특집 03

Key-Value Store를 위한 저장 매체 기술 연구 동향

임승호 (한국외국어대학교)

목 차

1. 서 론
2. Key-Value Store 연구동향
3. Key-Value Store와 저장매체
4. 결 론

1. 서 론

Key-Value Store 또는 Key-Value Database는 key-value 쌍으로 이루어진 데이터 구조 배열 값을 저장, 검색, 관리하는 데이터 저장 기술 패러다임이다[1,2]. 일반적으로 key-value로 관리되는 데이터 구조는 key값 해당하는 dictionary 또는 hash값에 대해서 value를 저장하는 배열 형식의 구조를 가진다. Key-Value Store는 기존의 데이터베이스 시스템에 널리 사용되는 관계형 데이터베이스(RDB)와는 매우 다른 방식으로 동작한다. 관계형 데이터베이스는 잘 정의 된 데이터 유형 필드를 포함하는 테이블의 일련의 데이터베이스의 데이터 구조를 미리 정의하기 때문에 때로는 무겁고 성능에 제약이 따를 수 있다. 이와는 대조적으로, Key-Value Store는 각 레코드에 대한 다양한 필드를 가질 수 있는 하나의 불투명 컬렉션과 같은 데이터를 취급하는데, 이것은 상당한 유연성과 단순성을 제공한다.

이러한 유연성으로 인해서 Key-Value Store는 인터넷의 넘쳐나는 데이터를 저장하고 검색하고 관리하는 빅데이터 시스템의 저장 구조로 널리 사용되고 있다. Key-Value Store기반의 빅데이터 시스템은 high-throughput과 low-latency를 제공함으로써 이를 요구하는 다양한 많은 인터넷 응용 서비스에 활용되고 있다. Key-Value Store는 인터넷 기반의 빅데이터 시스템의 특성상 많은 데이터의 저장, 검색, 관리에 사용되기 때문에 key-value를 저장하는 저장장치 기술이 주요한 기술 중 하나라 볼 수 있으며, 고속의 검색과 고가용성을 보장하기 위해서 저장 매체를 어떤 방식으로 사용하느냐가 최근의 저장매체의 발달과 함께 주요 이슈라고 볼 수 있다.

Key-Value Store의 저장매체 관련 연구는 전통적인 저장 매체인 HDD 기반의 저장 매체 활용 기술에서부터, In-memory 기반의 key-value 관리 방법, 그리고 최근에 각광받고 있는 Flash Memory기반의 저장매체를 기반으로 성능 향상

연구를 비롯하여 다양한 연구 및 기술개발이 페이스북, 트위터, 구글과 같은 글로벌 인터넷 기업에서 진행 중에 있다[6,7,8,9,10,17,18]. 본 고에서는 Key-Value Store의 저장매체 기술에 대한 최근 연구 동향과 글로벌 기업들이 자사의 제품에 직접 적용한 기술 사례들을 살펴보고 향후 Key-Value Store 관련 저장 매체 적용 기술이 나아갈 방향에 대해서 모색해 보도록 한다.

2. Key-Value Store 연구동향

2.1 NoSQL의 Key-Value Store

Key-Value 데이터 모델은 빅데이터 시스템에서 널리 사용되는 NoSQL의 기본 데이터 모델이다. Key-Value Store 모델은 그 구성방식에 따라서 단순한 Key-Value 형태의 데이터 모델, Document Oriented 데이터 모델, Column Family Oriented 데이터 모델 등 다양한 형태의 데이터 모델로 확장될 수 있는데, 이들이 RDBMS형식의 데이터베이스와는 다른 주요한 특징은 단순성, 가용성, 확장성, 그리고 속도적인 측면이 있다. RDBMS의 주요 특징인 ACID(원자성, 일관성, 고립성, 지속적인 트랜잭션)을 지원하는 대신에 BASE(기본적 가용성, 소프트 상태, 결과적 일관성)을 제공하는 것을 주요 목적으로 한다. 이러한 측면으로 인해서 Key-Value Store는 서버 여러 대로 이루어진 클러스터 환경에서 주로 사용된다.

Key-Value Store는 여러 가지 형식을 가진 데이터 모델이 있지만, 기본적으로 Key-Value 쌍으로 이루어진 수많은 Row 값을 배열 형태로 기록하는 데이터 모델이다. 많은 Key-Value Store에서는 이러한 데이터 모델을 기반으로 인터넷 응용 시스템에서 넘쳐나는 데이터를 신속하게 저장하고, 저장된 대용량의 Key-Value의 검색의 가용성 및 신속성을 보장하기 위해서, 로그 형식의 데이터 기록 및 저장 방식과, Hash기반 테이블의 분산 저장, In-Memory Cache 구조를 가진다.

2.2 Key-Value Store 종류

Key-Value Store는 초창기 임베디드용으로 시작하여 SQL을 단순화한 Key-Value Store부터 최근의 빅데이터 및 대용량 인터넷 서비스를 위한 디스크 기반 분산 데이터베이스와 메모리 기반 분산 데이터베이스 등 다양한 종류의 Key-Value Store가 있다. 현재 빅데이터 시스템에서 가장 널리 사용되는 Key-Value Store의 대표적인 예로는 BerkeleyDB[3], HashCache[4], Memcached[5], BigTable[6], DynamoDB[7], Cassandra[8], LevelDB[9], HBase[10]등이 있으며, 여기서는 주요한 Key-Value Store를 소개한다.

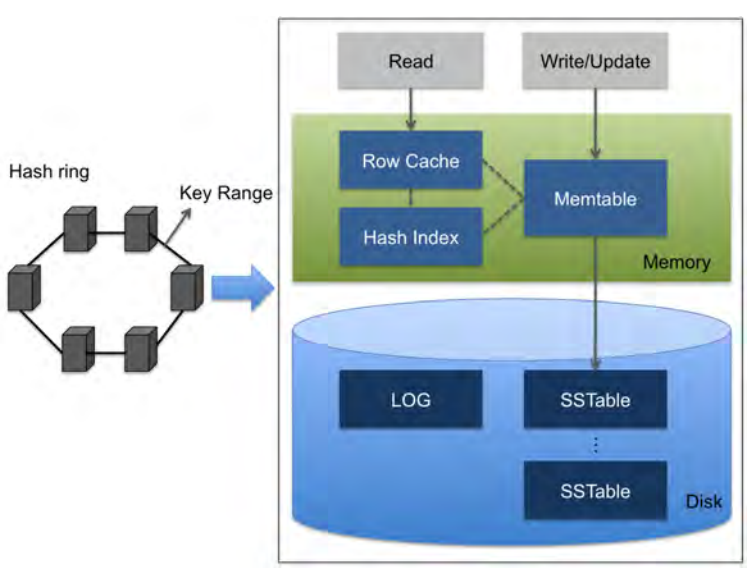
BerkeleyDB[3]는 1992년경 UCB에서 개발한 임베디드 Database이다. 유닉스 프로그램에서 보통 DB라 하면 이 버클리 DB를 말할정도로 대중적인 임베디드 데이터베이스로 알려졌으나, BerkeleyDB는 관계형 데이터베이스가 아니다. 임베디드용으로 활용되기 위해서 SQL의 복잡도와 무게를 제거하였으며, BerkeleyDB 저장 방식은 임의의 Key-Value의 쌍을 바이트 배열로 저장하는 방식이므로 초창기 Key-Value Store라고 할 수 있다. 그러나 Oracle Berkely DB 11g Release 2 버전부터 SQL과 SQLite API 를 지원하게 되었으며, 락이나 트랜잭션을 지원하며 Join나 Replication, In-Memory 데이터베이스도 지원하도록 진화되었다. BerkeleyDB에 SQL 분석기를 붙이고 인덱스 구성 가능하게 만든 것이 초기의 MySQL이다.

BitTable[6]은 구글 파일 시스템(Google File

System) 기반으로 개발된 고성능 Key-Value Store이다. BitTable은 분산 데이터베이스이며 Bigtable은 Row Key와 Column Key의 두 임의의 문자열 값과 timestamp값의 배열을 메모리에 테이블 형태의 tablet으로 기록한다. 각각의 테이블은 timestamp에 따라서 다수의 버전을 가지며 Google File System에 최적화 될 수 있도록 다수의 tablets으로 구성되며 200MB 크기를 가질 수 있다. tablet이 가득 차게 되면 GFS에 파일 형태의 로그를 기록한다. HBase[10]는 구글의 Bigtable의 기본 데이터모델을 기반으로 하여 Hadoop 그룹에서 개발한 HDFS(Hadoop Distributed File System)기반의 분산 저장 시스템을 활용한 Key-Value Store이다.

Cassandra[8]는 구글의 BigTable[6]과 아마존의 DynamoDB[7]를 기반으로 페이스북에서 개발한 분산 Key-Value Store이며 향후 Apache 오픈소스 프로젝트로 진행되었다. Apache Cassandra는 지속적인 가용성, 높은 확장성 및 성능, 강력한 보안 및 운영 단순성을 제공함으로써 현재 인터넷 빅데이터 시스템에서 가장 널리 사용되는 Key-Value Store 관리 시스템이라고 할 수 있다. 그림 1은 Cassandra Key-Value Store의 전형적인 구조를 도식화한 것이다. Cassandra의 데이터 모델은 구글의 BigTable기반으로 하였으며, 메모리상에 Memtable 구조의 Key-Value 어레이 값을 유지하며, Memtable의 Log형식의 값을 디스크상에 SSTable 구조로 저장한다. Cassandra의 분산 시스템의 구조는 아마존의 DynamoDB를 따르는데, Distributed Hash Table형식의 Hash Ring구조의 분산 시스템을 구성한다.

Memcached[5]는 데이터베이스 기반의 웹 서비스 시스템에서 스토리지에 저장되어 있는 데이터베이스 소스데이터를 읽는 부하를 줄이기 위해서 데이터베이스 오브젝트를 RAM에 Caching하여 관리할 수 있도록 만들어진 분산 메모리 캐싱 시스템으로 개발되었다. Memcached는 In-memory 분산DB를 관리하기 위해서 client-server 구조를 사용한다. 서버는 Key-Value



(그림 1) Cassandra 데이터 모델

관계에 대한 연관 배열을 유지하고 관리하며, Client는 이 배열의 값을 채우고 key에 대한 query를 담당한다. Memcached 자체는 메모리 상에서만 Key-Value값을 유지하기 때문에 Key-Value값을 저장하는 서버에서 메모리가 다 차게 되면 이전의 값을 버리는 방식으로 메모리에 최신 Key-Value값을 유지한다. Memcached의 단점을 보완 및 확장하여 디스크 기반의 저장까지 가능하게 만든 것이 MemcacheDB인데, MemcachedDB의 디스크 부분 관리는 BerkeleyDB [3] 구조를 사용한다.

3. Key-Value Store와 저장매체

3.1 Disk-Optimized Key-Value Store와 In-Memory Caching

Key-Value Store기반의 빅데이터 시스템에서도 실시간으로 급격히 생성되는 데이터의 저장 관리 데이터 규모가 테라(Tera)바이트를 넘어서 페타(Peta)바이트 단위로 돌입한 지 오래이다. HDD 기반의 디스크 저장장치는 값싼 가격 대비 대용량의 장점이 있지만, 기계적인 플래터와 헤더를 사용하는 구조적 한계로 인해서 임의(Random) 읽기/쓰기가 순차(Sequential) 읽기/쓰기에 비해서 급격히 느리다는 단점이 있다. Key-Value Store에서는 이러한 단점을 보완하기 위해서 읽기와 쓰기를 구분하여 다른 형식의 연산을 가지도록 구현하고 있다. 이는 GFS, HDFS, HFS등 기존의 대용량 분산 파일 시스템과 같은 방식인데, 기본적으로 기록을 위해서 생성된 요청연산(Requests)는 로그 형식으로 데이터를 순차적으로 기록하는 방식을 사용하며, 읽기를 위해서 생성된 요청연산(Requests)는 메모리 캐쉬를 이용하는 방식이다. 그렇기 때문에 기본적인 저장장치의 용량이 테라(Tera)~페타(Peta)바이트의 용량을 가지며, Key-Store Caching을 위한 메모리는 수십기가~수백기가바이트의 용량을 가진다. 초창기 Bigtable, Cassandra, MemcacheDB, MongoDB등이 이러한 구조로 동작하는 Key-Value Store라 할 수 있다.

Cassandra의 예를 들어보면, Cassandra의 모든 쓰기(Write/Update) 요청 연산은 대용량 메모리의 Key-Value 데이터배열인 Memtable에 순차적으로 기록된다. Memtable에 지정된 용량에 도달했을 경우, Memtable은 디스크에 Log 데이터 형식으로 저장되며 이를 SSTable에 기록되는 방식이다. Memtable의 크기가 수 MB에서 수십 MB의 크기로 디스크에 Log 형식으로 기록되기 때문에 디스크에 순차적인 쓰기 연산을 수행할 수 있는 것이다. 반면 읽기 요청 연산의 경우 메모리상의 Row Cache와 Memtable을 검색하여 해당 데이터를 존재하는 경우 읽기 요청 연산을 반환한다. 만약 메모리에 해당 요청 데이터가 없을 경우 디스크 저장장치의 SSTable을 검색하여 원하는 Key에 대한 Row를 구성하여 반환하는데, 이 때 다수의 SSTable에 존재하는 버전을 검색하여 Row를 구성해야 하기 때문에 다수의 랜덤 읽기가 발생할 수 있다.

Key-Value Store 저장 시스템에서 읽기/쓰기 연산의 비대칭으로 인해서 발생하는 또다른의 문제점은 업데이트되는 데이터의 Log 기록 연산에 의해서 발생하는 가비지(garbage)처리를 위한 compaction 또는 cleaning 작업이다. 특히 Key-Value Store는 row당 timestamp버전을 관리하기 때문에 이러한 compaction작업으로 인해서 내부적으로 처리해주는 쓰기 연산 오버헤드가 실제적인 외부 요청에 의한 쓰기 연산에 비해서 수배~수십배의 쓰기 오버헤드가 발생하는 문제점이 있다.

3.2 Flash Disk의 Evolution

플래시 메모리, 특히 NAND 플래시 메모리의 급격한 대용량화는 저장장치 최근 수년간 저장장치 시장의 패러다임 뿐만 아니라 컴퓨터 시스템 구조의 급격한 변화를 가져오고 있으며, 낸드 플래시 메모리 기반의 저장장치인 SSD(Solid State Disk)의 최근의 급격한 성능 향상과 대용량화는 전통적인 HDD를 대체하기에 이르고 있다. 낸드 플래시 메모리의 물리적인 특징으로 인해서 SSD의 순차 읽기/쓰기 연산속도 뿐만 아니라 임의 읽기/쓰기 속도가 HDD에 비해서 급격히 개선되고 있다.

낸드 플래시 메모리는 플로팅 게이트를 사용하는 반도체소자로써. 낸드 플래시 메모리의 기본적인 읽기쓰기 연산의 크기는 페이지(page)라는 단위로써 크기가 4KB~8KB크기이다. 쓰기 페이지에 쓰기 연산을 수행하기 위해서는 지우기(Erase)연산이 먼저 수반되어야 하는데, 그 크기는 블록(Block)이라는 단위이며 64~256개의 페이지그룹이다. 또한 블록 지우기 연산의 횟수는 블록당 제한이 있는데 집적도를 높이여서 용량을 점점 증대시킬수록 이 횟수는 점점 줄어든다. 인해서 낸드 플래시 기반의 SSD등 저장장치 내부에 플래시 변환 계층(Flash Translation Layer)라는 소프트웨어 또는 펌웨어가 물리적인 제약을 보완해 준다. 플래시 변환 계층의 주요한 기능으로는 호스트 논리 주소와 낸드 플래시 메모리의 물리주소 변환 관리, 웨어 레벨링 관리, 가비지 콜렉션등의 작업을 수행한다.

SSD와 같은 Flash Disk는 최근의 급격한 대용량화와 가격 하락으로 인해서 대용량 서버 시스템에서 그 점유율을 점점 더 높여가고 있으며, 서버의 성능하락의 주요 요인이었던 저장장치 IO의 한계를 극복할 수 있는 여건을 마련해주고 있다. 특히 임의 읽기/쓰기 연산 속도의 급격한 향상으로 인해서 기존 HDD가 가지고 있는 문제를 해결할 수 있는 대안으로 각광받고 있다. Key-Value Store 시스템에 플래시 디스크 저장장치를 활용하는 방법은 기존의 Memory-Disk 저장장치 기반에서 Flash Disk를 어떤 식으로 조합하느냐의 문제일 수 있다. Flash Disk 저장장치를 Key-Value Store에 적용한 기술 개발에는 IBM의 CaSSanDra[16], 페이스북의 McDipper [17], 트위터의 Fatcache[18], 그리고 NVMKV [20]등이 있다.

SSD boosted CaSSanDra[16]는 기존의 Cassandra Key-Value Store 시스템의 Memory -HDD사이에 SSD를 추가하여 Hybrid HDD/SSD를 활용한 Memory-SSD-HDD 저장장치 구조를 제안하고, SSD를 Second Level Cache로 활용한 연구를 진행하였다. SSD boosted CaSSanDra는 Key-Value Store에 저장되는 각 row의 column명들이 거의 변하지 않음에도 불구하고 SSTable에 저장되는 것이 중복되는 내용이며, HDD의 Ramdom I/O의 성능이 안좋은 영향을 미치는 것에 착안하여, Key-Value Store에 저장되는 Column 명들을 따로 Schema Catalogue라고 하는 것으로 분리하여 SSD에 Caching하는 Key-Value Store Architecture를 제안하였다.

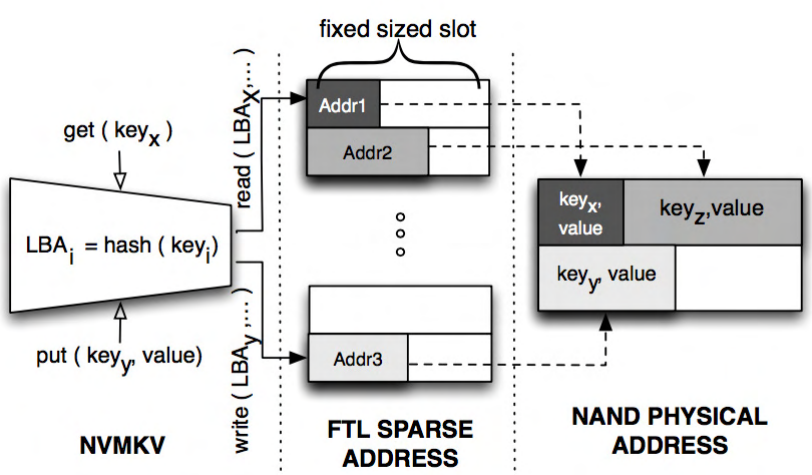
MCDipper[17]는 페이스북에서 In-memory Key-Value Store 캐시인 Memcache를 SSD 디바이스 기반으로 재설계 및 구현한 Key-Value Store이다 Memcache를 위해서 수십기가~수백기가바이트의 RAM이 필요한데, 이는 성능 향상에는 좋지만 고비용의 시스템 구성이 필요하다. 페이스북에서는 RAM보다는 값싸지만 초당 수십만 IOPS(IO Operations Per Second)의 성능을 보장하는 SSD로 RAM을 대체할 경우 가격효율

적인 Key-Value Store 캐쉬를 구현할 수 있다고 보았다. Memcache를 SSD기반으로 재구성한 프로젝트는 트위터에서 진행된 FatCache[18]도 있다. 메모리 시스템에 최적화된 Memcache 구현을 SSD기반으로 안정적으로 동작하도록 구현하는 것이 쉬운 일은 아니지만, MCDipper와 Fatcache는 SSD의 물리적인 특징을 설계시 고려하지 않았다는 점에서 한계점이 있을 수 있다.

FusionIO와 관련 연구그룹에서는 NVMKV [20] 연구결과를 발표하였는데, 이는 SSD의 낸드 플래시 메모리의 특성 및 FTL의 기능을 Key-Value Store의 API로 활용하여 Key-Value Store의 저장장치 성능을 향상시키기 위한 시도였다. 그림 2는 NVMKV의 Key-Value Hash Model을 도식화한 것으로써, NVMKV는 FTL의 주요 특징인 다이나믹 매핑 관리, 로그 구조의 데이터 기록, 트랜잭셔널 데이터 업데이트, 씬 프로비저닝(thin provisioning) 등이 Key-Value Store에서 사용되는 저장장치 관리 기법과의 유사점이 있다고 보고, FTL의 API를 유저레벨 Key-Value Store의 API로 확장하여 Key-Value Store관리에 사용할 수 있도록 라이브러리 형태로 제공하도록 구현하였다. 이를 통해서 Flash 최적화된 Key-Value Store 구성이 가능하다.

최근의 Open-Channel SSD[21]를 기반으로 한 SSD-transparent Key-Value Store 설계 연구는, NVMKV에서 더 나아가서 플래시 디스크의 낸드 플래시 메모리 관리 부분까지 Key-Value Store에서 운영 및 관리하고자 하는 시도이다. Open-Channel SSD는 기존의 SSD와는 다르게 FTL과 같은 플래시 소프트웨어를 운영체제 내에서 직접 관리하는 SSD 구조를 말한다. Key-Value Store에서 Open-Channel SSD 기반의 저장장치 운영 및 관리를 하게 될 경우의 가장 큰 이점은 기존의 SSD 기반 Open-Channel에서 발생하는 가장 큰 문제점인 Write Amplification을 줄여줄 수 있으므로 쓰기 최적화된 Key-Value Store 구성이 가능하다는 장점이 있다.

이상과 같이 플래시 디스크 기반의 Key-Value Store는 지난 수년간 SSD의 급속한 발전과 더불어 함께 진행되어 오고 있으며, 기존의 HDD기반의 저장장치 구조에서 탈피하여, Memory, SSD,



(그림 2) NVMKV의 Key-Value Hash Model[20]

HDD를 다양한 측면에서 적용하여 기술 진화를 이룩하였다. 향후에도 이러한 저장장치 관리 및 운영의 연구를 통해서 Key-Value Store 및 빅데이터 시스템의 진화는 계속 될 것으로 보인다.

4. 결 론

Key-Value Store는 NoSQL 데이터베이스의 데이터 저장 방식으로써 기존의 관계형 데이터베이스에 비해서 비정형화되어 단순성, 가용성, 확장성, 그리고 처리 속도에서 성능이 상대적으로 우수하여 빅데이터 시스템의 데이터 저장 및 검색에 널리 사용되고 있다. Key-Value Store의 대표적인 예로는 구글의 Bigtable, LevelDB등이 구글 클러스터 서버와 구글 파일 시스템의 빅데이터 저장 구조로 사용되며, 페이스북의 Cassandra, 아마존의 Dynamo, Hadoop의 HBase등이 있다.

이러한 Key-Value Store는 지속적으로 늘어나는 데이터의 저장 및 검색의 효율성을 위해서 저장장치 구조 및 동작은 Key-Value의 어레이를 In-Memory Cache에 기록한 후, 일정 크기의 로그를 디스크에 기록하는 쓰기/업데이트 연산과, In-Memory Cache 검색후, Cache에 존재하지 않는 경우 다수의 임의 읽기를 통해서 Key-Value Row를 구성해서 반환하는 읽기 연산으로 구성하며, Key-Value에 대한 분산 Hash를 적용하여 분산 데이터베이스 시스템을 구성한다.

Memory-HDD기반의 Key-Value Store는 구조상 빅데이터 저장장치 시스템의 성능 및 확장이 비효율적인 측면이 많이 나타났으며, 이를 극복하기 위해서 최근의 플래시 메모리 기반의 SSD가 Key-Value Store의 저장장치 시스템에 많이 도입되어 가격 및 성능 향상의 역할을 하고 있다. 향후 SSD는 집적도 향상으로 비용이 더욱 더 저렴해질 것이며, NVMe PCIe[22] 인터페이스와 같은 호스트 인터페이스의 진화, Open Channel SSD등 플래시 소프트웨어의 다변화로 인해서 더욱더 Key-Value Store를 비롯한 빅데이터 및 인터넷 서버 시스템에서의 활용이 증대될 것으로 기대된다.

참 고 문 헌

[1] J.Manyika, M.Chui, B.Brown, J.Bughin, R.Dobbs, C.Roxburgh, and A. H. Byers, “Big data: The Next Frontier for Innovation, Competition, and Productivity,” McKinsey Global Institute, Tech. Rep., 2011.

[2] T. Rabl, M. Sadoghi, H.-A. Jacobsen, S. Gómez-Villamor, V. Munte´s- Mulero, and S. Mankowskii, “Solving Big Data Challenges for Enter- prise Application Performance Management,” PVLDB, 2012.

[3] M. Olson, K. Bostic, and M. Seltzer. Berkeley db. In Proc. of USENIX Annual Technical Conference, 1999

[4] A. Badam, K. Park, V. Pai, and L. Peterson. Hashcache: cache storage for the next billion. In Proc. of USENIX Networked System Design and Implementation, 2009.

[5] memcached - a distributed memory object caching system https://memcached.org/

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” in OSDI, 2006.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon’s Highly Available Key-Value Store,” in SOSP, 2007.

[8] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” SIGOPS Review, 2010.
[9] LevelDB – a fast and lightweight key/value database library by Google. http://code.google.com/p/leveldb/.
[10] Apache HBase. http://hbase.apache.org/.
[11] M. Sadoghi, K. A. Ross, M. Canim, and B. Bhattacharjee, “Making updates disk-I/O friendly using SSDs,” PVLDB’13.
[12] R. Cartell, “Scalable SQL and NoSQL data stores,” SIGMOD Record, 2010.
[13] M. Cornwell, “Anatomy of a solid-state drive,” Communications of the ACM, 2012.
[14] L. Bouganim, B. r Jnsson, and P. Bonnet, “uFLIP: Understanding Flash IO Patterns,” in CIDR ’09: Fourth Biennial Conference on Innovative Data Systems Research.
[15] G. Graefe, “The Five-Minute Rule 20 Years Later: and How Flash Memory Changes the Rules ,” Communications of the ACM, 2009.
[16] Prashanth Menon, Tilmann Rabl, Mohammad Sadoghi, Hans-Arno Jacobsen, “CaSSanDra: An SSD Boosted Key-Value Store”, IEEE ICDE Conference, 2014.
[17] MCDipper, https://www.facebook.com/notes/facebook-engineering/mcdipper-a-key-value-cache-for-flash-storage/10151347090423920
[18] Fatcache, https://github.com/twitter/fatcache
[19] LIM, H., FAN, B., ANDERSON, D. G., AND KAMINSKY, M. SILT: A Memory-Efficient, High-Performance Key-Value Store. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP’11) (Cascais, Portugal, October 23-26 2011)
[20] MA ´RMOL, L., SUNDARARAMAN, S., TALAGALA, N., AND RAN- GASWAMI, R. NVMKV: A Scalable and Lightweight, FTL-aware Key-Value Store. In Prceedings of USENIX ATC’15 (2015).
[21] Open-Channel Solid State Drives, http://openchannelssd.readthedocs.io/en/latest/
[22] NVMe (Non-Volatile Memory Express), http://www.nvmexpress.org
[23] YANG, J., PLASSON, N., GILLIS, G., TALAGALA, N., AND SUN- DARARAMAN, S. Don’t Stack Your Log on My Log. In Proceedings of the 2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW’14) (Broomfield, CO, October 5 2014).
[24] ZHANG, Y., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. De-indirection for Flash-based SSDs with Nameless Writes. In Proceedings of FAST’12 (2012).
[25] ZHANG, Y., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. Removing the Costs and Retaining the Benefits of Flash-Based SSD Virtualization with FSDV. In Proceedings of MSST’15 (Santa Clara, CA, June 2015).
[26] L. Grupp, J. Davis, and S. Swanson. The bleak future of nand flash memory. In Proc. of USENIX FAST, 2012.

임 승 호

이메일: slim@hufs.ac.kr

- 2001년 한국과학기술원 전기 및 전자공학과 (학사)
- 2003년 한국과학기술원 전기 및 전자공학과 (석사)
- 2008년 한국과학기술원 전기 및 전자공학과 (박사)
- 2008년~2010년 (주)삼성전자 메모리사업부 책임연구원
- 2010년~현재 한국외국어대학교 컴퓨터.전자시스템공학부 부교수
- 관심분야 : 운영체제, 임베디드 시스템, 파일 시스템, 플래시 메모리, 빅데이터 저장장치