The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 22, No. 4, pp.17-22, Aug. 31, 2022. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2022.22.4.17
JIIBC 2022-4-3

키밸류 저장소 성능 제어를 위한 삭제 키 분리 LSM-Tree

A Tombstone Filtered LSM-Tree for Stable Performance of KVS

이은지*

Eunji Lee*

요 약 최근 웹 서비스의 확산과 함께 데이터의 형태는 더욱 다양해지고 있다. 이미지, 동영상, 텍스트 등 데이터를 저장하는 형태 뿐 아니라 해당 데이터를 표현하는 속성 및 메타데이터 등도 개수 및 형태가 데이터 별로 상이하다. 이러한 비정형 데이터를 효율적으로 처리하기 위해 키밸류 스토어(Key-Value Store)의 사용이 확산되고 있다. LSM-Tree(Log Structured Merge Tree)는 다양한 상용 키밸류 스토어의 핵심 자료구조이다. LSM-Tree 는 모든 쓰기 및 삭제 연산을 로그 방식으로 기록함으로써 소량의 쓰기에 높은 성능을 제공하도록 최적화 되어 있다. 그러나 최근 유효성 만료 데이터의 대용량 삭제 연산이 LSM-Tree에 특수 키밸류 데이터로 삽입됨에 따라 사용자 요청의 지연시간 및 처리속도가 저하된다는 문제점이 있다. 본 논문은 기존 LSM-Tree의 장점을 모두 유지하면서도 삭제된 키를 주요 트리 구조에서 분리하여 상기 문제를 해결하는 Filtered LSM-Tree (FLSM-Tree)를 제안한다. 제안하는 기법은 상용 키밸류 저장소인 LevelDB에 구현되었으며 성능 평가에서 읽기 성능이 최대 47% 향상됨을 보인다.

Abstract With the spread of web services, data types are becoming more diversified. In addition to the form of storing data such as images, videos, and texts, the number and form of properties and metadata expressing the data are different for each data. In order to efficiently process such unstructured data, a key-value store is widely used for state-of-the-art applications. LSM-Tree (Log Structured Merge Tree) is the core data structure of various commercial key-value stores. LSM-Tree is optimized to provide high performance for small writes by recording all write and delete operations in a log manner. However, there is a problem in that the delay time and processing speed of user requests are lowered as batches of deletion operations for expired data are inserted into the LSM-Tree as special key-value data. This paper presents a Filtered LSM-Tree (FLSM-Tree) that solves the above problem by separating the deleted key from the main tree structure while maintaining all the advantages of the existing LSM-Tree. The proposed method is implemented in LevelDB, a commercial key-value store and it shows that the read performance is improved by up to 47% in performance evaluation.

Key Words : Key-value Store, LSM-Tree, NoSQL System

*정회원, 숭실대학교 AI융합학부 (교신저자)
접수일자 2022년 7월 18일, 수정완료 2022년 7월 31일
게재확정일자 2022년 8월 5일

Received: 18 July, 2022 / Revised: 31 July, 2022 /
Accepted: 5 August, 2022
*Corresponding Author: ejlee@ssu.ac.kr
School of AI Convergence, Soongsil University, Korea

Ⅰ. 서 론

NoSQL 데이터베이스는 최근 웹 응용을 중심으로 생산되는 다양한 비정형 데이터를 저장하는 용도로 널리 활용되고 있다[1]. 다수의 데이터가 고정된 일련의 속성을 지니는 기존 SQL 기반 데이터베이스[2][3]와 달리 NoSQL 시스템은 키(Key)와 밸류(Value)라는 단순한 정보의 쌍으로 데이터를 저장한다. 연관된 데이터는 동일한 prefix를 사용하여 키를 표현함으로써 속성이 다양한 데이터를 유연하게 저장할 수 있다는 장점이 있다. 이에 NoSQL 시스템은 사실상 현재 저장 시스템의 표준이 되어가고 있으며 웹 서비스를 제공하는 많은 기업들에서 다양한 NoSQL 솔루션을 개발한 바 있다[4][5].

한편, 데이터 저장 시스템은 최근 데이터 규모가 거대해짐에 따라 빠르게 시스템이 고도화되고 있다. 단일 데이터베이스로 데이터를 처리하던 과거와 달리 최근 데이터 저장 시스템은 분산 환경에서 다양한 컴포넌트를 효율적으로 통합하여 운용하는 방식이 일반적이다. 예를 들어 대표적인 분산 NoSQL 시스템인 Ceph는 내부 데이터의 메타데이터를 관리하기 위해 LevelDB나 RocksDB와 같은 임베디드 KVS(Key-Value Store)를 사용한다[6]. 성능(Performance), 규모(Scalability), 신뢰성(Reliability), 그리고 개발 효율성(Development Velocity) 등 다차원적인 요구사항을 동시에 만족시키기 위해서는 단일 시스템을 넘어서는 수준의 저장 시스템이 필요하기 때문이다.

본 논문은 이러한 NoSQL 데이터 저장 시스템 구조에서는 대량의 삭제 연산이 빈번하게 발생하며 이로 인한 성능 저하가 초래된다는 것을 살펴보고, 이를 해결하기 위한 방안을 제안한다. NoSQL 시스템이 널리 활용되는 웹 서비스 환경에서는 수집 또는 생성된 데이터를 일정 시간 보관하고 있다가 특정 시간이 만료되면 시스템에서 삭제하는 동작을 수행하는 경우가 많다. 예를 들어 사용자 정보나 거래 내역 등 모두 위와 같은 방식으로 관리된다. 이로 인해 웹 서비스 환경에서는 특정 시점에 유효성이 만료되는 데이터에 대한 대량의 삭제 연산이 발생하는 경우가 많다. 문제는 현재 널리 사용되는 NoSQL 시스템이 삭제 연산을 효율적으로 처리하지 못해 성능이 저하되는 현상이 발생한다는 것이다.

현재 NoSQL 시스템에서 가장 널리 활용되는 자료구조 중 하나는 LSM-tree (Log-structured Merge Tree)이다. 데이터를 정렬된 다중 계층의 로그로 유지하는데 최근에 접근된 데이터가 상위 계층 로그에 위치한다. 데이터가 업데이트 될 때 기존의 데이터를 덮어쓰지 않고 새로운 데이터를 삽입하고, 읽기 시 가장 최근에 업데이트 된 데이터를 리턴함으로써 업데이트를 처리한다. 이전 데이터는 계층 간의 병합이 발생하는 컴팩션(Compaction) 과정에서 중복된 키를 제거함으로써 삭제된다. 위와 같은 방식은 쓰기 시 원본 데이터를 찾아 제거하는 비용을 부과하지 않기 때문에 작은 크기의 쓰기가 빈번한 웹 응용에 적합하다.

LSM-tree 에서는 삭제 연산 역시 tombstone 으로 정의된 특수 밸류를 해당 키와 함께 삽입함으로써 처리된다. 이에 유효기간 만료와 같은 이벤트로 대량의 삭제 연산이 발생하면 LSM-tree 내부에는 대량의 tombstone 이 삽입되는 결과가 초래된다. 이것은 tombstone 데이터로 LSM-tree를 불필요하게 팽창시키는 한편, 접근속도가 빠른 LSM-tree의 상위 계층을 재접근 가능성이 낮은 tombstone 데이터로 채워 읽기 연산의 성능을 크게 저하시킬 수 있다. 또한 동시 삽입을 지원하는 LSM-tree의 경우 비동기적 삭제 연산이 동기적 쓰기 연산의 지연시간을 증가시키는 결과도 가져올 수 있다. tombstone 으로 인한 성능 저하 및 공간 낭비는 해당 데이터가 LSM-tree 에 남아있는 동일 키와 병합되어 삭제될 때 해소되는 것으로 상당한 시간이 소요될 수 있다.

상기 문제를 해결하기 위해 본 논문에서는 삭제 시 tombstone을 LSM-tree에 삽입하는 대신 cemetery 테이블을 별도로 유지하여 삭제된 키에 대한 정보를 기존 데이터로부터 분리하여 관리하는 Filtered LSM-tree (FLSM-tree)를 제안한다. FLSM-tree 는 삭제 시 해당 키를 cemetery에 삽입한다. 읽기 및 쓰기 연산이 발생 시에는 cemetery를 적절히 업데이트 하거나 체크하여 가장 최근의 데이터 상태를 사용자에게 보여줄 수 있도록 한다. 이를 통해 대량의 tombstone 발생으로 인한 KVS 성능 저하를 최소화 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 LSM-tree의 구조 및 연산에 대해 설명한다. 3장에서는 제안하는 FLSM-tree의 동작에 대해 설명하고, 4장에서는 실험결과를 제시한다. 5장에서는 FLSM-tree의 효과 및 한계점에 대해 논의하고 결론을 맺는다.

Ⅱ. LSM-tree 기반 KVS

LSM-tree 는 Log-structured Merge Tree 의 약자로 데이터의 쓰기를 로그에 순차적으로 덧붙이다가 로그

가 일정 크기 이상이 되면 병합을 수행하여 새로운 로그로 만드는 형태의 자료구조이다[7]. 그림 1의 하부는 LSM-tree의 전체적인 구조를 보여준다. 가장 상단에는 memtable 이 존재하는데 이는 DRAM 영역에 유지되어 쓰기 버퍼로 사용된다. memtable 이 일정 크기 이상으로 증가하면 저장장치에 memtable을 저장하는데 해당 계층이 Level 0이다. LSM-Tree에서 데이터는 모두 정렬된 상태로 저장된다. memtable 에서는 통상 Skip List 자료구조를 이용해 memtable 에 속한 데이터를 정렬된 상태로 유지하고 저장장치에 데이터가 저장될 때에는 SST (Sorted String Table) 형식의 파일로 저장된다. Level 0는 memtable 의 내용을 그대로 저장하기 때문에 Level 0에 있는 파일들 간에 글로벌 정렬은 성립되지 않는다. 그러나 Level 1부터는 SST file 간에 오버랩이 존재하지 않으며 글로벌 정렬 상태가 유지된다.

KVS의 가장 기본이 되는 연산은 Put과 Get 이다. Put은 새로운 데이터를 저장하는 연산인데 데이터의 삽입, 업데이트, 삭제 모두 Put 연산으로 처리한다. Get 연산 발생 시에는 요청된 키에 대응하는 값들 중 가장 최신의 값을 반환해야 한다. LSM-Tree 에서는 모든 데이터의 변경을 삽입 방식으로 처리하기 때문에 LSM-Tree 에는 동일 키에 대한 다수의 값이 존재할 수 있다. 이 중 최신 버전을 찾기 위해서 상위 계층부터 탐색을 한다.

다중 계층 탐색 과정의 오버헤드를 줄이기 위해 최근 버전의 KVS에서는 블룸 필터(Bloom Filter) 등을 이용해 SST file 내에 요청된 키의 존재 여부를 빠르게 확인하고 여러 계층을 병렬적으로 검색하는 방식을 사용하기도 한다. 동시에 여러 버전의 데이터가 검색되면 타임스탬프(Timestamp)를 비교하여 가장 최신의 데이터를 반환한다. 이를 위해 통상 KVS는 사용자 전달한 키와 타임스탬프를 접합하여 내부키로 활용한다.

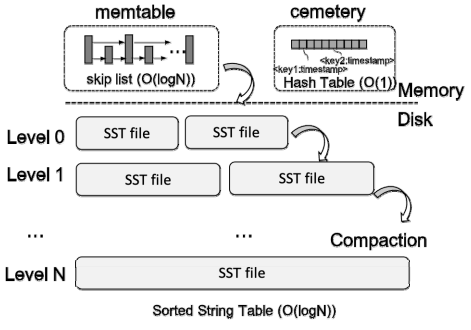


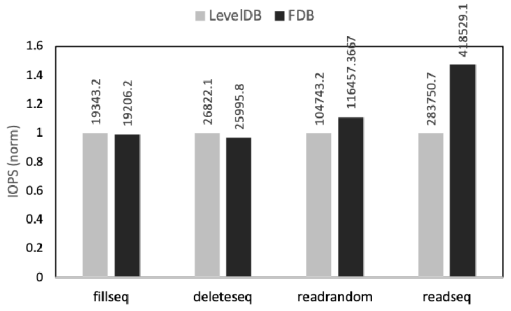
그림 1. FLSM-tree 구조
Fig. 1. FLSM-tree architecture.

III. FLSM-Tree 구조

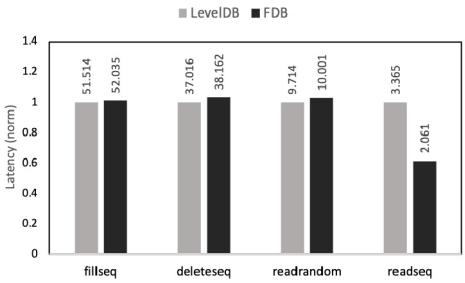
본 논문에서 제안하는 FLSM-Tree 는 데이터의 유효성 만료 및 임시 데이터 삭제 등과 같은 대량의 삭제 연산이 주요 자료구조인 LSM-Tree 에 유입되는 것을 차단하는 데에 목표를 둔다. 재접근 가능성이 없는 대량의 tombstsone 데이터의 삽입은 데이터 접근 워크로드의 시간 지역성을 활용해 고성능을 제공하는 LSM-Tree 의 기본 성질을 훼손하고, 심각한 공간 낭비를 초래할 수 있기 때문이다. FLSM-Tree 는 cemetery 테이블을 사용해 삭제된 키를 별도로 유지하는 기법을 제안한다.

cemetery 는 삭제된 키를 보관하는 것이 명확하기 때문에 별도로 tombstone 표시를 위한 밸류 공간을 할당하지 않고 키만 유지한다. cemetery 가 존재하는 경우 Put 연산 시에는 업데이트 된 키가 cemetery 에 있는지 확인하고 삭제해 주고, Get 연산 시에는 cemetery 에 요청된 키가 있는지 먼저 확인한 후 부재 시에만 LSM-Tree 내부를 탐색하도록 하였다. 또한 참조 연산을 빠르게 하기 위해 해쉬 테이블로 구현하였다. 동시쓰기를 허용하는 KVS의 경우 성능저하를 일으키지 않으려면 cemetery를 업데이트가 Non-blocking 방식으로 수행될 수 있어야 한다. 뮤텍스 락(Mutex Lock) 등을 사용해 임계 영역(Critical Section)에 단일 쓰레드 진입만을 보장해야 하는 경우에는 성능이 심각하게 저하될 수 있기 때문에다. cemetery는 해쉬 테이블로 구현되어 있어 임계 영역이 단일 포인터 업데이트로 축소될 수 있으며 이는 CAS (Compare And Swap) 등과 같은 명령어를 이용해 락프리(Lock-free) 방식으로 수정이 가능하다.

또한 읽기/쓰기 간의 동시 실행성도 제공해야 하는데 이는 MVCC (Multi Version Concurrency Control) 기법을 이용해 지원할 수 있다. MVCC는 쓰기 도중 읽기 요청이 블로킹되는 것을 방지하고자 타임스탬프를 이용해 동시적 읽기 쓰기를 지원한다. 데이터의 새로운 버전이 쓰여지는 동안 이전 버전의 데이터를 읽어가도록 하는 것이다. 이를 위해 KVS는 현재 읽기 가능한 (Visible) 가장 최신의 스냅샷 버전을 내부적으로 관리하고 있으며, 읽기 요청 시 해당 버전 이전 데이터를 읽어가도록 한다. 쓰기 요청 시 모든 요청은 순차적으로 증가하는 타임스탬프가 부여되며, 읽기 가능한 스냅샷 버전은 일정 그룹의 쓰기 요청이 memtable 과 cemetery를 모두 수정한 후 가장 늦은 타임스탬프로 업데이트 된다. 따라서 cemetery 자료구조가 추가되더라도 MVCC 기능을 저해하지 않는다. 예를 들어 임의의 키가 삭제되어 cemetery



(a) IOPS



(b) Latency

그림 2. FLSM-tree 기반 KVS 성능
Fig. 2. Performance of KVS with FLSM-tree

에 삽입되는 도중 읽기 쓰레드가 해당 키를 읽어갔다고 해보자. 해쉬 테이블의 경우 CAS로 원자적 업데이트가 가능하기 때문에 읽혀진 데이터는 무결성(Integrity)가 보장된다. 따라서 읽힌 데이터의 타임 스탬프를 현재 읽기 가능한 버전과 비교하여 읽기 가능한 버전보다 나중에 쓰여진 데이터라면 cemetery 에 데이터가 없다고 반환하고 LSM-Tree 에 접근하여 데이터를 찾을 수 있다.

III. 실험결과

제안하는 FLSM-Tree의 성능평가를 위해 구글에서 개발한 KVS인 LevelDB[2]에서 기존 LSM-Tree 대신 FLSM-Tree를 사용하도록 확장 구현하였다. 성능평가는 LevelDB에서 제공하는 벤치마크인 Db_bench를 사용했다. 초기에 4개의 쓰레드를 사용해 총 2백만 개의 Put을 발생시켜 순차 쓰기(fillseq)를 수행하였고, 이후 각각의 쓰레드가 순차 삭제(deleteseq), 임의 읽기(readrandom), 순차 읽기(seqrandom)을 1만회씩 차례로 수행하였다. 이는 일련의 데이터가 동시에 삭제되는 환경에서 이로 인한 읽기 쓰기 성능 변화를 관찰하기 위한 워크로드이다. 성능 결과는 총 5번의 실행을 수행한 후 평균값을 사용하였다.

그림 2는 FLSM-Tree를 사용하는 LevelDB 버전인 FDB와 기존 LevelDB 간의 성능을 비교하여 보여준다. fillseq 와 deleteseq 에서는 두 KVS 간의 성능차이가 거의 나타나지 않았다. 이는 FLSM-Tree의 경우 Put 연산을 수행할 때마다 cemetery 에 해당 키가 포함되어있는지 확인 후 삭제해주어야 하는 연산이 필요한데 해쉬 테이블을 사용해 O(1) 시간 복잡도 내에서 처리가 되기 때문에 실질적 오버헤드는 크지 않음을 보여준다. delete 의 경우 FLSM-Tree의 성능이 훨씬 좋을 것으로 예상했는데 실제로 비슷하게 관찰되었다. 이는 삭제 시 데이터가 삽입되는 Skip List의 시간복잡도가 원칙적으로는 O(logN)으로 O(1)의 해쉬 테이블보다 높지만 순차적으로 데이터를 삽입할 때에는 마지막으로 데이터가 삽입된 위치를 캐쉬하고 있어 O(1)시간 내에 처리가 가능하다. 읽기의 경우 FLSM-Tree를 사용해 상당한 성능 향상을 얻을 수 있었다. readseq는 FLSM-Tree를 사용했을 때 기존 LSM-Tree 대비 IOPS가 47% 향상되고 지연시간은 39% 감소하였다. readseq 는 특정 영역의 데이터를 순차적으로 읽게 되는데 FLSM-Tree 에서는 tombstone이 LSM-Tree의 상위계층을 잠식하여 성능이 저하되는 것을 막았고 삭제된 키는 cemetery를 이용해 빠르게 판별이 가능하기 때문에 성능 향상 폭이 컸다. 반면 readrandom 은 워크로드 자체에 시간지역성이 없이 임의의 키를 접근하기 때문에 tombstone 분리로 인한 성능 향상폭이 11% 정도로 작게 나타났다.

Ⅳ. 관련연구

데이터 저장 시스템에서 KVS 의 영향력이 점차 증가하면서 KVS의 성능을 다양한 관점에서 최적화하는 연구가 최근 활발히 수행되었다. SILK[8]는 LSM-Tree 에서 계층 간의 병합 연산이 발생할 때 지연시간이 크게 증가하는 것을 막는 태스크 스케줄링 기법을 제안하였다. JFDB[9]는 KVS의 삽입 기반 데이터 업데이트 방식이 중복키를 대량으로 발생시킴에 따라 탐색 연산의 성능저하가 심각하게 발생함을 지적하고 키당 수직적 리스트를

유지하여 해결하였다. 유사한 문제를 동일하지 않은 키를 포인팅하는 스트라이드 포인터를 이용해 해결한 연구도 수행된 바 있다[10]. SLM-DB[11]는 LSM-Tree가 쓰기에는 최적화되어 있으나 읽기에는 B-Tree 보다 성능이 좋지 않음을 관찰하고 이를 효율적으로 접목하여 읽기/쓰기 연산의 성능이 모두 좋은 Single-Level LSM Tree를 제안하였다. 이는 기존 LSM-Tree KVS 보다 공간 사용량이 많지만 최근 등장한 고집적도 메모리인 Persistent Memory를 사용하면 비용 증가 없이 실행가능한 기법이라고 할 수 있다. NoveLSM[12] 역시 Persistent Memory 환경에서 LSM-Tree 의 성능 및 병합 오버헤드를 절감할 수 있는 기법을 제안하였다.

본 논문과 가장 연관성이 높은 연구는 Lethe[13] 이다. Lethe는 LSM-Tree에서 삭제 연산이 실제 데이터가 삭제되는 것이 아니라 기존 데이터는 남아있는 상태에서 tombstone 데이터가 상위 계층에 추가되는 것이기 때문에 복구가 가능할 수 있으며 이에 GDPR에 어긋날 수 있음을 지적하였다. 이에 특정 시점이 되면 강제로 병합 연산을 시행하여 데이터의 실질적인 삭제를 보장하는 기법을 제안하였다. 본 연구는 삭제 연산으로 인해 LSM-Tree의 성능저하가 발생하는 것을 차단하는 연구로 Lethe 연구와는 차별적이며 상호 보완적으로 사용할 수 있을 것으로 기대된다.

V. 결 론

본 논문에서는 웹 기반 응용을 중심으로 널리 활용되고 있는 LSM-Tree 기반 키밸류 스토어에서 대량의 삭제 연산을 처리하는 과정에서 발생하는 비효율성을 적시하고 이를 해결하기 위한 방법을 제안하였다. cemetery라는 별도의 자료구조로 삭제된 키의 정보를 유지하고 LSM-Tree 에는 실제 유효한 키만 삽입하는 필터링 방식을 통해 읽기 성능 저하를 최소화하였다. 제안하는 기법은 상용 KVS인 LevelDB에 구현되었으며 기존 기법 대비 읽기 성능을 평균 47% 향상시켰다.

References

[1] J. Kim, K. J. Kwak, and J. M. Park, "NoSQL-based Sensor Web System for Fine Particles Analysis Services," The Journal of The Institute of Internet, Broadcasting and Communication (JIIBC), Vol. 19, No. 2, pp.119-125, 2019.
DOI: https://doi.org/10.7236/JIIBC.2019.19.2.119

[2] U. Park, "SQL Based Graph Pattern Query Performance on Relational DBMS," The Journal of Korean Institute of Information Technology, Vol. 17, No. 4, pp 9-20, 2019.
DOI: 10.14801/jkiit.2019.17.4.9

[3] N. Seo, Y. Kim, S. Kim, "Design, Implementation, and Performance Evaluation of an Embedded RDBMS Miracle," Journal of the Korea Academia-Industrial cooperation Society(JKAIS), Vol. 12, No. 7, pp. 3227-3235, 2011.
DOI: https://doi.org/10.5762/KAIS.2011.12.7.3227

[4] https://github.com/google/leveldb

[5] http://rocksdb.org/

[6] https://docs.ceph.com/en/latest/rados/configuration/storage-devices/

[7] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil. "The log-structured merge-tree (LSM-tree)." Acta Informatica, Vol. 33, No. 4, pp. 351-385, 1996.

[8] O. Balmau, F. Dinu, and W. Zwaenepoel, "SILK: Preventing Latency Spikes in Log-Structured Merge Key-Value Stores," USENIX Annual Technical Conference, pp. 753-766, 2019.

[9] J. Yeon, L. Kim, Y. Han, H. G. Lee, E. Lee, E, and B.S. Kim, "JellyFish: A Fast Skip List with MVCC," In Proceedings of the 21st International Middleware Conference, pp. 134-148, 2020.

[10] L. Kim and E. Lee. "An Enhancing Technique for Scan Performance of a Skip List with MVCC," The Journal of the Institute of Internet, Broadcasting and Communication (JIIBC), Vol. 20, No. 5, pp. 107-112.
DOI: https://doi.org/10.7236/JIIBC.2020.20.5.107

[11] K. Olzhas, S. Lee, B. Nam, S. H. Noh, and Y. Choi, "SLM-DB: Single-Level Key-Value Store with Persistent Memory." In 17th USENIX Conference on File and Storage Technologies (FAST 19), pp. 191-205. 2019.

[12] Kannan, S., Bhat, N., Gavrilovska, A., Arpaci-Dusseau, A. and Arpaci-Dusseau, R., 2018. Redesigning {LSMs} for Nonvolatile Memory with {NoveLSM}. In 2018 USENIX Annual Technical Conference, pp. 993-1005, 2018.

[13] S. Sarkar, T. I. Papon, D. Staratzis, and M. Athanassoulis. "Lethe: A tunable delete-aware LSM engine." In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 893-908. 2020.

저 자 소 개

이 은 지(정회원)

- 2005년 : 이화여자대학교 컴퓨터학과 학사
- 2012년 : 서울대학교 컴퓨터공학부 박사
- 2014 ~ 2018년 : 충북대학교 소프트웨어학과 전임교수
- 2019년 ~ 현재 : 숭실대학교 AI융합학부 부교수

- 주관심분야 : 데이터 저장 시스템

※ 이 연구는 미래창조과학부의 재원으로 한국연구재단(No.NRF-2019R1A2C1090337)의 지원을 받아 수행된 연구임.