The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 20, No. 5, pp.107-112, Oct. 31, 2020. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2020.20.5.107
JIIBC 2020-5-15

MVCC 지원 스킵 리스트의 범위 탐색 향상 기법

An Enhancing Technique for Scan Performance of a Skip List with MVCC

김이주*, 이은지**

Leeju Kim*, Eunji Lee**

요 약 본 논문에서는 LSM-tree 기반 키밸류 스토어에서 인메모리 데이터 관리를 위해 사용되는 스킵 리스트에 대한 연구를 수행하였다. 키밸류 스토어에서 사용되는 스킵 리스트는 덮어쓰기를 허용하지 않고 삽입만으로 모든 변경을 처리하는 삽입 기반 스킵 리스트이다. 이러한 동작 방식은 스냅샷 분리(Snapshot Isolation)을 통해 다중 읽기/쓰기 요청을 동시다발적으로 처리할 수 있는 MVCC(Multi-Version Concurrency Control)을 지원할 수 있다. 그러나 중복된 키가 다수 스킵 리스트에 존재함에 따라 리스트 탐색 시 불필요한 노드 방문으로 성능이 심각하게 저하될 수 있다. 특히 특정 범위의 데이터를 집합적으로 탐색하는 범위 탐색(Range Query)나 스캔(Scan) 연산 발생 시 심각한 오버헤드가 발생한다. 본 논문은 이러한 오버헤드를 줄이기 위해 새롭게 고안된 스트라이드 스킵 리스트(Stride Skip List)를 제안한다. 스트라이드 스킵 리스트는 동일 키의 마지막 노드에 대한 인덱싱 포인터를 추가적으로 유지하여 불필요한 노드 방문을 피할 수 있도록 한다. 제안된 기법은 RocksDB의 인메모리 컴포넌트를 활용하여 구현되었으며 다양한 워크로드에서 SCAN 연산의 성능을 기존 스킵 리스트 대비 최대 350배까지 향상시켰다.

Abstract Recently, unstructured data is rapidly being produced based on web-based services. NoSQL systems and key value stores that process unstructured data as key and value pairs are widely used in various applications. In this paper, a study was conducted on a skip list used for in-memory data management in an LSM-tree based key value store. The skip list used in the key value store is an insertion-based skip list that does not allow overwriting and processes all changes only by inserting. This behavior can support Multi-Version Concurrency Control (MVCC), which can simultaneously process multiple read/write requests through snapshot isolation. However, since duplicate keys exist in the skip list, the performance significantly degrades due to unnecessary node visits during a list traverse. In particular, serious overhead occurs when a range query or scan operation that collectively searches a specific range of data occurs. This paper proposes a newly designed Stride SkipList to reduce this overhead. The stride skip list additionally maintains an indexing pointer for the last node of the same key to avoid unnecessary node visits. The proposed scheme is implemented using RocksDB's in-memory component, and the performance evaluation shows that the performance of SCAN operation improves by up to 350 times compared to the existing skip list for various workloads.

Key Words : Key-value Store, Skip List, MVCC, NoSQL Systems

*비회원, 숭실대학교 스마트시스템소프트웨어학과
**정회원, 숭실대학교 스마트시스템소프트웨어학과 (교신저자)
접수일자 2020년 9월 7일, 수정완료 2020년 10월 7일
게재확정일자 2020년 10월 9일

Received: 7 September, 2020 / Revised: 7 October, 2020 / Accepted: 9 October, 2020
*Corresponding Author: ejlee@ssu.ac.kr
Dept. of Smart Systems Software, Soongsil University, Korea

Ⅰ. 서 론

최근 IT기술의 고도화와 함께 디지털 데이터가 기하급수적으로 증가하고 있다. 페이스북이나 인스타그램과 같은 SNS 서비스부터 유투브나 틱톡에 이르는 동영상 기반 서비스까지 디지털 데이터는 유래 없이 빠른 속도로 생산 및 확산되고 있다. 이러한 웹 기반 응용에서 생산되는 데이터들은 기존의 정형화 된 데이터와는 달리 대부분 속성과 형태가 다른 비정형 데이터다[1]. 비정형 데이터는 테이블 형태로 데이터를 관리하는 전통적인 데이터베이스 시스템으로 관리할 때 많은 문제가 생긴다. 각 데이터(행)에 대해 고정된 개수의 속성(열)을 부여하여 정형화 된 형태로 데이터를 관리할 경우 다양한 속성을 지닌 비정형 데이터를 관리하기 위해서는 무의미하게 유지되는 셀이 다수 발생할 수 있기 때문이다.

NoSQL 시스템은 이러한 데이터의 특성 변화에 대해 비정형 데이터를 효율적으로 관리할 수 있도록 만들어진 데이터 서비스 플랫폼이다[2]. 데이터를 키(Key)와 밸류(Value)의 쌍으로만 표현하여 유연하게 다양한 형태의 데이터를 정의하고 관리할 수 있도록 하는 것이다. 키밸류 스토어(Key-value Store), 오브젝트 스토어(Object Store) 등은 인터페이스나 규모 상에 약간의 차이는 존재하지만 근본적으로 NoSQL 시스템으로 분류될 수 있다.

본 논문은 현재 다양한 응용에서 널리 사용되는 키밸류 스토어의 메모리 컴포넌트에서 공통적으로 발견되는 비효율성 문제를 규명하고 해결하는 연구를 진행하였다. 키밸류 스토어에서는 메모리 내부 데이터를 관리하기 위해 대부분 스킵 리스트(SkipList)를 사용한다[3][4][5]. 키밸류 스토어에서 사용되는 동시실행성을 높이기 위해 삽입 전용 형태로 동작하는데, 이 과정에서 중복된 데이터가 스킵 리스트에 존재하게 된다. 이로 인해 범위 탐색(Range Query)나 스캔(Scan) 연산의 성능이 심각하게 저하되는 현상이 관찰되었다.

이에 본 저자들은 동일 키 그룹의 탐색을 효율적으로 수행할 수 있도록 하는 스트라이드 스킵 리스트(Stride Skip List)를 제안한다. 기존에 존재하던 키가 업데이트 되는 경우 해당 키 그룹의 가장 오래된 데이터를 가리키는 스트라이드 포인터를 함께 유지하여 범위 탐색 시 다음 키로 넘어가는 시간을 단축시킨다.

제안된 기법은 RocksDB의 메모리 컴포넌트로 활용되는 스킵 리스트를 변형하여 구현하였으며 다양한 분포를 지니는 워크로드에서 평가되었다. PUT, GET, SCAN 연산에 대해 성능평가를 수행하였으며 SCAN 연산의 경우 스트라이드 스킵 리스트가 기존 삽입 전용 스킵 리스트 대비 350배 높은 성능을 제공하였다.

본 논문의 구성은 다음과 같다. 2장에서는 LSM-tree 기반의 키밸류 스토어에 대해 설명하고 3장에서는 기존 스킵리스트와 제안하는 스트라이드 스킵 리스트의 구조를 자세히 설명한다. 4장에서는 성능평가를 수행하고 5장에서는 결론을 맺는다.

Ⅱ. 연구배경

현재 상용 시스템에서 많이 사용되는 키밸류 스토어들은 대부분 LSM(Log structured Merge)-tree 기반의 구조로 데이터를 관리한다[6]. 데이터를 로그를 쓰듯이 덧붙이는 방식으로 저장하는데 주기적으로 백그라운드 병합 과정을 통해 stale 데이터를 수거하는 방식으로 동작한다. LSM-tree는 데이터의 로그를 계층적으로 나누어 최신 데이터를 포함한 로그 파일은 고속의 디바이스에 유지하고 오래 전에 쓰여진 데이터는 상대적으로 느린 디바이스에 관리한다. 이러한 방식은 저장시스템의 비용 효율성을 증가시키고 임의 쓰기가 스토리지 성능을 저하시키는 것도 방지하기 때문에 작은 크기의 쓰기가 빈번하게 발생하는 웹 기반 서비스에 매우 적합하다. 그림 1은 LSM-tree의 구조를 보여주고 있다. L0은 Level 0에 있는 파일들을 나타내며 0번부터 N번 계층까지 있는 LSM-tree 구조를 나타낸다.

키밸류 스토어에서 사용되는 LSM-tree는 가장 상위 계층의 데이터를 메인 메모리 버퍼에 유지한다. memtable

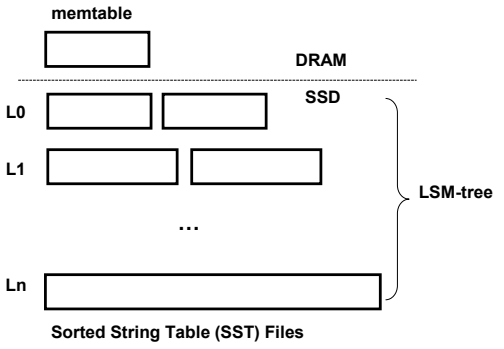


그림 1. LSM-tree 기반 키밸류 스토어의 데이터 저장 구조
Fig. 1. LSM-tree structure.

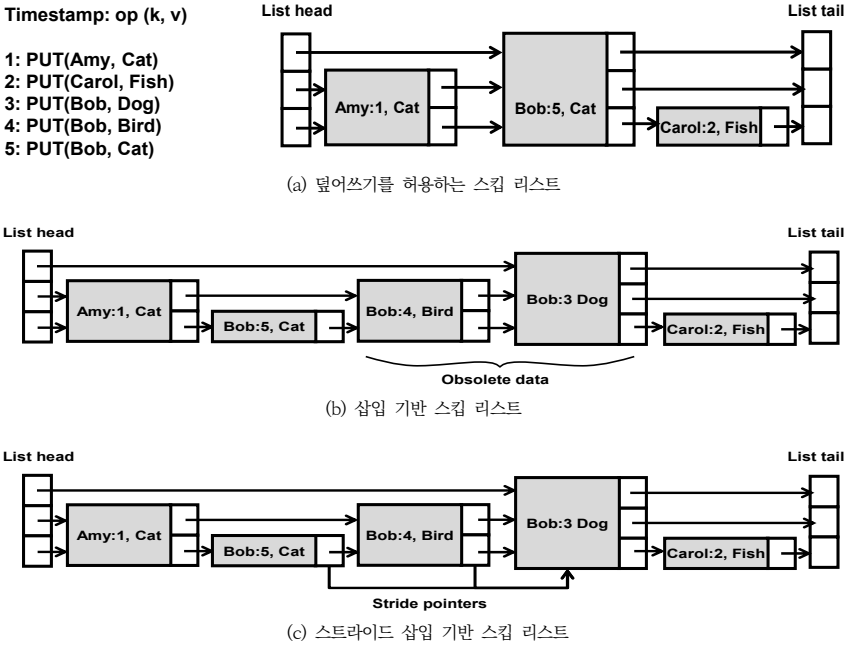
이라고 불리는 이 버퍼는 통상 수십MB 크기를 가진다 (예: RocksDB는 64MB 크기가 기본 설정값임). 대부분의 키밸류 스토어에서 memtable을 관리하기 위한 자료구조로 스킵 리스트(SkipList)를 사용한다. 스킵 리스트는 일반적인 연결 리스트(LinkedList)에서 검색속도를 높이기 위해 인덱싱 기능을 강화한 자료구조이다. 스킵 리스트는 추가적인 인덱싱 계층을 유지함으로써 삽입과 검색 연산에서 O(LogN)의 시간복잡도를 제공한다.

Ⅲ. 스트라이드 스킵 리스트

본 논문에서 해결하고자 하는 문제는 현재 대부분의 키밸류 스토어는 스킵 리스트에서 데이터 업데이트가 발생하는 경우 덮어쓰지 않고 삽입 연산으로 처리하는 데에서 발생하는 것이다. 이러한 쓰기 방식은 MVCC (Multi-version Concurrency Control) 속성을 지원하기 위함이다[7]. MVCC는 데이터 셋에 변경이 일어날 때 스냅샷을 생성하여 일관성을 유지하면서도 동시다발적 읽기/쓰기 요청을 처리할 수 있도록 해주는 데이터 관리 기법이다. 그러나 이러한 삽입만 수행하는 특성은 스킵 리스트 내부에 중복된 데이터를 계속해서 유지하도록 하여 스캔(Scan) 연산이나 범위 탐색과 같은 연산(Range Query)의 성능을 크게 저하시킨다.

그림 2는 일련의 PUT 연산이 발생할 때 덮어쓰기를 허용하는 버전((a))과 키밸류 스토어에서 사용하는 삽입 기반 스킵 리스트((b))의 상태를 비교하여 보여준다. 주어진 워크로드는 그림2(a)의 옆에 제시되어 있다. 워크로드는 Bob 키에 대하여 세 번의 업데이트를 발생시킨다. MVCC를 지원하는 키밸류 스토어들은 PUT연산이 발생할 때 각 연산 별로 타임스탬프를 부여한다. 이는 해당 키에 읽기 연산이 발생할 때 가장 최신의 일관된 데이터를 찾기 위함이다.

그림2(a)의 덮어쓰기를 허용하는 스킵 리스트의 경우 유효한 세 개의 아이템(키/밸류의 쌍)만을 유지하고 있는 반면, 그림2(b)의 MVCC를 지원하는 스킵 리스트의 경우 Bob에 대해 이전 버전의 모든 데이터(Obsolete data)를 유지하고 있다. 이 경우 전체 데이터를 탐색하는 스캔 연산이 발생하면 덮어쓰기를 허용하는 스킵 리스트의 경우 3회 노드 방문이 필요하지만 현재 키밸류 스토어에서 사용되는 삽입 기반 스킵 리스트는 5회 노드 방문이 필요하다. 이러한 오버헤드는 업데이트가 빈번한 워크로드에서는 더욱 심각해진다. 예를 들어, 키밸류 스토어는 실제 분산 데이터 저장시스템과 같은 대규모 시스

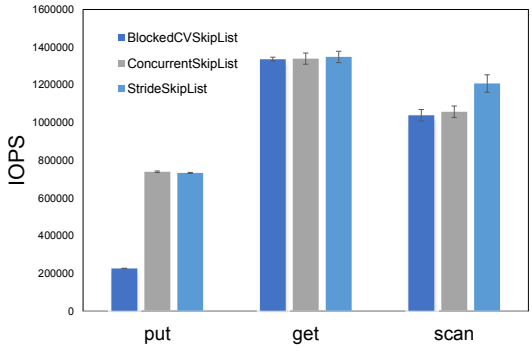


(a) 덮어쓰기를 허용하는 스킵 리스트

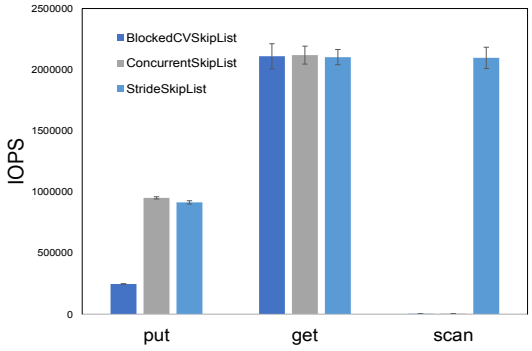
(b) 삽입 기반 스킵 리스트

(c) 스트라이드 삽입 기반 스킵 리스트

그림 2. 키밸류 스토어에서 스킵 리스트 유형에 따른 상태 비교
Fig. 2. Skip list comparison for KVS



(a) Uniform-distribution workload



(b) Zipfian-distribution workload

그림 3. 스트라이드 스킵 리스트의 성능
Fig. 3. IOPS of Stride skip list.

템에서 오브젝트의 메타데이터를 저장하는 용도로 활용되는 경우가 많은데 이 경우 업데이트가 빈번하게 발생한다. 이러한 데이터에 스캔 연산이 발생하면 성능이 기하급수적으로 저하될 수 있다.

이러한 문제를 해결하기 위해 본 논문에서는 스트라이드 스킵 리스트(Stride SkipList)를 제안한다. 제안된 스킵 리스트는 각 노드에 자신의 키와 동일한 키를 가진 노드 중 가장 오래된 노드를 가리키는 스트라이드 포인터를 함께 유지한다. 그림2(c)는 제안된 스트라이드 스킵 리스트의 구조를 보여준다. Bob 키를 가진 중복된 노드들이 추가적으로 유지하는 포인터가 스트라이드 포인터이다. 스캔(Scan) 연산이 발생하면 제안된 스킵 리스트는 스트라이드 포인터를 이용해 빠르게 데이터를 탐색할 수 있다. 그림 2의 예시에서는 Bob 키가 담긴 첫 번째 노드에서 다음 노드로 이동하지 않고 Bob 키를 가진 노드의 가장 끝 노드로 곧바로 이동하기 때문에 노드 탐색의 수가 4회가 된다. 중복된 키가 많을수록 스트라이드 포인터를 활용해 절약할 수 있는 노드 방문 횟수는 감소될 수 있다.

스킵 리스트에서 동일한 키들 간에는 어떠한 순서로 정렬되는지 간략히 설명하고자 한다. MVCC를 지원하는 키밸류 스토어에서는 실제 스킵 리스트에서 저장하는 데이터의 내부 키로 사용자가 전달한 키와 타임스탬프의 조합을 사용한다. 따라서 사용자의 키가 동일한 데이터는 Prefix가 동일하기 때문에 스킵 리스트 상에서 일렬로 배치된다. 동일 키 간에는 타임스탬프가 큰(최신의) 데이터가 앞선 위치에 놓이도록 정렬된다. 이것은 GET 연산을 수행할 때 가장 최신의 데이터를 돌려주기 위함이다. 삽입 기반 스킵 리스트에서는 이를 위해 탐색 중간에 찾고자하는 키를 발견하더라도 가장 하위 레벨에 도착할 때까지 탐색을 지속한다.

Ⅳ. 성능평가

제안된 스트라이드 스킵 리스트는 RocksDB의 인메모리 컴포넌트인 InlineSkipList 모듈을 활용하여 구현하였다. 성능 비교를 위하여 세 가지 버전의 스킵 리스트를 구현하였다. BlockedCVSkipList 는 Google에서 개발된 LevelDB에서 사용하는 스킵 리스트를 모사한 것으로 이것 역시 업데이트 연산을 삽입으로 처리하지만 쓰기 연산이 Lock-free로 구현되어 있지 않아 Condition Variable을 사용해 동시 쓰기 요청을 제어한다[4]. ConcurrentSkipList는 RocksDB에서 사용하는 SkipList를 나타내며 이는 RocksDB를 변형해 만든 다양한 버전의 키밸류 스토어에서도 널리 사용된다[3]. 마지막으로 StrideSkipList 는 본 논문에서 제안한 스킵 리스트이다. 본 논문에서는 두 가지 워크로드를 대상으로 스킵 리스트의 성능을 평가하였다. Uniform 워크로드는 데이터의 업데이트가 발생하지 않고 각 데이터가 모두 서로 다른 키를 가진 워크로드이다. Skewed 워크로드는 특정 데이터 셋에 업데이트가 빈번하게 발생하는 워크로드로 파이썬의 Numpy 패키지를 이용해 Zipfian distribution을 따르는 워크로드를 생성했다. Skewness 계수로는 1.2를 사용하였다. 워크로드는 SET, GET, SCAN 연산을 100만번 실행하도록 하였으며 SCAN 시에는 10개의 데이터를 탐색하도록 하였다.

그림 3은 워크로드 별로 스킵 리스트의 성능을 보여주

고 있다. Uniform-distribution을 따르는 워크로드에서는 StrideSkipList가 기존 Lock-free의 삽입 기반 스킵 리스트(ConcurrentSkipList)와 PUT과 GET에서는 비슷한 성능을 나타냈다. SCAN연산의 경우 14% 높은 IOPS를 제공하였다. 이것은 사실 예상과 다른 결과이다. 중복된 키가 존재하지 않기 때문에 SCAN 연산에서도 StrideSkipList 의 성능이 향상될 부분이 없기 때문이다. 이에 내부 동작을 분석한 결과 성능 향상은 SCAN 연산을 수행할 때 스트라이드 포인터를 활용해 문자열 비교 횟수가 감소되었기 때문으로 파악되었다. SCAN 연산 수행 시 서로 다른 키 n개를 찾아야 하는데, 기존 스킵 리스트의 경우 현재 접근하고 있는 노드의 키가 새로 찾은 노드인지를 판단하기 위해 문자열 비교를 수행한다. 그런데 StrideSkipList 에서는 해당 노드가 Stride Pointer를 가지고 있지 않은 노드를 카운트 하면 사실상 접근한 서로 다른 키의 개수를 나타내기 때문에 문자열 비교를 하지 않고도 탐색을 종료해야 하는 시점을 파악할 수 있다. 이러한 효과 때문에 Uniform-distribution workload에서도 SCAN 연산에 대해서는 성능 향상이 있었다. BlockedCVSkipList는 GET과 SCAN 연산에서는 ConcurrentSkipList와 동일한 성능을 제공하지만 쓰기 연산에서는 Condition Variable을 통해 쓰기 요청이 직렬화 되기 때문에 성능이 낮게 관찰되었다.

Zipfian distribution을 따르는 워크로드에서는 StrideSkipList 가 기존 ConcurrentSkipList 대비 350배 높은 SCAN 성능을 제공하였다. 이것은 업데이트가 빈번하게 발생하여 중복키를 다수 포함하고 있는 상황에서 StrideSkipList는 스트라이드 포인터를 이용해 오래된 데이터를 탐색하느라 발생하는 비용을 절감할 수 있기 때문이다. PUT과 GET에서는 기존 ConcurrentSkipList와 유사한 성능을 나타내었다.

Ⅳ. 관련연구

스킵 리스트는 1993년도에 William Pugh에 의해 최초로 제안된 이후 다양한 응용에서 널리 활용되고 있다[8]. O(LogN) 시간 복잡도를 지원하면서도 구조가 단순하기 때문에 효율적이면서도 구현 난이도가 낮기 때문이다. 이에 키밸류 스토어와 같은 데이터 저장 시스템부터 분산 응용[9][10][11], 태스크 스케줄러에 이르기까지 그 활용도가 점차 확대되고 있는 추세이다[12]. 이러한 스킵 리스트의 동시실행성을 높이기 위해 Lock-free 스킵 리스트가 제안되어 실제 JavaTM SE 6 Platform에 구현된 바 있다[13]. 최근에는 스킵 리스트가 노드 삽입 시 다수의 링크를 업데이트 함에 따라 다중 쓰레드 환경에서 경쟁이 심해지는 것을 방지하기 위해 상위 레벨 인덱싱 업데이트는 비동기적으로 실행하는 스킵 리스트가 제안된 바 있다[14]. 이러한 스킵 리스트는 삽입 연산의 성능은 향상될 수 있으나 인덱싱 형성이 단일 쓰레드에 의존하여 속도가 느리기 때문에 읽기 성능이 저하된다는 단점이 있다. 또한 최근에는 대용량 메모리를 사용하는 NUMA 환경에서 스킵 리스트의 성능을 향상시키기 위해 인덱싱 역할을 수행하는 상위 레벨의 노드들을 CPU 노드마다 유지하는 기법이 제안된 바 있다[15]. 본 논문에서 제안한 스트라이드 스킵 리스트는 MVCC를 지원하는 관점에서의 최적화로 기존 연구에서 제안된 기법들과 차별적이며 통합적으로 적용할 수 있는 기법이다.

Ⅴ. 결 론

본 논문에서는 키밸류 스토어에서 메모리 데이터를 관리하기 위해 널리 사용되고 있는 스킵 리스트를 타겟으로 MVCC를 지원할 때 발생하는 성능 비효율성을 규명하고 개선하였다. 스냅샷 생성을 위해 삽입 기반으로 동작하는 기존의 스킵 리스트는 중복키를 유지함에 따라 범위 탐색 연산의 속도가 저하된다. 제안된 스트라이드 스킵 리스트는 동일 키를 빠르게 스킵 할 수 있는 스트라이드 포인터를 유지하여 MVCC를 지원하면서도 고속 스캔이 가능하도록 하였다. 제안된 기법은 상용 키밸류 스토어의 메모리 컴포넌트를 활용해 구현되었으며 트레이스 기반 성능평가에서 기존 기법 대비 최대 350배 SCAN 성능을 향상시켰다.

References

[1] J. Kim, K. Kwak, and J. 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] K. Kwak, H. Yoon, D. Shin, J. Park, and J. Kim, “Spatial Operator for Spatial MongoDB,” The Journal of The Institute of Internet, Broadcasting and Communication (JIIBC), Vol. 18, No. 6, pp.237-242,

2018.
DOI: https://doi.org/10.7236/JIIBC.2018.18.6.237

[3] https://rocksdb.org/

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

[5] https://github.com/rescrv/HyperLevelDB

[6] X. Wu, Y. Xu, Z. Shao, S. Jiang, "LSM-trie: An LSM-tree-based ultra-large key-value store for small data items," USENIX Annual Technical Conference, 2015.

[7] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A Critique of ANSI SQL Isolation Levels," ACM SIGMOD Record, Vol. 24, No. 2, pp.1-10, 1995. DOI: https://doi.org/10.1145/568271.223785

[8] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," Communications of ACM, Vol. 33, No. 6, pp. 668-676, 1990.
DOI: https://doi.org/10.1145/78973.78977

[9] https://cassandra.apache.org/

[10] https://www.memsql.com/

[11] N. Shavit and I. Lotan,, "Skiplist-based concurrent priority queues," In Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS), 2000.
DOI: https://doi.org/10.1109/IPDPS.2000.845994

[12] https://lwn.net/Articles/551896/

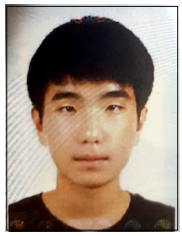
[13] K. Fraser, "Practical lock-freedom," University of Cambridge, Computer Laboratory Tech. Rep. UCAM-CL-TR-579, Feb. 2004.

[14] T. Crain, V. Gramoli, and M. Raynal, "No hot spot non-blocking skip list," In Proceedings of IEEE 33rd International Conference on Distributed Computing Systems. 2013.
DOI: https://doi.org/10.1109/ICDCS.2013.42

[15] H. Daly, A. Hassan, M. F. Spear, and R. Palmieri, "NUMASK: High Performance Scalable Skip List for NUMA," In 32nd International Symposium on Distributed Computing (DISC), Vol. 121, pp.18:1-18:19, 2018.
DOI: https://doi.org/10.4230/LIPIcs.DISC.2018.18

저 자 소 개

김 이 주(비회원)



- 2020년 숭실대학교 스마트 시스템소프트웨어학과 학사과정
- 주관심분야 : 키밸류 스토어, 분산시스템, 스토리지 시스템

이 은 지(정회원)

- 2005년 이화여자대학교 컴퓨터학과 학사
- 2012년 서울대학교 컴퓨터공학부 박사
- 2019년 숭실대학교 스마트시스템 소프트웨어학과 조교수
- 주관심분야: 데이터 시스템, 차세대 메모리, 분산 시스템, 클라우드 컴퓨팅

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