Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

A Recent Trend of Database for Big Data Handling using Key-value database

Kyung-Tae Song, Sang-Hyun Park*

Department of Computer Science, Yonsei University

ABSTRACT

Modern database systems use different methods from traditional database processing systems. Since more and more different types of data need to be stored, modern database systems have become more important to efficiently process big data through distributed processing and system scalability. This importance will continue to grow. For this reason, databases for big data processing are being studied and developed in a service that continuously deals with big data such as search engines and SNS services. In particular, the key-value database model has advantages of fast I / O and scalability. Therefore, it is suitable for big data processing and it is used as a system of various applications. Also, studies on key-value databases have been actively conducted in big data researches, because other big data database models also share the characteristics of the key-value database model and borrow many of the techniques used in the key-value database model. However, there are some areas that can further improve the key-value database such as optimization research, next-generation storage. In this paper, we define the key-value database that emerged as a hot topic in the big data platform field, list the related technologies and platforms being studied, and present a methodology to complement existing platforms.

KEYWORDS: Key-value stores, Big data, Database systems, NVRAM, SSDs, Distributed systems

ARTICLE INFO: Received 7 December 2016, Revised 2 January 2017, Accepted 10 February 2017.

*Corresponding author is with the Department of Computer Science, Yonsei University, 50 Yonsei-ro Seodaemun-gu, Seoul, 03722, Republic of Korea

E-mail address: *sanghyun@yonsei.ac.kr*

1. 서 론

SNS와 여러 모바일 기기의 등장에 따라 많은 양의 다양한 데이터들이 생산되고 있다. 기존의 데이터베이스는 정형화된 데이터만을 처리하며, 많은 양의 데이터를 처리하기에는 부족하였다. 미국의 정보 기술 및 자문 회사인 가트너(Gartner)는 이러한 데이터를 데이터의 양(Volume), 데이터 처리 속도(Velocity), 데이터 종류의 다양성(Variety)을 특징으로 갖는 빅데이터라고 정의하였다 [1]. 최근의 데이터베이스 시스템은 기존 전통적인 데이터베이스 시스템과는 다른 방식의 접근이 필요하다. 시간이 지날수록 많은 양의 데이터가 생산되고 있고, 이는 확장성이 적은 기존의 관계형 데이터베이스에서 다루기에는 데이터의 크기가 너무 큰 상황이다. 이에 따라 빅데이터를 처리하기 위해 NoSQL 혹은 NewSQL이라 불리는 많은 데이터베이스들이 등장하였다. 이들 중 주로 사용되는 빅데이터 데이터베이스는 크게 4가지로 분류할 수 있다. Key-value 데이터베이스, 문서형 데이터베이스, 그래프형 데이터베이스, 컬럼-지향 데이터베이스가 그것이다. 주류로 쓰이는 데이터베이스들 중 특히 key-value 데이터베이스는 다른 데이터베이스들의 분모 집합에 해당되는 데이터베이스라 할 수 있어서 다양한 연구가 진행되어 왔으며, 비교적 간단한 데이터 모델을 사용하기에 SSD, NVRAM 등의 새로운 저장 장치들이 개발됨에 따라 더욱 많은 방법들이 적용될 수 있는 데이터베이스이다. [2-4] 특히, 비휘발성의 바이트 접근 가능한 NVRAM은 차세대 저장 장치로 떠오르고 있으며 key-value 데이터베이스 관련 연구는 많이 진행되고 있지 않아 앞으로 이를 적용한 연구가 많이 진행될 것으로 예상된다.

우선, key-value 데이터베이스는 데이터를 저장할 때 key 와 value 만을 저장하며, 맵, 해시 등을 사용한 사전식 데이터 모델을 사용한다. 사전식 데이터 모델은 트리 밸런싱과 같은 비용이 없어서 빠른 데이터 입출력이 가능한 모델이다. 이러한 구조에서 key값과 value값은 특별히 정의되어 있지 않기 때문에 구조적인 데이터(structured data)와 비구조적인 데이터를 모두 다룰 수 있다. 이것은 다양성이 필요한 빅데이터 처리에 있어 적합한 처리 방식이라고 할 수 있다. 게다가 key-value 데이터베이스는 스키마와 같이 미리 정의된 데이터 타입을 사용하지 않기 때문에 RDBMS보다 빠른 데이터 입출력이 가능하다. Key-value 데이터베이스는 주로 Get, Set, Delete의 간단한 API를 제공하며, RDBMS보다 비교적 가벼운 시스템이기 때문에 내장형 데이터베이스에 많이 사용된다. Key-value 데이터베이스의 예로는 레디스(Redis), RocksDB [5,6] 등이 있으며, key-value 데이터베이스 관련 연구들이 활발히 이루어지고 있다. 다른 데이터베이스들은 key-value 데이터베이스의 하위 분류로 구분될 수 있다. 기존의 key-value 데이터베이스와 다른 점은 데이터를 처리하는 방식이다.

Key-value 데이터베이스의 하위 분류 중 컬럼-지향 데이터베이스는 디스크 기반의 데이터베이스에 많이 쓰인다. 이 시스템은 데이터를 처리할 때 행(row) 기준이 아닌 열(column) 기준으로 처리한다. 기존의 데이터베이스는 행을 기준으로 데이터들을 처리하였는데, 이는 연락처와 같이 특정 객체에 대한 여러 정보를 가져올 때 효율적이다. 그러나, 열 단위의 접근이 필요한 범위형 질의와 같은 작업을 처리할 때 비효율적이다. 빅데이터 시대에서 컬럼-지향 데이터베이스의 기원은 구글사의 빅테이블(Bigtable) [7] 이다. 구글은 수많은 웹페이지들에 대하여 효율적으로 처리하기 위해 빅테이블을 만들었다. 특히, 빅테이블에서 사용한 LSM-트리 [8] 구조는 디스크 상에서 빠른 쓰기 연산을 가능하게 하였고, Hbase, Cassandra [9,10] 등 다른 컬럼-지향 데이터베이스들에 많은 영향을 미쳤다.

Key-value 데이터베이스는 캐시의 역할, 내장형 데이터베이스의 역할, 데이터베이스 자체로서의 역할을 할 수 있어 다양한 방면에서 연구 할 부분이 존재한다.

본 논문에서는 key-value 데이터베이스를 개발하고 활용하여 빅데이터를 다루는 것이 중요한 연구 주제라고 판단하여 이와 관련된 데이터베이스 시스템들과 그 원리에 대해 살펴보고자 한다. 논문의 구성은 다음과 같다. 2장에서는 key-value 데이터베이스를 이해하기 위한 사전 지식을 설명하고, 3장에서는 현재 많이 쓰이고 있는 key-value 데이터베이스와 key-value 데이터베이스 관련 최신 연구들을 소개한다. 4장에서는 현재 사용되고 있는 key-value 데이터베이스들을 토대로 향후 빅데이터 처리 시스템들의 동향 제시와 결론을 기술한다.

2. 관련연구

빅데이터 데이터베이스가 많아진 이유 중 하나는 데이터베이스 시스템 상에서 모든 기능을 전부 취할 수 없다는 점 때문이다. 이 문제는 여러 학자들에 의해 증명 되었고, 이 중 가장 많이 인용되는 정리는 Brewer의 CAP정리 [11]다. 2000년에 Brewer가 PODC에서 CAP정리를 제시하였고, 2002년 Gilbert와 Nancy [12] 가 이 정리를 증명하였다. CAP정리란, 분산 환경에서 데이터의 일관성(consistency), 가용성(availability), 분할 내성(partition tolerance)을 동시에 모두 만족시킬 수 없고, 시스템에 따라 선택적으로 두 가지 요소만을 만족 할 수밖에 없다는 것이다. 일관성이란 모든 노드가 요청에 대하여 올바른 결과를 제공해야 한다는 것이고, 가용성은 모든 요청이 결과적으로 응답을 받을 수 있다는 것이며 분할 내성은 임의의 메시지가 누락되더라도 네트워크에서 용인한다는 뜻이다. 특히 빅데이터 데이터베이스 시스템 상에서 분할 내성은 제외할 수 없는 요소이기 때문에 많은 시스템들이 일관성과 가용성 중 하나를 선택한다. CAP정리는 많은 빅데이터 시스템을 설명할 수 있었으나, 2000년대 이후 많은 서비스들이 등장하면서 더욱 다양한 플랫폼들이 나타났다. Brewer은 2012년의 글 [13] 을 통해 CAP정리의 설명이 부족하고, 오해가 있었다고 하였다. 대부분의 빅데이터 데이터베이스가 정상적인 상태와 장애 상태에서 다르게 행동하기 때문이다. Abadi는 CAP의 설명을 보충해줄 수 있는 PACELC정리 [14] 를 제시하였다. 〈그림 1〉 은 PACELC 정리를 설명한다. PACELC 정리는 PAC와 ELC로 구분 지을 수 있다. PAC는 장애상황 과 같이 네트워크 분할이 필요한 경우(Partition) 가용성(Availability)과 일관성(Consistency) 중 양자 택일을 해야 한다는 것이다. 반면 ELC는 정상상황에서의 trade-off를 의미 한다. 장애 상황이 아닐 경우(Else) 시스템은 속도(Latency)와 일관성(Consistency) 중에서 선택해야 한다는 것이다. 이는 기존 CAP정리의 설명 보다 더 명확하게 많은 빅데이터 데이터베이스들을 설명할 수 있게 되었고 key-value 데이터베이스 또한 PACELC로 설명될 수 있다.

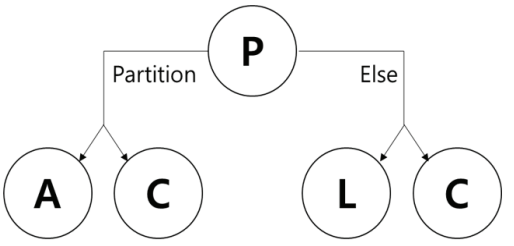

그림 1. PACELC 정리
Figure 1. PACELC theorem

3. Key-value 데이터베이스

다양한 key-value 데이터베이스들이 있지만 본

논문에서는 key-value 데이터베이스들을 크게 두 가지로 분류 하였다. 데이터를 메모리에 저장하여 빠른 처리를 가능하게 하는 in-memory 기반의 데이터베이스와 저장소로서의 기능에 집중한 디스크 기반의 데이터베이스다. 3장에서는 널리 쓰이는 key-value 데이터베이스와 최신 연구들에 대하여 소개하고 그 의의에 대하여 이야기 한다.

3.1 In-memory key-value 데이터베이스

3.1.1. Redis

레디스는 대표적인 오픈 소스 in-memory key-value 데이터베이스다. In-memory key-value 데이터베이스가 널리 쓰이게 된 것은 memcached [15] 의 역할이 컸으나, 다양한 데이터 구조와 기능을 제공한다는 장점이 있어 현재는 레디스가 가장 많이 사용되고 있다. 레디스는 데이터베이스, cache, 혹은 메시지 전달자의 역할을 수행할 수 있다. 레디스는 PC/EL 이라고 볼 수 있다.

레디스의 강력한 특징 중 하나는 다양한 데이터 타입을 제공한다는 것이다. 레디스에서는 특정 key에 대한 value로써 기본적인 문자열 저장 타입 이외에 해시, 리스트, 집합, 그리고 정렬된 집합을 저장할 수 있도록 제공한다. <그림 2> 레디스는 데이터를 저장할 때 해시 기반의 사전 구조체를 이용한다. 이는 빠른 데이터 입출력을 가능하게 하며, 레디스 클러스터에서는 해시 값 기반의 샤딩(sharding)을 통해 데이터를 분산하여 저장한다.

레디스가 널리 쓰이는 또다른 이유는 레디스 클러스터 때문일 것이다. 초창기 레디스 버전은 단일 노드에서만 작동 가능하였고, 이는 데이터베이스의 확장성에 영향을 미쳤다. 확장성은 빅데이터를 처리할 때 있어서 매우 중요한 요소인데, 최신 레디스 에서는 이를 해결한 레디스 클러스터 모드를

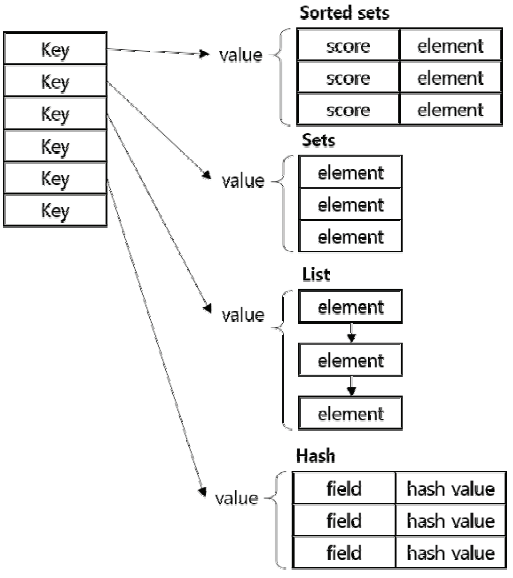

그림 2. 레디스의 데이터 타입
Figure 2. Data types of Redis

지원한다. 레디스 클러스터 모드는 레디스가 작동하고 있는 여러 개의 노드들을 묶어 하나의 데이터베이스처럼 작동한다. 또한 데이터 내구성을 위하여 마스터 노드의 데이터를 복제하여 슬레이브 노드에 저장한다. 하나의 마스터 노드에 여러 개의 슬레이브 노드가 연결되며, 마스터 노드에 문제가 생기면 복제된 데이터를 갖고 있던 슬레이브 노드가 마스터 노드의 역할을 하게 된다. 슬레이브 노드가 처음 마스터 노드와 연결될 때는 마스터 노드의 데이터를 이용하여 복구 작업을 수행한다.

레디스는 내고장성을 위해 주기적으로 데이터를 디스크에 저장하며 문제가 발생하였을 시 저장했던 데이터를 이용하여 데이터베이스를 복구한다. 레디스는 두 가지 복구 모드를 제공하는데, 하나는 AOF(Append Only File)이고, 다른 하나는 RDB(Redis DataBase)이다. AOF는 논리적 로깅 방법을 사용하는데 로그를 남길 때 입력 받은 명령과 key-value 값을 저장한다. RDB는 물리적 로깅 방법을 사용하여 전체 데이터베이스에 대한 스냅

샷을 저장한다. AOF 방식은 RDB로깅에 비해 상대적으로 가볍기 때문에 로깅의 주기가 짧아 좀 더 안정적이라는 장점이 있으나 로그 파일의 크기가 RDB보다 크다는 단점이 있다.

레디스는 많은 장점을 가지고 있으나 병렬처리를 지원하면 보다 빠른 처리를 가능하게 할 수 있을 것이다. 또한, 다른 key-value 데이터베이스와 같이 복구 방법을 더욱 최적화 할 수 있을 것으로 판단된다[16].

3.1.2. MICA

MICA는 빠른 데이터 처리를 위해 연구된 확장성을 띄는 in-memory key-value 저장소다. MICA는 캐시와 데이터 저장소로써 사용될 수 있으며, 전체적인 관점에서 다양한 방법들을 사용하여 데이터베이스 기능을 최적화 시켰다. 단, 클러스터와 내구성 면은 초점을 두지 않은 저장소다. MICA의 특징은 크게 3가지로 나눌 수 있다. 첫째, MICA는 분할된 데이터에 대하여 병렬 처리를 가능하게 하였다. 둘째, MICA는 효율적인 병렬 처리 및 빠른 네트워크 처리를 위해 커널을 거치지 않고 자체적으로 네트워크 인터페이스 카드(NIC)에 접근하여 데이터를 보낸다. 셋째, MICA는 in-memory key-value 데이터베이스에서 새로운 데이터 구조를 적용하였다.

MICA는 각 CPU 코어 당 하나 이상의 데이터 파티션을 만들고, key의 해시값을 기준으로 데이터를 분할한다. MICA는 EREW(Exclusive Read Exclusive Write) 혹은 CREW(Concurrent Read Exclusive Write) 방식으로 분할된 데이터에 접근한다. EREW 모드는 각 CPU 코어가 모든 작업을 자신에게 할당된 데이터 파티션 에서 처리하는 것이다. CREW 모드는 데이터 파티션에 접근 할 때 읽기 작업에 대해서는 동시 접근을 허용하되 쓰기 작업에 대해서는 배타적인 방법으로 할당된 코어만 접근 할 수 있는 방법이다. 두 가지 방법 모두 코어 간의 통신이 적고 동기화 문제가 적기 때문에 병렬화 된 데이터 접근에 있어서 좋은 효율을 보여준다.

MICA의 네트워크 입출력 방법은 기존의 통신방법과는 다르다. 기존의 통신방법들은 소켓 입출력과 같은 커널에 NIC 접근을 위임하는 방법을 사용한다. 이는 유저레벨에서 커널 레벨로 데이터 전달이 필요하고, 추가적인 비용이 소요된다. MICA는 인텔의 DPDK [18] 인터페이스를 사용한다. DPDK 인터페이스는 유저 레벨의 서버 소프트웨어가 운영체제의 도움 없이 직접 NIC를 조절할 수 있도록 하는 인터페이스를 갖고 있다. 즉, 기존의 다른 방법들 보다 더 적은 비용의 네트워크 처리가 가능하게 되었다.

MICA가 캐시로 활용되는 경우, 원형 로그를 사용하여 메모리를 관리하고, 손실가능한 해시 인덱스를 사용하여 데이터를 관리한다. 캐시로써 사용되는 경우 데이터는 로그의 끝에 데이터를 삽입하는 형식으로 저장된다. 원형 로그를 사용하기 때문에 선입 선출 방법으로 메모리를 관리하여 오래된 데이터가 먼저 삭제된다. 손실 가능한 해시 인덱스 또한 선입 선출 방법으로 관리되며 이 해시 인덱스는 CPU 캐시에서 사용되는 집합-연관 캐시와 비슷한 방식으로 인덱스를 유지한다. 해시 인덱스가 손실 가능하기 때문에 해시 충돌 등이 일어났을 경우 간편하게 처리할 수 있어 빠른 삽입이 가능하다. MICA가 저장소로써 사용되는 경우 손실가능했던 해시 구조를 chaining을 사용하여 비 손실 해시 구조로 변경한다. 또한 원형 로그가 아닌, malloc과 비슷한 방식의 메모리 관리 기법을 사용한다. 이는 캐시 방식의 선입 선출과는 달리, 메모리 내의 빈 공간들의 위치와 크기를 확인하며 할당해야 할 데이터의 크기보다 큰 공간 중 가장 작은 공간을 할당해 주는 방식으로 관리된다. 이를

통해 빠른 데이터 처리를 지원하면서도 데이터 손실이 일어나지 않도록 한다.

MICA는 in-memory 데이터베이스 관점에서 여러 가지 시도를 적용한 연구이다. 기존의 데이터베이스에서 사용하던 구조를 변형하였고, 데이터베이스 외의 다른 분야들에서 쓰이던 접근 방법을 사용하였다는 점에서 많은 연구자들이 참고하는 논문이다. 하지만 논문에서 언급하였듯 데이터 내구성과 클러스터 부분에 대한 부분은 주목하지 않았기 때문에 해당 부분과 관련한 연구가 진행되면 더욱 향상된 key-value 데이터베이스를 구축할 수 있을 것으로 보인다. 이 중 내구성 부분에 대한 내용은 비휘발성 저장 장치인 NVRAM을 활용함으로써 개선할 수 있을 것으로 판단된다. 또한, NVRAM은 SSD와 HDD보다 빠른 입출력 성능을 보이며 바이트 단위의 접근이 가능하기 때문에 기존 DRAM에 적용하였던 연구방향들을 적용할 수 있을 것으로 보인다. 클러스터 관련 연구는 레디스 클러스터를 참고하면 좋을 듯 하다.

3.2 디스크 기반의 key-value 데이터베이스

3.2.1. Bigtable

빅테이블(Bigtable)은 2006년 구글에서 발표하였고, 분산환경에서 구조적 데이터를 처리하기 위해 만든 컬럼-지향 key-value 데이터베이스다. PAELC 정리에 따르면 빅테이블은 PC/EC에 속한다. 빅테이블은 데이터를 저장할 때 분산 구글 File System(GFS) [19]를 사용하며, 주로 쓰기 연산이 많은 상황에서 디스크 접근이 필요한 경우 효율적으로 작동할 수 있도록 설계되었다. 빅테이블은 구글 SSTable 파일 포맷을 이용하여 데이터를 저장한다. SSTable은 key와 value가 임의의 값일 때 파일 내부에서 key 순서로 정렬되어 있는 상태를 유지한다. 이는 특정 key 범위에서 key/value 쌍을 순회하거나 key에 대한 value를 찾을 때 효율적인 구조이다. 빅테이블의 기본적인 데이터 구조는 <그림 3> 에 나타나 있듯 LSM-트리를 기반으로 한다. 빅테이블이 LSM-트리를 사용하면서 페이스북의 빅데이터 저장 플랫폼인 Cassandra, RocksDB를 포함하여 Hbase, MongoDB, 등 많은 빅데이터 데이터베이스들이 LSM-트리를 사용한다. 이에 따라 LSM-트리 관련 연구도 활발히 진행되고 있으며, 여러 논문에서 LSM-트리를 다뤘다 [20, 21]. LSM-트리는 log structured merge tree의 약자로, 트리 균형을 나중으로 미루고, 삽입 명령과 같은 쓰기 작업을 빠르게 처리할 수 있도록 데이터를 로그 형식으로 쌓는다. 빅테이블은 LSM-트리 구조를 이용하여 memtable이라는 하나의 메모리 내 테이블과 여러 개의 디스크 내의 SSTable 파일로 이루어진다. 멤테이블(Memtable)에서 처음 데이터가 삽입되거나 삭제되었을 때 해당 작업이 이루어지며, 멤테이블의 크기가 너무 커지면 SSTable로 데이터를 병합시킨다. 특히 데이터 삭제가 이루어질 때는 즉시 데이터가 지워지는 것이 아니라 데이터가 지워졌다는 표시만 남긴 후 트리 병합 과정을 통해 실제 삭제작업을 수행한다. 빅테이블은 compaction이라는

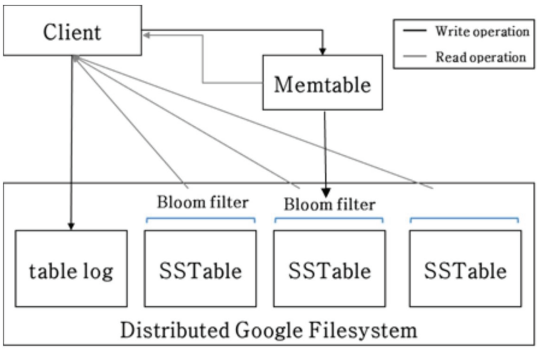

그림 3. 빅테이블의 LSM-트리 기반 구조
Figrure 3. LSM-tree based architecture of Bigtable

병합과정을 통해 멤테이블에 존재하는 데이터를 SSTable로 옮기고, SSTable 내의 오래된 데이터들을 제거한다. LSM-트리는 읽기 성능이 낮아지는 것을 감수하고 쓰기 성능을 높인 구조인데, 빅테이블은 읽기 성능을 빠르게 하기 위해 데이터 구조에 bloom filter를 적용하고, 자체적인 파일 타입인 SSTable 파일을 사용하며 캐싱을 활용하여 읽기 성능을 향상 시킬 수 있도록 최적화 하였다.

빅테이블은 행 범위를 기준으로 데이터를 분할한다. 빅테이블에서는 행 범위를 태블릿(tablet)이라 칭하며, 여러 개의 태블릿 서버들이 데이터를 관리한다. <그림 4>에 나타나듯, 태블릿 서버들의 계층 구조는 3단계로 이루어진다. 빅테이블은 Chubby [22] 라는 락 서비스에 의해 관리되는데, Chubby는 태블릿 계층 구조의 첫 단계인 루트 태블릿 서버에 대한 정보를 갖는다. 루트 태블릿은 메타데이터 테이블을 통해 다른 모든 태블릿들에 대한 위치 정보를 갖고 있고, 두 번째 단계인 태블릿 서버들이 실제 데이터 입출력들이 이루어지는 마지막 단계의 유저테이블을 관리한다.

빅테이블은 행, 열, 시간을 key로 사용하여 데이터에 접근한다. 빅테이블은 열 기준으로 데이터를 처리하는데, 이는 기존의 행 기준의 데이터베이스들 보다 같은 요소를 갖는 데이터들을 효율적으로 처리할 수 있도록 한다. 열에 대한 key는 묶어서 column family 라는 집합으로 사용하는데, 이는 데이터 접근을 조절하는 기본적인 단위이다. 그래서 열에 대한 key를 저장할 때 family:qualifier와 같은 문법으로 저장한다. 예를 들어, 웹사이트를 저장할 때 '언어'라는 요소를 키로써 사용한다면, '언어'라는 column family의 구분자로 각각의 언어가 들어간다. 이를 통해 같은 언어를 사용하는 웹사이트들을 쉽게 분류하고 처리할 수 있다.

3.2.2. HBase

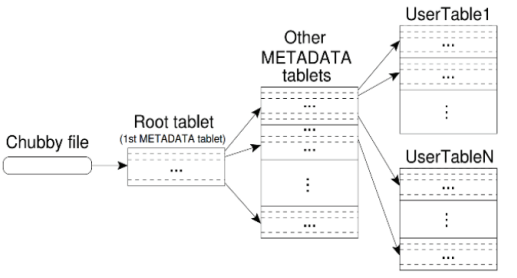

그림 4. Tablet 위치의 계층구조 [7]
Figure 4. Tablet location hierarchy

HBase는 Google의 빅테이블을 본 따 만든 분산처리 기반의 환경에서 데이터를 key-value 형태로 저장하는 아파치 오픈소스 프로젝트이다. HBase는 빅테이블과 유사한 구조를 갖기 때문에 PC/EC라 할 수 있다. 기존 대용량 배치 작업(Batch processing)에 적합한 형태로 설계된 하둡 분산 파일 시스템(Hadoop Distributed Filesystem)과는 달리 실시간 소규모 데이터의 입출력 처리를 목적으로 구조가 설계 되었다. 그러면서도 하둡 분산 파일 시스템에 HBase파일을 저장하는 형태를 취하여 하둡 환경과의 호환성을 보장하며 하둡 환경이 가진 특성으로 인하여 데이터 크기 증가에 따른 확장성 역시 보장한다.

<그림 5>에서 보듯이, HBase는 마스터(Hmaster), 주키퍼(Zookeeper) [23], 리전 서버(Regionserver) 세 가지 요소로 구성되어 있다. HBase는 데이터를 저장할 때 데이터가 가진 키를 기준으로 범위를 나눈 리전 단위로 저장한다. 마스터는 HBase 구동 시HBase를 구성하는 노드들에게 리전을 할당하고 리전 간의 로드 밸런싱 작업을 담당한다. 주키퍼는 아파치 오픈소스 프로젝트로 분산환경을 구성하는 클러스터들의 상태를 유지하고 조율하는 담당하는 역할을 한다. HBase에서도 주키퍼를 사용하여 리전에 해당하는 데이터를 저장하는 노드들에 대한 위치 정보 및 상태를 관리하며, HBase이 구성하는 데

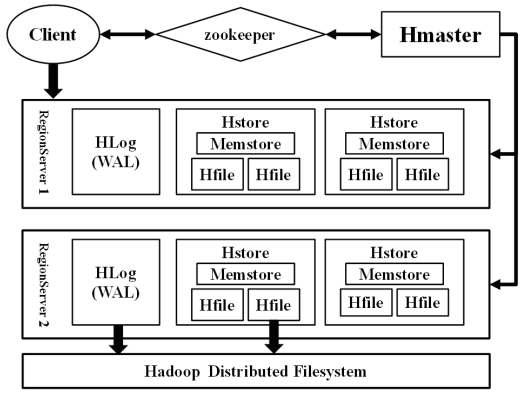

그림 5. Hbase의 구조
Figure 5. Hbase architecture

이터 테이블에 대한 메타 정보를 저장하는 테이블의 위치 정보를 저장하고 있기 때문에 분산 환경에서의 주키퍼의 역할은 필수적이다. 리전 서버는 하둡 분산 파일 시스템과 연동하여 실질적으로 데이터를 저장하는 역할을 한다. HBase 는 LSM-tree 기반의 저장 시스템으로, 리전 서버는 크게 메모리 기반의 데이터 저장소 멤스토어(Memstore)와 멤스토어 내 데이터에 대한 손실을 방지하기 위하여 필요한 Write Ahead Log(WAL) 형태의 HLog, 그리고 디스크 기반에 저장되는 파일 형태의 HBase 파일(Hfile)로 구성되어 있다. 사용자가 데이터를 입력하게 되면 처음에는 데이터 손실을 방지하기 위한 로그 정보를 디스크에 기록한 후 메모리 상의 멤스토어에 데이터를 저장하게 된다. 이후 저장되는 데이터의 크기가 지정한 멤스토어의 크기를 초과하게 되는 경우, 멤스토어 내부 데이터를 Hfile 형태로 디스크에 저장하게 된다. 타 key-value 데이터베이스와는 달리 HLog 및 Hfile 이 로컬 파일 시스템에 저장되는 것이 아닌, 하둡 분산 파일 시스템에 저장된다. Hfile의 구조는 〈그림 6〉 과 같다. Hfile 은 기본 64KB 단위의 데이터 블락과 함께 메타 정보 및 인덱스 블락으로 구성되어 있고, 한 데이터 블락 안에는 key-value가 일정한 순서로 정렬

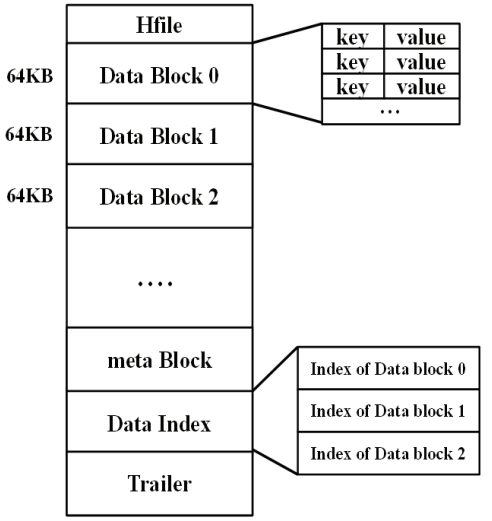

그림 6. Hfile의 구조
Figure 6. Hfile architecture

이 되어 저장되게 되며, 인덱스 블락에는 각 데이터 블락을 가르키는 인덱스 정보를 저장하여 HBase가 하둡 분산 파일 시스템 기반으로 저장되지만 소규모 임의 패턴의 데이터 입출력을 지원할 수 있도록 한다.

HBase는 하둡 환경에서 작동하기 때문에 데이터 용량 증가에 따른 확장성이 좋으며 하둡 에코 시스템과의 호환성이 좋다. 그리고 분산 환경에서 데이터의 일관성 보장해주는 장점을 가지고 있다. 하지만 하둡 파일 시스템을 기반으로 동작하기 때문에 HBase의 파일(HLog, Hfile)을 읽고 저장하는 데 추가적인 네트워크 비용과 디스크 비용이 든다. 이는 compaction을 비롯한 여러 디스크 관련 작업에서 성능상 문제점이 될 수 있다. 이러한 성능상 문제들을 완화하기 위하여 SSD와 같은 고속 저장 장치를 활용하는 등 HBase 구조 개선에 대한 많은 연구들이 진행되고 있다 [24].

4. 결 론

Key-value 데이터베이스는 캐시의 역할, 내장형 데이터베이스의 역할, 데이터베이스 자체로서의 역할을 할 수 있어 많은 이들이 사용하고 있다. 본 논문에서는 key-value 데이터베이스를 메모리와 디스크 관점의 접근으로 나누었다. In-memory key-value 데이터베이스는 캐시의 역할이 두드러지며, 빠른 데이터 입출력과 확장성이 주된 특징이다. 대부분의 in-memory key-value 데이터베이스는 데이터를 전부 메모리 내에 저장하여 빠른 처리를 가능하게 하는 대신 내구성 혹은 일관성을 어느 정도 희생하는 경우가 있다. 상용 데이터베이스 및 개발되고 있는 오픈소스 데이터베이스들은 로그와 복구를 지원 하지만 최적화 및 다양한 복구 방법들을 적용할 수 있을 것이다. 또한, 차세대 저장 장치가 발전함에 따라 in-memory key-value 데이터베이스를 더욱 향상 시킬 수 있을 것으로 기대된다. 지금까지 연구되었던 in-memory key-value 데이터베이스에 NVRAM을 적용하여 그에 맞게 내구성을 확보하는 방향의 연구가 유망할 것으로 판단된다.

디스크 기반의 key-value storage의 경우 LSM-트리를 활용한 연구들이 많다. LSM-트리는 로그 형식으로 데이터를 저장하기 때문에 쓰기 연산에서 좋은 효율을 보인다. LSM-트리는 쓰기 작업을 빠르게 하기 위해서 가비지 컬렉션(garbage collection) 및 밸런싱을 뒤로 미루는 방식을 택하는데, 이는 추후 비용이 큰 연산인 compaction을 수반한다. 여러 데이터베이스에서 compaction 알고리즘을 개선하기 위해 노력하고 있으며, 특히 페이스북 사의 RocksDB는 병렬 처리를 이용하여 compaction 방법을 개선 시켰다. 그러나 여전히 많은 데이터베이스들에서 compaction을 향상 시키려 노력하고 있으며 이 분야에 대한 연구와 더불어 디스크 기반의 key-value 데이터베이스에 차세대 저장 장치를 사용하는 방법 또한 더욱 연구되어야 할 것으로 생각된다.

References

[1] Gartner, Big data, http://www.gartner.com/it-glossary/big-data/, Dec. 2016.

[2] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky, *SILT: A memory-efficient, high-performance key-value store*, In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 1-13, Oct. 2011.

[3] Y. Mao, E. Kohler, and R. T. Morris, *Cache craftiness for fast multicore key-value storage,* Proceedings of the 7th ACM european conference on Computer Systems, pp. 183-196, Apr. 2012.

[4] B. Debnath, S. Sengupta, and J. Li, *SkimpyStash: RAM space skimpy key-value store on flash-based storage*, Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pp. 25-36, Jun. 2011.

[5] Redis, http://redis.io , Dec. 2016.

[6] Rocksdb, http://rocksdb.org , Dec. 2016.

[7] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, and R. E. Gruber, *Bigtable: A distributed storage system for structured data*, ACM Transactions on Computer Systems (TOCS), Vol. 26 No. 2 Article 4, Jun. 2008.

[8] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil, *The log-structured merge-tree (LSM-tree)*, Acta Informatica, Vol. 33, Issue

4, pp. 351-385, Jun. 1996.

[9] Hbase, https://hbase.apache.org , Dec. 2016.

[10] https://cassandra.apache.org , Dec. 2016.

[11] Dr. Eric A. Brewer, *Towards robust distributed systems*, PODC Keynote, Jul. 2000.

[12] S. Gilbert, and N. Lynch, *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*, ACM SIGACT News, Vol. 33 Issue 2, pp. 51-59, Jun. 2002.

[13] E. Brewe, *CAP twelve years later: How the rules have changed*, IEEE Computer, pp. 23-29, Jan. 2012

[14] IEEE Computer issue on the CAP Theorem , Daniel Abadi, http://dbmsmusings.blogspot.kr/2012/10/ieee-computer-issue-on-cap-theorem.html, Dec. 2016.

[15] Memcached, https://memcached.org/, Dec. 2016.

[16] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhou, and M. Rosenblum, *Fast crash recovery in RAMCloud,* In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 29-41, Oct. 2011.

[17] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, *MICA: a holistic approach to fast in-memory key-value storage*, In 11th USENIX Symposium on Networked Systems Design and Implementation, pp. 429-444, Apr. 2014.

[18] Data Plane Development Kit, http://www.intel.com/content/www/us/en/communications/data-plane-development-kit.html, Dec. 2016.

[19] S. Ghemawat, H. Gobioff, and S. T. Leung, *The google file system*, In ACM SIGOPS operating systems review , Vol. 37 No. 5, pp. 29-43, Oct. 2003.

[20] X. Wu, Y. Xu, Z. Shao, and S. Jiang, *LSM-trie: An LSM-tree-based ultra-large key-value store for small data items*, In 2015 USENIX Annual Technical Conference (USENIX ATC 15), pp. 71-82, Jul, 2015.

[21] L. Lu, T. S. Pillai, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, *WiscKey: separating keys from values in SSD-conscious storage*, In 14th USENIX Conference on File and Storage Technologies (FAST 16), pp. 133-148, Feb. 2016.

[22] M. Burrows, *The chubby lock service for loosely-coupled distributed systems*, In Proceedings of the 7th symposium on Operating systems design and implementation, pp. 335-350, Nov. 2006.

[23] Zookeeper, https://zookeeper.apache.org/, Dec. 2016.

Key-value 데이터베이스를 활용한 빅데이터 처리 플랫폼의 동향 분석

송경태, 박상현

연세대학교 컴퓨터과학과

요 약

최근의 데이터베이스 시스템은 전통적인 데이터베이스 처리 시스템과 다른 방법을 사용한다. 기존보다 많고 다양한 종류의 데이터를 저장해야 하기 때문에 최근의 데이터베이스 시스템은 분산 처리와 시스템의 확장성을 통해 빅데이터를 효율적으로 처리하는 것이 가장 중요해졌고, 이 중요성은 앞으로도 계속될 전망

이다. 이 때문에 최근 검색엔진, SNS 서비스와 같이 지속적으로 빅데이터를 다뤄야 하는 서비스에서 빅데이터 처리를 위한 데이터베이스들을 개발, 연구 되고 있다. 특히 key-value 데이터베이스 모델은 빠른 입출력과 확장성이 용이하다는 장점이 있어 빅데이터 처리에 알맞아 여러 어플리케이션의 시스템으로 사용되고 있다. 다른 빅데이터 데이터베이스 모델들도 key-value 데이터베이스 모델의 특징을 공유하고, key-value 데이터베이스 모델에서 쓰이는 많은 기법들을 차용하고 있기 때문에 빅데이터 연구에 있어서 key-value 데이터베이스에 관한 연구가 활발히 이루어지고 있다. 하지만 아직 최적화 관련 연구, 차세대 저장 장치를 적용한 연구 등 key-value 데이터베이스를 더욱 향상 시킬 수 있는 부분들이 있어 관련 내용에 대한 연구가 필요한 상황이다. 본 논문은 빅데이터 플랫폼 분야의 화두로 떠오른 key-value 데이터베이스에 대한 정의와 관련 기술들 및 연구되고 있는 플랫폼들을 열거하고, 기존 플랫폼들을 보완할 수 있는 방법론에 대해 제시한다.

감사의 글

본 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (NRF-2015R1A2A1A05001845)

Kyungtae Song received the B.S. degree in computer engineering from Yonsei University in 2015. He has been studying for M.S. degree since 2015 in Department of Computer Science, Yonsei University, Seoul, Korea. His current research interests include database, big data system, and flash memory.
E-mail address: skt8776@yonsei.ac.kr

Sanghyun Park received the B.S. and M.S. degrees in computer engineering from Seoul National University in 1989 and 1991, respectively. He received Ph.D. degree in the Department of Computer Science from University of California at LosAngeles (UCLA) in 2001. He is now a Professor in Department of Computer Science, Yonsei University, Seoul, Korea. His current research interests include database, data mining, bioinformatics, and flash memory.
E-mail address: sanghyun@yonsei.ac.kr