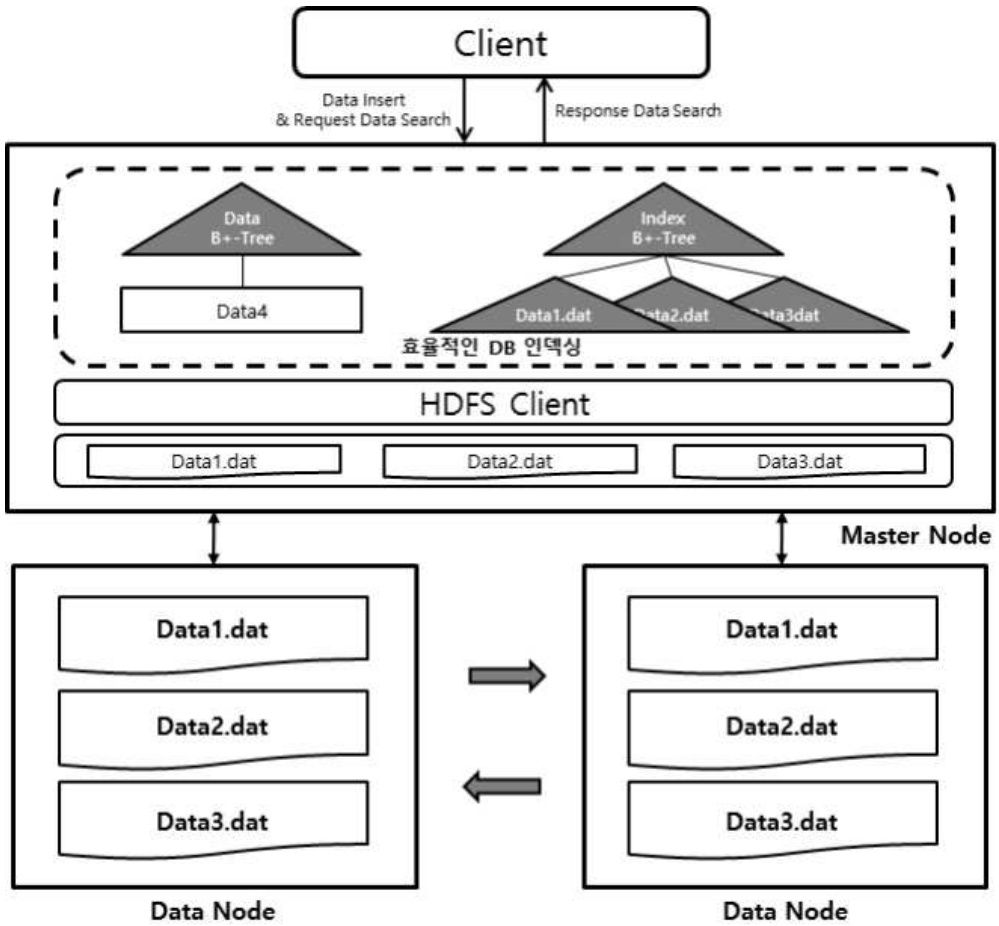

그림 4.1 실험 환경 구조도

그림4.1은 최초 사용자로부터 데이터를 입력 받아 데이터가 데이터 B+-tree에 쌓이고 일정크기로 쌓인 데이터를 색인하여 색인 B+-tree에 삽입하고 데이터는 HDFS에 적재되거 사용자의 데이터 탐색 요청 시 데이터를 탐색 실험 과정을 전체 구조도로 보인다.

위에 설명한 것과 같이 HDFS Client는 Data B+-tree로부터 데이터를 받아서 HDFS의 데이터 노드에 데이터를 저장 하거나 색인 B+- 트리에서 색인 데이터의 정보로 HDFS에 있는 데이터 노드의 파일을 탐색하는 작업을 수행한다.

4.2.2 테스트 데이터

실험에 사용하는 데이터는 TPC-H의 lineitem 데이블을 사용하였으며 색인 키는 l_orederykey를 선정하였다. 천만 개, 천5백만 개, 2천만 개를를 가진 총 세 개의 테스트 파일이 사용되었으며 각 파일 상세 정보는 표 4.4와 같다.

표 4.4 테스트 데이터

레코드 개수	생성 데이터 크기
천만 개	약 1 GB
천5백만 개	약 1.5 GB
2천만 개	약 2 GB

제안하는 색인 기법의 효율성을 증명하기 위하여 Hive 색인과의 색인 생성 시간, 데이터 탐색 시간의 성능 비교를 진행하였다. Hive 색인은 Hive에서 기본으로 사용하는 색인인 Hive Compact index가 사용되었으며 입력 및 저장 포맷은 모두 기본 텍스트 파일 포맷으로 설정하였다. 쿼리는 SELECT * FROM lineitem WHERE l_orderkey == [Variable]을 사용하였다.

4.3 테스트 결과 분석

4.3.1 색인 생성 시간

그림 4.2는 기존 기법과 제안 기법의 색인 생성 시간을 비교한 결과이다. 제안 기법은 기존 색인 생성 비용에 비해 약 60% 더 빠른 성능을 보이고 Hive 색인은 매우 큰 생성 비용이 소모함을 확인할 수 있다. Hive 색인 기법은 색인을 생성할 때 입력 데이터를 HDFS에서 읽어 들이고 MapReduce를 이용하여 데이터 처리를 하기 때문에 추가적인 네트워크 비용과 디스크의 비용이 들어간다. 제안 색인 기법은 색인 생성한 데이터에 대해서 로컬 디스크에 저장하는 디스크 비용이 들지만 색인 생성시 한 노드에서 독립적으로 메모리 기반에서 색인을 생성하기 때문에 Hive 기반에 비하여 추가적인 비용이 들어가지 않는다.

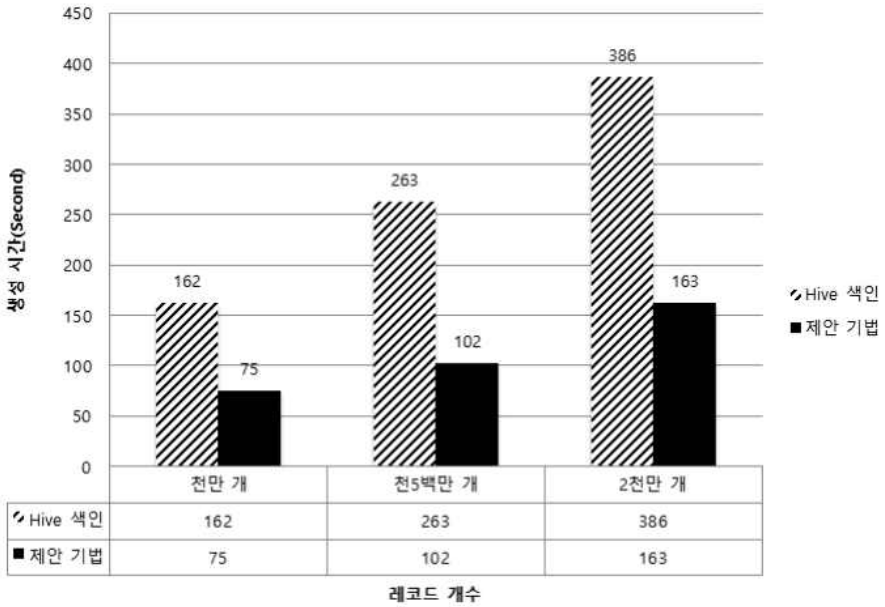

그림 4.2 색인 생성 시간 비교

4.3.2 데이터 탐색

색인 탐색 실험은 전체 데이터 검색 작업이 종료했을 때를 기준으로 하고 검색 키는 데이터 범위 내에서 무작위 값을 선정해 수행되었다. 그림 4.3은 Hive 기법과 제안 기법의 탐색 속도를 나타낸다. 제안 기법은 기존 색인 생성 비용에 비해 약 10배 더 우수한 탐색 속도를 보여준다. Hive 색인 기법은 색인 데이터의 탐색이 디스크 기반인 HDFS에서 수행되고 전체 탐색을 하기 때문에 추가적인 디스크 IO 비용과 네트워크 비용이 들어간다. 제안 기법은 색인 기법은 메모리 기반인 B+-tree로부터 수행한다. 네트워크의 부하와 디스크의 읽기 비용이 없고, 부분 적인 색인 탐색하기 때문에 Hive 색인보다 탐색 시간이 감소되었다.

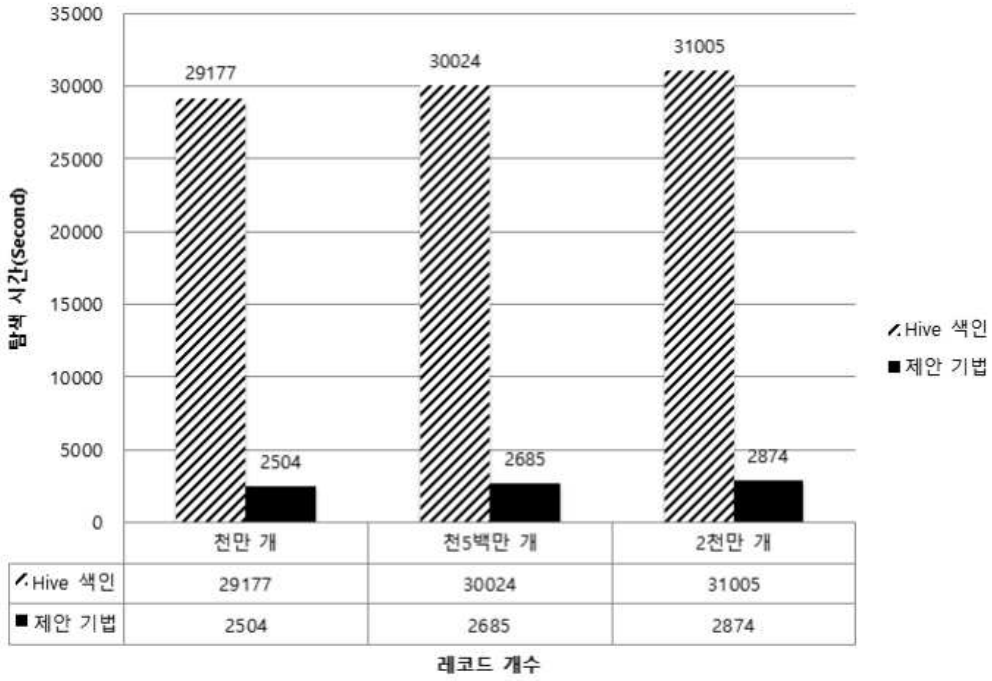

그림 4.3 데이터 탐색 시간 비교

V. 결론

최근 빅데이터 시대가 도래하며 기존에 기업의 경쟁력 강화를 위해 구축된 데이터 웨어하우스(Data Warehouse) 환경의 변화가 요구되었다. 기존 관계형 DBMS는 빅데이터 처리에 한계가 있어서 낮은 비용과 확장성이 용이한 Hadoop을 이용하여 처리하기 시작하였다.

기존의 Hadoop 기반의 색인 기법들은 색인 생성 시간은 색인의 Update가 불가하여 전체 파일 단위로 재생성하기 때문에 증가하거나 데이터 탐색 시간은 색인 파일을 DB에 저장하여 디스크 I/O를 증가로 데이터 탐색 시간이 증가하는 문제가 있다.

따라서 본 논문에서는 이러한 문제를 개선하기 위하여 제안하는 기법에서는 색인의 추가적인 Update를 가능하게하기 위해 데이터 B+-tree를 이용하여 추가된 데이터에 대해서만 일정 크기로 색인을 생성하고 색인의 탐색 시간을 단축하기 위해 메모리 기반 B+-tree에서 색인 데이터를 저장, 관리하는 향상된 색인 기법을 제안하였다.

본 논문에서 제안하는 색인 기법은 사용자에게 입력 받은 데이터를 일정 크기로 (Key, FileName, Offset, Length) 형태의 색인을 생성하고 HDFS에 적재한 후에, 사용자가 요청하는 데이터를 색인을 통해 HDFS에서 탐색하는 색인 기법을 설계하였다.

본 논문에서 설계한 효율적인 DB 인덱싱 기법을 활용한 테스트 시스템을 구현하기 위하여 Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz 환경에서 Jdk1.8, Hadoop 1.2.1언어를 사용하였으며, Hadoop의 맵리듀스와 HDFS를 이용하여 효율적인 DB 인덱싱 기법을 구현하였다.

본 논문에서 설계 및 구현한 효율적인 DB 인덱싱 기법을 실험 및 평가하기 위하여 TPC에서 제공하는 벤치마크 중 OLAP 용으로 사용하는 TPC-H를 사용하여 진행하였다. TPC-H는 테스트 데이터 테이블과 쿼리를 제공하는 툴이다. 이를 사용하여 제안 기법이 기존 기법에 대비 얼마나 성능이 향상되었는지 확인하기 위하여 색인 생성과 데이터 탐색 항목으로 성능 테스트를 진행하였다.

테스트 결과 제안한 색인 기법이 기존 색인 기법보다 색인 생성 속도가 60% 향상된 생성 속도를 나타내었으며, 데이터 탐색 속도는 10배 향상된 성능을 확인하였다.

본 연구에서 제안하는 색인 기법은 Hadoop 기반 데이터웨어하우스 시스템에서 빠른 데이터 분석에 활용할 수 있을 것으로 사료된다.

향후 연구로는 더 많은 차원을 색인할 수 있는 기법들에 대한 연구가 필요하다.

참 고 문 헌

[1] S. Madden, “From databases to big data,” IEEE Internet Compu ting, vol. 16, no. 3, pp. 4-6, 2012.

[2] S. Singh and N. Singh, “Big Data Analytics,” 2012 International Conference on Communication, Information and Computing Technology, 2012.

[3] W.H.Inmon, DerekStrauss, GeniaNeushloss, “The Architecture for the Next Generation of Data Warehousing,” ELSEVIER, Ji&Son, 2009.

[4] S. Chaudhuri , U. Dayal, “An overview of data warehousing and OLAP technology,” ACM SIGMOD Record, vol. 26, pp. 65-74, 1997.

[5] A. McAfee, E. Brynjolfsson, T.H. Davenport, D. Patil and D. Ba rton, “Big data,” The Management Revolution, Harvard Bus Re v, vol. 90, pp. 61-67, 2012.

[6] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu and R. Murthy, “Hive-a petabyte scale data warehouse using hadoop,” 2010 IEEE 26th International Confer ence on Data Engineering (ICDE), pp. 996-1005, 2010.

[7] R. Cattell, “Scalable SQL and NoSQL data stores,” ACM SIGMOD Record, vol. 39, pp. 12-27, 2011

[8] J. Dean and S. Ghemawat, “MapReduce: simplified data processi ng on large clusters,” Communications of the ACM, vol. 51, pp. 107-113, 2008.

[9] Vu. Trong-Tuan, F. Huet, “A Lightweight Continuous Jobs Mechanism for MapReduce Frameworks,” Proc.ofCluster,Cloud and GridComputing, pp.269-279,2013.

[10] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. “The Google file system,” ACM SIGOPS Operating Systems Review. Vol. 37.No. 5. ACM, 2003.

[11] Shvachko, Konstantin, et al. “The hadoop distributed file system,” Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE, 2010.

[12] Hadoop : http://hadoop.apache.org.

[13] V.K. Vavilapalli, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah and S. Seth, “Apache hadoop yarn: Yet another resource negotiator,” In Proceedings of the 4th Annual Symposium on Cloud Computing, 2013

[14] Mingyuan An, Yang Wang, Weiping Wang, “Using Index in the MapReduce Framework,” Web Conference (APWEB), 2010 12th International Asia-Pacific, 2010

[15] Vinitha Reddy Gankidi, Nikhil Teletia, Jignesh M. Patel, Alan Halverson, David J. DeWitt, “Indexing HDFS Data in PDW:

Splitting the data from the index," Proceedings of the VLDB Endowment Volume 7 Issue 13, August 2014 Pages 1520-1528, 2014

[16] Douglas Comer, "Ubiquitous B-Tree," ACM Computing Surveys Volume 11 Issue 2, June 1979 Pages 121-137, 1979

[17] TPC-H, http://www.tpc.org/tpch/

[18] 장한을, "모바일 클라우드 컴퓨팅 서비스를 위한 Hadoop 기반의 어플리케이션 캐쉬 테이블의 설계 및 구현", 세종대학교, 2012.

[19] 구윤선, "대용량 뉴스정보의 실시간 검색을 위한 UNIX ISAM 파일 색인 구조", 동국대학교, 2005.

[20] Thusoo, Ashish, et al. "Hive: a warehousing solution over a map-reduce framework," Proceedings of the VLDB Endowment 2.2 (2009), 2009.

ABSTRACT

Efficient Database Indexing Technipue for Data Warehouse System Based on Hadoop

Lee Jung Been
Advised by Prof. Seok Cheon Park
Dept. of Mobile Software
Graduate School of Gachon University

Existing enterprises tried to make better decisions by analyzing the business by using data generated from business activities in data warehouse system. But, due to the recent advent of big data era, the existing data warehouse environment, RDBMS, had a limit to which it can process therefore it was replaced with Hadoop based platform and researches were made using the indexing techniques to improve the search ability. However, the existing indexing techniques can not be updated or the data is processed on a disk basis so there are problems with index generation and increase in the search speed. Therefore, in this paper, itl proposedan efficient DB indexing technique that can update and reduce the search time of the index data to solve the problem above.

In order to design the test system for the indexing scheme

proposed in this paper, it was conducted with an environment of 1 master node and 2 slave nodes with Jdk1.7 Java language. Additionally, in order to find data from HDFS, it created index data and performed Hadoop based index technique which uses the created indexes to search for data in HDFS. Finally, the performance evaluation of the proposed efficient DB indexing technique used lineitem which is the data of TPC-H benchmark, it is used to measure the performance of data warehouse provided by TPC. By comparing the existing system's and the proposed system's time to create a new index and the search time, the proposed algorithm performed, compared to the existing algorithm, it was confirmed that for creation of new index was 60% faster and the search time was 10 times faster.

감사의 글

인연의 끈이 닿아서 대학원에 입학을 하게 되었고 벌써 2년이라는 시간이 지나 가게 되었습니다. 졸업을 앞두고 지난 2년을 돌아보니 석사과정을 마무리하면서 저한테 많은 도움을 주신 고마운 분들께 감사의 말씀을 전하고자 합니다.

먼저 저를 지금까지 열정적으로 지도해 주신 박석천 교수님께 진심으로 감사드립니다. 저에게 주신 가르침, 격려와 따뜻한 조언 덕분에 석사 학위과정을 끝까지 마칠 수 있었습니다. 교수님께 진심으로 감사드립니다. 많이 부족하지만 앞으로 교수님의 가르침을 마음에 새겨 부끄럽지 않은 모습을 보여드리도록 노력하겠습니다.

또한 바쁘신 와중에도 석사학위 논문의 심사를 맡아서 부족한 핵심적인 사항들을 보완하여 논문의 틀을 잡아 주신 김용수 교수님과 한기태 교수님께 감사드립니다.

그리고 오늘날 여기까지 결실을 맺기까지 부족한 저를 이끌어주신 천승태 소장님, DI 팀의 모든 선임님, 주임님, 연구원님들에게 모두 감사드립니다.

그리고 대학원에서 함께 지내며 많은 도움을 주신 선배님들, 후배님들에게 감사하다는 말을 전하고 싶고, 석사과정 2년 동안 동고동락한 동기들에게 그동안 수고했다는 말을 전하고 싶습니다.

마지막으로 언제나 저를 믿어주시고, 제 곁에서 많은 격려와 지원을 아끼지 않으신 사랑하는 부모님 감사드립니다. 항상 건강하세요. 또한 언제나 그래왔듯이 기쁠 때나 슬플 때나 힘들 때나 제 곁에서 평생을 든든하게 지켜줄 친구들에게 고마움을 전합니다.

이 외에 제가 미처 언급하지 못한 고마운 분들이 너무나 많습니다. 그분들의 이름을 모두 새기지 못함을 죄송하게 생각하며 대신 제 깊은 감사의 말로 이 글을 마칠까 합니다. “진심으로 감사드립니다.”

2017년 1월

이 정 빈