ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 45, No. 11, pp. 1193-1202, 2018. 11
https://doi.org/10.5626/JOK.2018.45.11.1193

인-메모리 키-값 데이터베이스를 위한 비 휘발성 메모리 기반 영속적 로그 버퍼 기법

(A Persistent Log Buffer Technique using Non-volatile Memory for In-Memory Key-Value Databases)

김 도 영 † 최 원 기 †† 성 한 승 † 이 지 환 † 박 상 현 †††
(Doyoung Kim) (Won Gi Choi) (Hanseung Sung) (Jihwan Lee) (Sanghyun Park)

요 약 Redis는 인-메모리 키-값 데이터베이스로, 실시간 데이터 처리 및 저장이 필요한 서비스에서 많이 사용되고 있다. 하지만 DRAM의 특징인 휘발성으로 인하여, 시스템이 비정상적으로 종료되면 Redis에 저장된 데이터들이 손실되는 문제가 발생한다. 이를 방지하기 위해 Redis는 로그를 디스크에 저장하고, 로그를 이용하여 데이터를 복구함으로써 데이터 손실을 방지한다. 요청된 명령을 로그 형태로 디스크에 저장하는 방식인 AOF 복구 메커니즘은 1초마다 주기적으로 로그를 기록하는 everysec 정책과 명령이 요청될 때마다 로그를 기록하는 always 정책으로 동작한다. everysec 정책은 Redis의 성능을 저하시키지 않지만, 시스템이 1초내에 비정상적으로 종료되면 데이터 손실이 발생할 수 있다. 반면 always 정책은 데이터 손실은 발생하지 않지만, 명령이 요청될 때마다, 디스크 연산을 수행하기 때문에 성능 저하의 원인이 된다. 본 논문에서는 everysec 정책에서 아직 디스크에 동기화되지 않고 AOF 버퍼에 저장되어 있는 AOF 로그들을 비 휘발성 메모리에 저장함으로써, 데이터 손실이 발생하는 단점을 극복하고 always 정책에 비해 약 100배 더 향상된 성능을 가진 시스템 모델을 제시한다.

키워드: 인-메모리 키-값 데이터베이스, 복구, 비 휘발성 메모리, 레디스

Abstract Redis, an In-Memory Key-Value Database, is widely used in services that require real time data processing and storage. Since main memory is volatile, Redis has a problem of data loss if the system is terminated abnormally. To prevent this problem, Redis stores logs on disk, preventing data loss by restoring logs when the system is terminated. The AOF recovery mechanism, a method of appending requested commands in disk as a log format, operates with the "everysec" policy that writes logs every second, and the "always" policy that writes a log every time a command is requested. The "everysec" policy does not degrade performance of Redis, but data loss can occur if the system is terminated abnormally within one second. Conversely, the "always" policy does not cause data loss, but it requires disk operation for every command, causing performance degradation. We propose a system model that constructs AOF buffer in non-volatile memory and stores logs in the buffer, which are not synchronized to disk in the "everysec" policy. The proposed model prevents data loss and has approximately 100 times better performance than the "always" policy.

Keywords: in-memory key-value database, recovery, NVRAM, Redis

· 이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(IITP-2017-0-00477, (SW스타랩) IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발)
· 이 논문은 2018 한국컴퓨터종합학술대회에서 '비휘발성 메모리를 이용하여 데이터 영속성을 유지한 인 메모리 키-값 데이터베이스'의 제목으로 발표된 논문을 확장한 것임

† 비 회 원 : 연세대학교 컴퓨터과학과
kem2182@yonsei.ac.kr
hssung@yonsei.ac.kr
ji_hwan43@yonsei.ac.kr
†† 학생회원 : 연세대학교 컴퓨터과학과
cwk1412@yonsei.ac.kr
††† 종신회원 : 연세대학교 컴퓨터과학과 교수(Yonsei Univ.)
sanghyun@yonsei.ac.kr
(Corresponding author임)

논문접수 : 2018년 8월 6일
(Received 6 August 2018)
논문수정 : 2018년 9월 3일
(Revised 3 September 2018)
심사완료 : 2018년 9월 11일
(Accepted 11 September 2018)

정보과학회논문지 제45권 제11호(2018. 11)

1. 서 론

인-메모리 키-값 데이터베이스는 키-값 데이터를 포함하여 데이터베이스를 이루는 모든 메타 데이터들을 메인 메모리(DRAM)에 유지하는 데이터베이스이다[1]. 하지만 DRAM은 휘발성 성질을 가지고 있으므로, 전원이 종료되면 DRAM에 저장된 데이터들이 모두 손실된다. 인-메모리 키-값 데이터베이스 중 하나인 Redis[2]는 이 문제를 방지하기 위하여, 데이터를 DRAM에 저장할 때 별도의 로그 파일을 디스크에 저장한다. 이후에 시스템이 비정상적으로 종료되면 Redis는 복구 메커니즘을 실행하여, 디스크에 저장되어 있는 로그 파일로 DRAM의 데이터를 복구함으로써 데이터 지속성을 유지한다.

Redis는 로그 기법으로 Redis-Database File(RDB)와 Append-Only File(AOF)를 제공한다[3]. RDB는 Redis에 저장된 데이터를 스냅샷(Snapshot) 형태로 로그 파일에 기록하는 기법이며, AOF는 Redis에 요청된 명령어를 메모리에 있는 AOF 버퍼에 임시로 저장했다가 디스크에 로그로 기록하는 방법이다. 하지만 RDB와 AOF 모두 디스크 연산을 필요로 하므로 Redis의 성능이 저하되는 요인이 된다. 이를 방지하기 위해 RDB와 AOF-everysec 로그 정책은 백그라운드 쓰레드에서 주기적으로 디스크 연산을 처리하지만 이 과정에서 데이터 손실이 필연적으로 발생한다. AOF는 데이터 손실을 방지하기 위하여 명령어가 들어올 때마다 디스크에 로그를 작성하는 always 정책으로 사용될 수 있지만, 이는 데이터를 변경하는 모든 명령마다 디스크 연산을 필요로 하므로 성능이 극도로 저하된다.

비 휘발성 메모리(Non-Volatile Memory, NVM, NVRAM)는 DRAM과는 다르게 전원이 차단되더라도 메모리 안에 데이터가 손실되지 않고 존재하는 특징을 가지는 메모리이다[4]. 활발히 연구되고 있는 NVRAM의 종류로는 PCM(Phase Change Memory), STT-RAM(Spin Transfer Torque RAM), ReRAM(Resistive RAM), 3D XPoint가 있다. Block 단위로 데이터를 접근하는 하드디스크(HDD)와 Page 단위로 데이터를 접근하는 솔리드 스테이트 드라이브(SSD)와는 다르게 NVRAM은 DRAM처럼 Byte 단위로 데이터를 접근한다. 그리고 NVRAM은 DRAM에 준하는 접근 속도를 가지고 있다. 빠른 접근 속도와 비 휘발성의 특징으로 인하여, NVRAM은 차세대 메모리로 각광받고 있다. 인-메모리 키-값 데이터베이스는 높은 처리율을 가지고 있지만, 데이터 보존을 위한 로그 기법들이 디스크에 데이터를 저장하기 때문에 일시적으로 성능 저하가 발생하게 되거나 데이터 손실이 발생할 수 있다. 하지만 NVRAM을 사용하면 빠른 접근 성능을 가지고 데이터 지속성도 보장할 수 있으므로, 디스크를 대신하여 NVRAM을 활용하는 [5-13]과 같은 연구들이 진행되고 있다.

인텔에서는 NVRAM을 조작하는 Persistent Memory Development Kit(PMDK) API[14]를 이용하여 Redis의 모든 키-값 데이터들을 NVRAM에 저장함으로써 데이터 지속성을 보장하고 Redis의 성능을 저하시키지 않는 시스템 모델을 제안하였다[15]. 하지만 NVRAM은 비싼 가격과 가격 대비 용량으로 인하여 상용화하기 어렵다는 단점이 있으므로[16], 인텔이 제안한 모델처럼 모든 데이터를 NVRAM에 운용하기에 한계가 존재한다.

본 논문에서는 NVRAM에 AOF 버퍼를 구현하여 AOF everysec 정책이 가지고 있는 데이터 지속성 문제를 해결하고, Redis의 처리 성능도 크게 저하되지 않는 Persistent-Buffer Redis(PB-Redis)을 제안한다. PB-Redis는 키-값 데이터를 DRAM에 저장하고 AOF 버퍼를 NVRAM에 저장하기 때문에, 인텔에서 구현한 PMDK-Redis와 비교하여 상대적으로 적은 용량의 데이터들만 NVRAM에 저장한다. PB-Redis는 AOF-always 정책과 같은 데이터 지속성을 보장하고 PMDK-Redis가 가지고 있는 NVRAM 용량 제한 문제를 개선하면서, AOF-everysec 정책을 사용하는 Redis에 준하는 처리 성능을 기대할 수 있다.

본 논문은 다음 섹션들로 구성된다. 섹션 2에서는 논문의 이해에 필요한 배경 지식들을 간략하게 설명한다. Redis의 해쉬 테이블 구조와 다양한 로그 기법들을 설명하고, NVRAM을 조작하는 인터페이스 라이브러리와 인텔에서 제안한 시스템 모델을 설명한다. 섹션 3에서는 본 논문이 제안한 PB-Redis 모델의 구조를 설명하고, 섹션 4에서는 다양한 벤치마크 프로그램들을 사용하여 PB-Redis의 성능을 측정하고 비교한다. 마지막으로 섹션 5에서는 PB-Redis가 제시한 방향의 결론에 대해 기술하고 본 연구의 향후 계획에 대하여 서술한다.

2. 배 경

2.1 Redis의 구조

Redis는 저장된 데이터들을 그림 1과 같이 해쉬 테이블 구조로 DRAM에서 관리한다. Redis에서 키-값 데이터를 처리하는 과정은 다음과 같다. 먼저 클라이언트 프로그램에서 Redis에 키-값 데이터를 전달하여 저장을 요청한다. 클라이언트와 Redis 간에 데이터 전송이 성공적으로 완료되면 Redis는 해쉬 함수에 전달받은 키 데이터를 파라메터로 입력하여 키 데이터에 해당하는 해쉬 값을 계산한다. 계산된 해쉬 값은 해쉬 테이블의 색인 값으로 사용되고 Redis는 해쉬 테이블의 색인 위치에 키-값 데이터를 엔트리(Entry)로 만들어 저장한다.

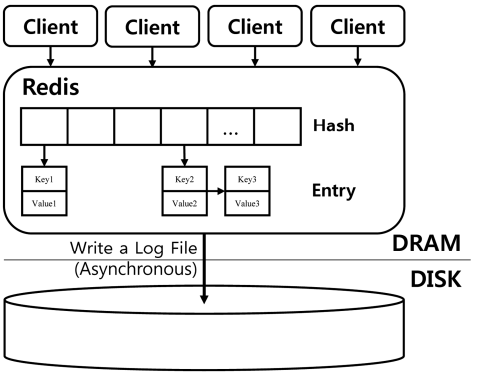

그림 1 Redis의 해쉬 테이블 구조
Fig. 1 Hash table structure of Redis

만약 색인 위치에 이미 다른 키-값 데이터가 저장된 상태라면 체이닝 기법을 통해 저장되어 있는 데이터 엔트리와 새로운 데이터 엔트리를 리스트로 연결하여 저장한다.

체이닝 기법은 구현이 쉽다는 장점이 있지만 연결된 데이터가 많아지면 처리 성능이 저하되는 단점이 있다. Redis에서는 해쉬 테이블 크기와 저장된 엔트리 수를 비교하여 일정 비율이 넘어가면 해쉬 테이블을 확장하고 리해쉬(Rehash)하여 엔트리를 재배치한다. 리해쉬를 통해 엔트리를 재배치함으로서, Redis의 처리 성능이 체이닝으로 인하여 저하되는 것을 방지한다.

2.2 RDB 파일 (Redis-Database File)

RDB 파일은 Redis의 메모리에 있는 모든 데이터들을 스냅샷(Snapshot) 형태로 로그 파일에 기록하는 방법이다. RDB 파일에 스냅샷 형태로 저장되는 모든 데이터들은 그림 2와 같이 모두 바이너리 포맷으로 저장된다. Redis에서 RDB 파일을 이용한 로깅은 다음과 같은 순서로 진행된다. 특정 시점에 존재하는 Redis 데이터를 저장하기 위해 RDB 파일을 새로 생성하고, 생성된 파일에 그림 2와 같이 RDB 파일의 고유 번호(Magic Number)와 데이터베이스 번호(Database Number)를 파일에 작성한다. 그 다음 고유 번호와 데이터베이스 번호를 모두 작성한 후, 데이터베이스 번호에 해당하는 모든 키-값 데이터들을 RDB 파일에 순차적으로 작성한다. 마지막으로 이전에 작성했던 RDB 파일들을 모두 제거하면 RDB의 로그 기록이 완료된다.

위와 같은 과정을 통해 생성된 RDB 파일을 이용해서 데이터를 복구할 경우, RDB 파일에 저장 되어있는 키-값 데이터를 읽으며 데이터 복구를 진행한다. 한 개의 키에 대해서 오직 한 개의 데이터만이 RDB 파일에 저장되므로 로그 파일의 크기가 작고 복구 시간이 단축되는 장점이 있다. 하지만 RDB 파일은 특정 시점 간격으로 로그를 생성하기 때문에 새로운 RDB 파일이 작성되

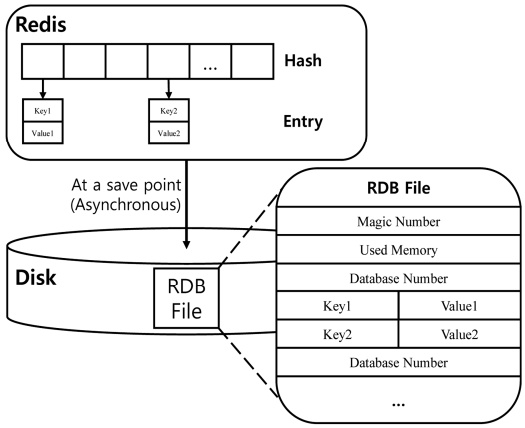

그림 2 RDB 파일 로그 기법
Fig. 2 RDB file log method

기 전에 시스템이 비정상적으로 종료된다면 그 사이에 생성된 데이터들은 복구할 수 없다.

2.3 AOF 파일 (Append Only File)

AOF 파일은 Redis에 요청된 명령 중에서 데이터를 변경하는 명령(SET, UPDATE, DELETE)을 로그 파일에 기록하는 방법이다. Redis의 데이터를 변경하는 명령들은 그림 3과 같이 요청된 명령 순서대로 AOF 파일에 기록된다. Redis에서 AOF 파일을 이용한 로깅은 다음과 같은 순서로 진행된다. 먼저 Redis의 데이터를 변경하는 명령이 요청되는 경우, Redis의 데이터를 변경한 다음 AOF 버퍼에 요청된 명령을 추가한다. 그리고 fsync 함수가 호출되면 AOF 버퍼에 저장되어 있는 명령들을 AOF 파일에 작성하면 AOF의 로그 기록이 완료된다.

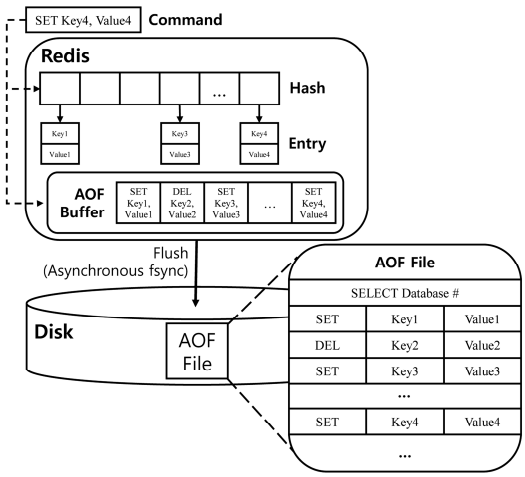

그림 3 AOF 파일 로그 기법
Fig. 3 AOF file log method

AOF 파일 로그 기법은 fsync 함수를 호출하는 정책에 따라서 Redis의 데이터 지속성이 결정된다. “always” 정책은 Redis의 데이터를 변경하는 명령이 요청될 때마다 fsync 함수를 호출하는 정책이고 “everysec” 정책은 1초마다 백그라운드 쓰레드에서 fsync 함수를 호출하는 정책이다. “always” 정책은 명령이 요청될 때마다 디스크에 로그를 작성하므로 데이터 지속성이 보장되지만, 매번 디스크 연산이 수행되므로 Redis의 처리 성능이 극도로 저하된다. 그에 반해 “everysec” 정책은 1초마다 백그라운드 쓰레드에서 디스크에 로그를 작성하기 때문에 Redis의 처리 성능이 저하되지 않지만, 시스템이 비정상적으로 종료되면 AOF 버퍼에 기록된 로그들이 손실되기 때문에 데이터 지속성이 보장되지 않는다.

AOF 파일은 Redis에 요청되는 명령을 로그에 지속적으로 작성하기 때문에, 그 크기가 무한정 커지게 된다. Redis는 이 문제를 방지하기 위하여 AOF 재작성(AOF-rewrite) 작업을 주기적으로 수행한다. AOF 재작성은 AOF 파일의 크기가 일정 이상 커지면, Redis에 들어오는 요청을 잠시 지연하고, 현재 Redis에 저장되어 있는 데이터를 새로운 AOF 파일에 작성한다. 새로운 AOF 파일 작성이 완료되면 기존의 AOF 파일을 대체한다. AOF 재작성이 수행되는 동안 Redis에 들어오는 명령들이 지연되므로, Redis의 처리 성능을 일시적으로 굉장히 저하시키는 요인이 된다.

2.4 PMDK와 Redis

인텔에서는 NVRAM을 조작하는 인터페이스 라이브러리 Persistent Memory Development Kit(PMDK) API를 제작하여 오픈소스로 제공하고 있다. 직접 접근 방식(Direct Access, DAX)을 사용하는 파일 시스템으로 NVRAM을 마운트하면, PMDK API를 사용하는 응용 프로그램(application)은 NVRAM에 있는 파일을 특정 가상 메모리 주소에 사상화(memory-mapping)하여 DRAM에 있는 데이터에 접근하는 것처럼 접근할 수 있다. PMDK API는 이러한 기능을 인터페이스 함수들로 손쉽게 사용할 수 있도록 지원하고 있다.

인텔에서는 PMDK API를 사용하여 Redis의 모든 키-값 데이터들을 모두 NVRAM에 저장하는 시스템 모델, PMDK-Redis를 제안했다. PMDK-Redis의 구조는 그림 4와 같다. 키-값 데이터들은 NVRAM에 리스트 형태로 유지하고 해쉬 테이블은 기존의 Redis와 동일하게 DRAM에 구축한다. 해쉬 테이블에서 관리하고 있는 데이터 엔트리들은 NVRAM에 저장되어 있는 키-값 데이터의 주소를 저장한다. 이후 시스템이 비정상적으로 종료되면 PMDK-Redis는 NVRAM에서 유지하고 있는 리스트를 통해 DRAM에 해쉬 테이블을 다시 복구한다.

PMDK-Redis는 모든 데이터를 NVRAM에 저장하므로 RDB 혹은 AOF 같은 별도의 로그 기법이 필요없다. 그 때문에 PMDK-Redis는 디스크 연산에 의한 성능 저하가 발생하지 않는다. 하지만 NVRAM은 아직 DRAM보다 가격 대비 사용할 수 있는 용량에 제한이 있다는 단점이 있다. PMDK-Redis는 가지고 있는 모든 키-값 데이터를 NVRAM에 저장하기 때문에, 키-값 데이터 누적양이 증가할수록 NVRAM의 사용 용량이 매우 커진다. NVRAM은 용량대비 가격이 비싸므로, PMDK-Redis와 같이 모든 키-값 데이터를 NVRAM에 저장하는 모델은 가격 비 효율성을 가진다.

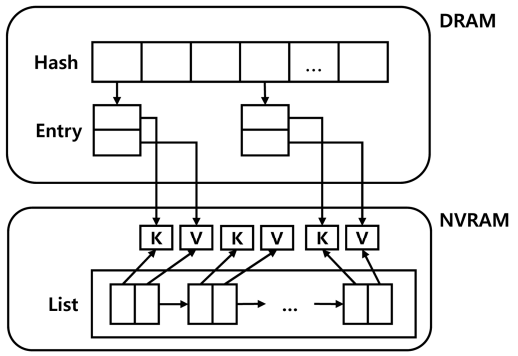

그림 4 PMDK Redis의 구조
Fig. 4 Structure of PMDK Redis

3. 시스템 설계

3.1 Persistent-Buffer Redis

AOF-everysec 정책은 Redis의 처리 성능을 저하시키지 않지만, 데이터 지속성이 손실되는 문제가 있다. 그리고 AOF-always 정책은 데이터 지속성을 보장하지만 Redis의 처리 성능을 저하시키는 문제점이 있다. 마지막으로 PMDK-Redis는 데이터 지속성이 보장되고 Redis의 처리 성능이 준수하지만, 저장할 수 있는 키-값 데이터양에 제한이 존재한다.

본 논문에서는 AOF-everysec 정책을 사용하는 Redis에서 AOF 버퍼를 PMDK를 통해 NVRAM에 구축함으로서, Redis의 처리 성능을 저하하지 않고 데이터 지속성도 보장하는 Persistent-Buffer Redis(PB-Redis)를 제안한다. PB-Redis는 그림 5와 같은 구조로 구현되어 있다. 먼저 데이터를 변경하는 명령이 Redis에 요청되면 DRAM에 저장되어 있는 해쉬 테이블의 데이터를 변경한다. 데이터 변경이 완료된 후, 요청된 명령을 NVRAM에 리스트 형태로 추가한다. 1초 뒤에 fsync 함수가 호출되어 AOF 파일에 로그가 작성되면 NVRAM에 새로운 AOF 버퍼를 생성하고 기존의 AOF 버퍼는 백그라운드 쓰레드에서 제거한다.

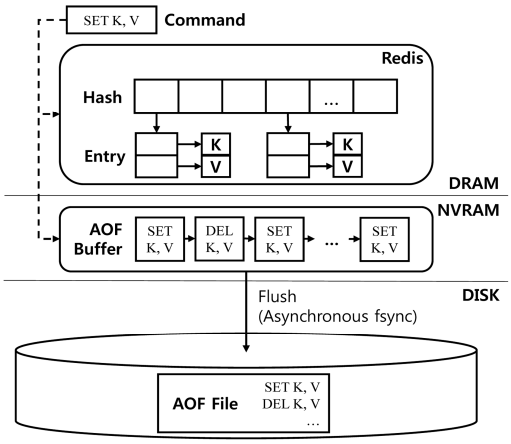

그림 5 Persistent-Buffer Redis의 구조
Fig. 5 Structure of Persistent-Buffer Redis

수행된 명령들을 기록한 로그들은 fsync로 디스크에 완전히 작성되기 전까지 NVRAM에 구축되어 있는 AOF 버퍼에서 유지하고 있기 때문에, 시스템이 비정상적으로 종료되어도 Redis의 데이터는 유실되지 않는다. PB-Redis에서는 Redis의 AOF-everysec 로그 정책처럼 1초마다 fsync 함수를 백그라운드 쓰레드에서 호출하므로 Redis의 처리 속도를 저하시키지 않는다. 또한 AOF 버퍼만을 NVRAM에 저장하고 로그가 디스크에 작성되면 버퍼를 삭제하기 때문에 NVRAM 사용량이 매우 적다. NVRAM의 용량 대비 가격을 고려했을 때, PB-Redis를 사용하는 경우, 적은 비용으로도 운용이 가능하다는 것을 알 수 있다.

3.2 복구

PB-Redis에서 시스템이 비정상적으로 종료되어 해쉬 테이블에서 유지하고 있는 데이터들이 삭제되는 경우, 디스크에 저장된 AOF 파일과 NVRAM에서 유지하고 있는 AOF 버퍼를 사용하여 해쉬 테이블을 복구한다. 알고리즘 1에서는 PB-Redis의 복구 과정을 수도코드(Pseudo-Code)로 설명한다. 먼저 알고리즘 1.1에서 디스크에 저장된 AOF 파일을 읽은 후, 프로그램이 이해할 수 있는 명령으로 파싱(parsing)한다. 파싱 단계를 거쳐 재구성된 명령은 PB-Redis로 다시 수행 요청이 되며, 이를 통해 유실된 데이터를 해쉬 테이블에 복구하게 된다. 디스크에 저장된 AOF 파일 내용을 복구하고 나면, 알고리즘 1.2에서 NVRAM에 있는 AOF 버퍼를 복구한다. NVRAM에서 유지되고 있는 AOF 버퍼는 리스트 형태로 로그를 저장하고 있기 때문에 리스트를 조회하며 명령들을 파싱한다. 그리고 그 명령들을 PB-Redis에서 재실행하여 해쉬 테이블에 데이터들을 복구하면, PB-Redis의 복구 작업이 모두 완료된다.

알고리즘 1 Persistent-Buffer Redis의 복구 프로세스
Alg. 1 Recovery process of PB-Redis

```
Algorithm 1 reconstructDatabase()
    Description
    Reconstructs database using AOF log file and
    NVRAM AOF Buffer.

    /* Reconstruct data from AOF log file */
1   cmds = loadAppendOnlyFile();
    While (cmd of cmds) {
      processCommand(cmd);
    }

    /* Reconstruct data from NVRAM AOF Buffer */
2   buffer = getAOFBuffer(NVRAMPool);
    head = buffer.getHead();
    While (head != NULL) {
      cmd = head.cmd;
      processCommand(cmd);
      head = head->next;
    }

    End
```

AOF 버퍼에는 저장되어 있지만, AOF 파일로 저장되지 않은 로그 데이터들은 NVRAM에서 유지된다. 만약 시스템 비정상 종료로 인하여 PB-Redis가 재실행 되는 경우, 앞서 서술한 복구 메커니즘을 통해 NVRAM에 존재하는 AOF 버퍼의 데이터도 함께 복구한다. 따라서 AOF-everysec 정책에서 발생할 수 있는 AOF 로그 데이터 유실 문제가 PB-Redis에서는 발생하지 않게 되며, 데이터 지속성을 보장할 수 있다.

4. 실 험

Redis에서 사용하고 있는 AOF-always 정책, AOF-everysec 정책, 그리고 인텔에서 제안한 PMDK-Redis와 본 논문에서 제안한 PB-Redis의 성능을 두가지 벤치마크 프로그램으로 측정하였다. 첫번째 벤치마크 프로그램은 Redis에서 기본으로 제공하고 있는 벤치마크 프로그램인 Redis-benchmark[17]를 사용하여 비교하려는 모델들의 성능을 측정하였다. 두번째 벤치마크 프로그램은 RedisLabs에서 구현한 벤치마크 프로그램인 Memtier-benchmark[18]을 사용하여 모델들의 성능을 측정하였다. 데이터 크기는 32, 128, 512, 2048, 8192Byte로 점차 증가시키며 성능 변화를 측정했다.

4.1 실험 환경

실험 환경은 표 1과 같다. CPU는 Intel i7 6700K를 사용하였고 DRAM은 64GB로 구성하였다. 그리고 AOF 로그 파일은 Seagate 3TB Barracuda ST3000DM001 하드디스크에 저장하였다. NVRAM의 크기는 인텔에서 개발한 NVRAM 에뮬레이터[19]를 사용하여 DRAM에 8GB로 설정했다.

NVRAM 에뮬레이터는 DRAM에 NVRAM을 가상으

표 1 실험 환경
Table 1 Evaluation environment

CPU	Intel i7 6700K (Quadcore, 4.0 GHz)
DRAM	64GB (2400MHz)
NVRAM	8GB (emulated in DRAM)
HDD	3TB (7200 RPM)

로 구축하는 방법이므로, 구축된 가상 NVRAM의 지연대기(latency)는 DRAM와 동일하다. 따라서 NVRAM의 정확한 지연대기 정보를 가지고 성능 측정을 하기에 어려움이 있다. 실험 4.6에서는 PMDK 인터페이스에 지연대기를 만들어내는 시뮬레이션 함수를 사용하여, 실제 NVRAM의 지연대기를 모사하고, 모사한 NVRAM의 지연대기에 따른 PB-Redis의 성능 변화를 측정한다.

4.2 Redis-benchmark 실험

Redis-benchmark 프로그램은 Redis에서 기본적으로 제공해주는 벤치마크 프로그램이다. Redis-benchmark에서는 Redis의 여러가지 명령들의 성능을 측정하는 기능을 제공한다. 본 실험에서는 AOF 로그 기법의 “always” 정책, “everysec” 정책, 인텔에서 제안한 PMDK-Redis, 그리고 본 논문에서 제안한 PB-Redis의 100,000건의 SET 명령에 대하여 초당 SET 명령 처리 횟수를 측정하였다.

그림 6을 통하여 PB-Redis의 SET 명령 성능을 AOF-everysec, PMDK-Redis와 비교한 결과, PB-Redis의 성능이 AOF-everysec에 상응하는 것을 확인할 수 있었다. PB-Redis의 SET 명령 성능은 AOF-everysec보다 평균적으로 약 1.31배만큼 느리고, PMDK-Redis보다 약 1.4배정도 느린 성능을 보였다. 반면에 AOF-always에 비해서는 약 100배 정도의 성능 향상을 가지는 것을 보였다. AOF-everysec과 AOF-always와의 비교 실험 결과, PB-Redis는 AOF-always의 장점인 데이터 지속성을 보장하면서 AOF-everysec에 상응하는 SET 명령 성능을 가지고 있음을 보인다. PMDK-Redis와의 실험 비교 결과는 PB-Redis가 PMDK-Redis와 동일하게 데이터 지속성을 보장하고, 크게 뒤처지지 않는 SET 명령 성능을 가지고 있음을 보인다. PB-Redis는 AOF-everysec, PMDK-Redis에 비해 약 1.25배정도 느린 성능을 보였지만, AOF-always에 비해 약 175.49배 더 빠른 성능을 보였다.

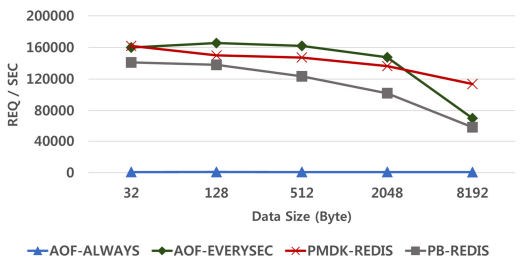

그림 6 Redis-benchmark 실험 결과
Fig. 6 Evaluation result of Redis-benchmark

4.3 데이터 지속성 실험

본 실험에서는 Redis가 비정상적으로 종료되고 다시 복구되었을 때, Redis의 로그 종류와 구현 종류에 따라서 데이터 지속성이 지켜지는지 확인한다. 먼저, 60,000개의 독립적인 키-값 데이터를 저장하고 Redis를 강제 종료한다. 그 다음 Redis를 다시 복구하고, 복구된 키-값 데이터와 손실된 키-값 데이터의 비율을 산출한다.

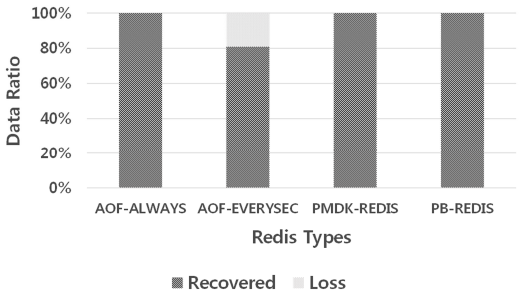

그림 7 데이터 지속성 실험 결과
Fig. 7 Evaluation result of data consistency

그림 7을 통하여 AOF-everysec의 데이터 지속성이 다른 Redis 타입에 비해서 불완전한 것을 확인할 수 있었다. 시스템이 비정상적으로 종료되고 복구되었을 때, AOF-everysec에 저장된 데이터 중, 최대 20%만큼의 손실이 발생하는 것을 확인할 수 있었다. 이는 AOF-everysec의 fsync 호출 방법에 기인한다. AOF-everysec은 1초마다 백그라운드 쓰레드에서 fsync 함수를 호출하여 로그를 작성한다. 그 때문에 fsync가 호출되기 전에 Redis가 강제종료되어, 그 사이에 요청된 데이터들의 지속성을 보장하지 못하였다. 반면에 AOF-always는 명령이 요청될 때마다 fsync 함수를 호출하여 데이터 지속성을 유지하였고, PMDK-Redis와 PB-Redis는 NVRAM으로 데이터 지속성을 유지하였다.

4.4 DRAM-NVRAM 사용용량 실험

본 실험에서는 AOF-always, AOF-everysec, PMDK-Redis, PB-Redis가 사용하는 DRAM과 NVRAM의 사용 용량을 측정하였다. 벤치마크 프로그램으로 Redis-benchmark를 사용하였고, 중복되지 않는 키-값 데이터를 저장하였다. 저장하는 값 데이터 사이즈는 32Byte로 동일하게 설정하였고, 키-값 데이터의 개수는 100,000개, 500,000개, 1,000,000개, 2,000,000개로 확장하여 사용 용량을 측정하였다.

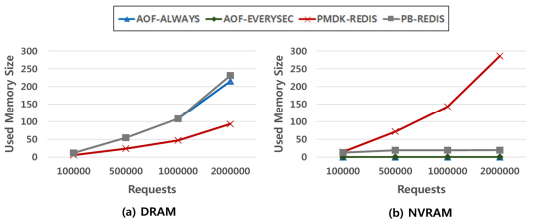

그림 8 DRAM-NVRAM 사용 용량 실험 결과
Fig. 8 Evaluation result of DRAM-NVRAM usage

그림 8을 통하여 각 Redis 타입에서 사용하는 DRAM, NVRAM 사용 용량을 측정한 결과, PB-Redis가 사용하는 NVRAM 사용 용량이 PMDK-Redis에 비하여 월등히 적고 일정한 것을 확인하였다. PB-Redis에 저장되는 데이터는 DRAM에 저장되므로 DRAM의 사용 용량은 AOF-everysec과 동일하게 증가하였지만, NVRAM에는 AOF 버퍼에 임시로 저장되는 로그들만 저장되므로 PMDK-Redis에 비하여 NVRAM 사용 용량이 굉장히 적고 일정하다. PB-Redis는 100,000개의 데이터를 저장할 때 PMDK-Redis과 동일한 NVRAM 사용 용량을 보였지만, 2,000,000개의 데이터를 저장할 때에는 PMDK-Redis의 NVRAM 사용 용량에 비하여 0.06배 가량 감소하는 것을 확인할 수 있었다.

4.5 Memtier-benchmark 실험

Memtier-benchmark 프로그램은 RedisLabs에서 구현한 벤치마크 프로그램이다. Memtier-benchmark에서는 Redis-benchmark보다 더 다양한 측정 기능을 제공한다. 첫번째로 SET과 GET이 번갈아 발생하는 상황에서의 성능을 측정할 수 있다. 두번째로 여러 개의 Redis를 다중 클러스터로 운용하는 상황에서의 성능을 측정할 수 있다. 마지막으로 정해진 사이즈가 아닌, 특정 범위의 사이즈를 가진 데이터가 저장되는 상황에서의 성능을 측정할 수 있다. 본 실험에서는 Redis-benchmark 실험과 같이 기존에 제시한 방법들과 PB-Redis와의 초당 명령 처리 성능을 측정하고 비교했다. 이에 더하여 Memtier-benchmark를 이용한 실험에서는 SET과 GET의 비율을 변경하며 실험을 진행하고 성능을 측정했다.

그림 9를 통하여 Redis-benchmark 결과와 비슷하게 SET과 GET이 번갈아 발생하는 상황에서도 PB-Redis는 AOF-everysec, PMDK-Redis에 필적하는 성능을 보이는 것을 확인할 수 있었다. 평균적으로 PB-Redis는 AOF-everysec, PMDK-Redis에 비해 약 1.54배정도 느린 성능을 가진 반면, AOF-always에 비해 약 34배 정도의 성능 향상을 가지는 것을 보였다. Redis-benchmark 실험과 Memtier-benchmark 실험 모두 데이터 사이즈가 8192B일 때 PB-Redis의 성능이 급격하게 감소하였다. 이때 PB-Redis는 AOF-everysec보다 약 3.77배, PMDK-Redis보다 약 12.86배 가량 성능이 저하되었다.

데이터 사이즈가 8192B일 때, PB-Redis의 성능이 급격하게 감소한 이유는, AOF 재작성(AOF rewrite)이 빈번하게 발생하기 때문인 것을 확인하였다. 데이터 사이즈가 커질수록 AOF 로그의 사이즈도 커지기 때문에, AOF 재작성의 수행 횟수가 많아진다. AOF 재작성이 빈번하게 발생하면 Redis의 전체적인 처리 성능이 저하되므로, AOF 재작성 문제에 대한 해결이 필요하다.

4.6 NVRAM 지연대기 실험

NVRAM은 그 종류마다 각기 다른 지연대기를 가지고 있다[4], [16]. DRAM과 NVRAM 종류들의 대략적인 지연대기는 표 2와 같다. 먼저 PCM은 DRAM과 읽기 지연대기는 같지만, 쓰기 지연대기는 500ns로 NVRAM 중에서 가장 느린 지연대기를 가지고 있다. 이는 PCM에 데이터를 작성할 때, 데이터를 지우는 RESET 작업은 굉장히 빠르지만 데이터를 작성하는 SET 작업에서 부하가 발생하기 때문이다. 두번째로 3D XPoint(Memory Mapped)는 읽기, 쓰기 지연대기가 모두 100ns로 준수한 성능을 가지고 있다. 하지만 3D XPoint에서 Memory Mapped 방식은 메모리의 비 휘발성을 완전히 보장하지 못한다. 마지막으로 3D XPoint(Storage Mapped)는 Memory Mapped 방식의 3D XPoint보다 읽기, 쓰기 지연대기가 2배 높지만 메모리의 비 휘발성을 완전히 보장한다.

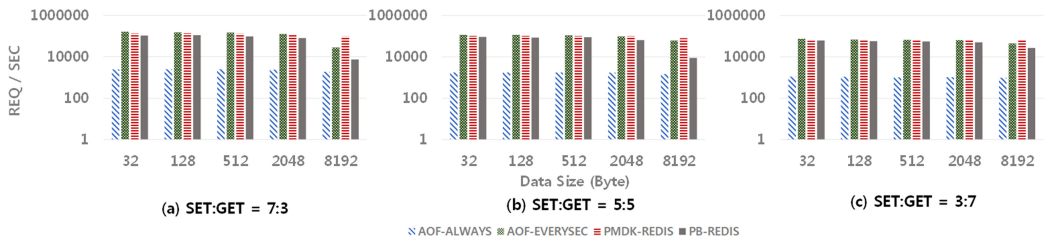

그림 9 Memtier-benchmark 실험 결과
Fig. 9 Evaluation result of Memtier-benchmark

표 2 NVRAM 지연대기
Table 2 Latency of NVRAM

	Read (ns)	Write (ns)
DRAM	50	50
PCM	50	500
3D XPoint (Memory Mapped)	100	100
3D XPoint (Storage Mapped)	200	200

본 실험에서는 NVRAM의 종류 변화에 따른 PB-Redis의 성능 변화를 비교했다. PMDK에서 제공하는 NVRAM 에뮬레이터는 지연대기 옵션을 줄 수 없으므로 알고리즘 2와 같이 PMDK 인터페이스에 별도의 래퍼 함수를 구현하여 가상으로 지연대기를 설정했다. 먼저 알고리즘 2.1에서는 사전에 설정한 지연대기 시간만큼 빈 루프문(While Loop Statement)을 실행한다. 그리고 알고리즘 2.2에서는 PMDK의 인터페이스 함수를 실행한다. 가상으로 구현한 지연대기 옵션이므로 실제 NVRAM 디바이스의 지연대기와 완전히 동일하지 않지만, 지연대기 옵션에 따른 PB-Redis의 성능을 근사적으로 비교할 수 있다. 지연대기 실험은 각 NVRAM 지연대기 설정에서 4.2 Redis-benchmark 실험과 동일하게 데이터 사이즈를 32B, 128B, 512B, 2048B, 8192B로 변경하며 초당 SET 명령 요청 횟수를 측정하였다.

PB-Redis에서 다양한 NVRAM 환경을 가정하여 성능을 측정한 결과를 그림 10을 통하여 확인하였다. PCM은 기존의 DRAM에 비해 약 1.52배, 3D XPoint (Memory Mapped)는 약 1.14배, 3D XPoint(Storage Mapped)는 약 1.31배 느린 처리 속도를 보였다. PCM은 나열된 NVRAM 중에서 가장 높은 쓰기 지연대기를 가지고 있기 때문에 모든 데이터에서 느린 성능을 보였다. 3D XPoint(Memory Mapped)는 DRAM에 가장 근접한 지연대기를 가지고 있기 때문에 NVRAM 중에서

알고리즘 2. NVRAM 지연대기 시뮬레이션
Alg. 2 Latency simulation of NVRAM

```
Algorithm 2 pmemobj_direct_latency()
    Description
    Adds read latency to 'pmemobj_direct()' interface
    function.

    /* Run empty loop until the specified latency */
1   time target = nstime() + pm_read_latency;
    While (target > nstime()) {
        /* Empty */
    }

    /* Run 'pmemobj_direct()' function */
2   return pmemobj_direct();

    End
```

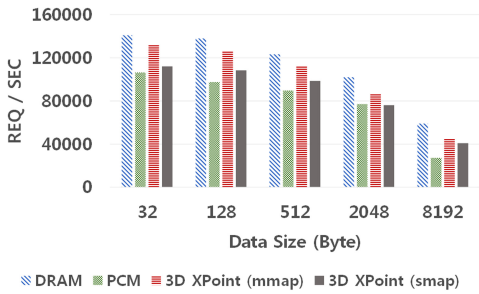

그림 10 PB-Redis의 NVRAM 지연대기 실험 결과
Fig. 10 Evaluation result of NVRAM latency for PB-Redis

가장 우수한 처리 성능을 보였다. 마지막으로 3D XPoint (Storage Mapped)는 Memory Mapped 방식의 3D XPoint보다는 느리지만 PCM보다 더 우수한 처리 성능을 보였다. 실험에 사용된 NVRAM중에서 3D XPoint (Memory Mapped)가 가장 우수한 처리 성능을 보였지만 비 휘발성이 불완전한 특성 때문에 데이터 지속성을 보장할 수 없다.

5. 결론 및 향후 연구

Redis 로그 방식 중 하나인 AOF 로그 방식은 정책에 따라 장단점이 존재한다. AOF-everysec 정책은 Redis의 처리 성능을 저하시키지 않지만 데이터 지속성을 보장하지 못하는 문제가 있다. AOF-always 정책은 데이터 지속성을 보장하지만 Redis의 처리 성능을 굉장히 저하시켰다. 그리고 인텔에서 제안한 PMDK-Redis는 성능을 저하하지 않고 데이터 지속성을 보장하지만, NVRAM의 비싼 가격을 고려했을 때, 가격 대비 용량 한계를 극복하기에는 부족함이 많다.

본 논문에서는 AOF-everysec, AOF-always, PMDK-Redis의 장점만을 취하고, 작은 양의 NVRAM을 사용하면서도 빠른 처리 속도와 데이터 지속성을 보장할 수 있는 로그 기법인 PB-Redis를 제안하였다. 그리고 실험을 통하여 여러 상황에서 AOF-everysec에 준하는 성능을 보이는 것을 확인하였다.

하지만 데이터 삽입이 지속되어 AOF 파일의 크기가 매우 증가될 경우, 성능이 굉장히 저하되는 현상을 발견하였다. 이는 AOF 파일의 크기가 커지면 AOF 재작성 작업이 수행되는데, 이 작업이 데이터 사이즈가 커질수록 빈번하게 발생하기 때문에 성능이 저하된다는 것을 확인하였다. 향후 연구로서 AOF 파일을 다시 작성하는 상황에서도 NVRAM을 활용하여 성능을 높일 수 있는 연구를 하고자 한다. Redis는 새로운 자식 프로세스를 생성하여 AOF 재작성을 수행하는데, 그 동안 Redis에 새로운 명령들이 요청되면 로그 간의 동기화를 위하여

새로운 명령들의 처리를 지연시킨다. AOF 재작성 중에 요청된 명령들의 로그를 NVRAM에 저장하고 AOF 재작성 작업이 끝난 후에 NVRAM과 새로운 AOF 파일 간의 동기화를 진행한다면, AOF 재작성에 의한 Redis의 처리 속도 저하를 해결할 수 있을 것으로 기대한다.

References

[1] J. Han, H. E, G. Le, and J. Du, "Survey on NoSQL database," *Pervasive computing and applications (ICPCA), IEEE 2011 6th international conference*, pp. 363-366, Oct. 2011.

[2] RedisLabs, 2018, Introduction to Redis, [Online]. Available: https://redis.io/topics/introduction

[3] RedisLabs, 2018, Redis Persistence, [Online]. Available: https://redis.io/topics/persistence

[4] S. Mittal and J. S. Vetter, "A survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems," *IEEE Transactions on Parrallel and Distributed Systems*, Vol. 27, No. 5, pp. 1731-1733, May. 2016.

[5] Y. Son, H. Kang, H.Y. Yeom, and H. Han, "A log-structured buffer for database systems using non-volatile memory," *SAC'17 Proceedings of the Symposium on Applied Computing*, pp. 880-886, Apr. 2017.

[6] R. Fang, H. Hsiao, B. He, C. Mohan, and Y. Wang, "High performance database logging using storage class memory," *ICDE'11 Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, pp. 1221-1231, Apr. 2011.

[7] S. Lee, K. Lim, H. Song, B. Nam, and S. Noh, "WORT: Write Optimal Radix Tree for Persistent Memory Storage Systems," *FAST' 17 Proceedings of the 15th Usenix Conference on File and Storage Technologies*, pp. 257-270, Mar. 2017.

[8] P. Zuo and Y. Hua, "A Write-friendly and Cache-optimized Hashing Scheme for Non-volatile Memory Systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 29, No. 5, pp. 985-998, May. 2018.

[9] E. Lee, H. Bahn, and S.H. Noh, "Unioning of the Buffer Cache and Journaling Layers with Non-volatile Memory," *FAST'13 Proceedings of the 11th USENIX Conference on File and Storage Technologies*, pp. 73-80, Feb. 2013.

[10] J. Arulraj, M. Perron, and A. Pavlo, "Write-Behind Logging," *Journal Proceedings of the VLDB Endowment*, pp. 337-348, Nov. 2016.

[11] W.H. Kim, J. Kim, W. Baek, B. Nam, and Y. Won, "NVWAL: Exploiting NVRAM in Write-Ahead Logging," *ASPLOS'16 Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 385-398, Apr. 2016.

[12] L. Lersch, I. Oukid, W. Lehner, and I. Schreter, "An analysis of LSM caching in NVRAM," *DAMON'17 Proceedings of the 13th International Workshop on Data Management on New Hardware*, Article No. 9, May. 2017.

[13] J. Arulraj and A. Paylo, "How to Build a Non-Volatile Memory Database Management System," *SIGMOD '17 Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 1753-1758, May. 2017.

[14] Intel, 2018, Persistent Memory Development Kit, [Online]. Available: https://github.com/pmem/pmdk

[15] Intel, 2018, PMDK Implementation Redis, [Online]. Available: https://github.com/pmem/redis

[16] P. Cappelletti, "Non volatile memory evolution and revolution," *IEEE International Electron Devices Meeting (IEDM)*, pp. 10.1.1-10.1.4, Dec. 2015.

[17] RedisLabs, 2018, Redis-benchmark, [Online]. Available: https://redis.io/topics/benchmarks

[18] RedisLabs, 2018, Memtier-benchmark, [Online]. Available: https://github.com/RedisLabs/memtier_benchmark

[19] S. R. Dullor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," *EuroSys'14 Proceedings of the 9th European Conference on Computer Systems*, Article No. 15, Apr. 2014.

김 도 영

2018년 연세대학교 컴퓨터과학과(학사) 2018년~현재 연세대학교 컴퓨터과학과 석사과정. 관심분야는 데이터베이스 시스템, 빅데이터, 분산처리 시스템

최 원 기

2014년 연세대학교 컴퓨터과학과(학사) 2014년~현재 연세대학교 컴퓨터과학과 석박사통합과정. 관심분야는 데이터베이스 시스템, 빅데이터, 분산처리 시스템

성 한 승

2017년 항공대학교 소프트웨어학과(학사). 2017년~현재 연세대학교 컴퓨터과학과 석박사통합과정. 관심분야는 데이터베이스 시스템, 키-값 데이터베이스, 리커버리

이 지 환

2017년 연세대학교 컴퓨터과학과(학사) 2017년~현재 연세대학교 컴퓨터과학과 석사과정. 관심분야는 데이터베이스 시스템, 기계 학습

박 상 현

1989년 서울대학교 컴퓨터공학과 졸업(학사). 1991년 서울대학교 대학원 컴퓨터공학과(공학석사). 2001년 UCLA 대학원 컴퓨터과학과(공학박사). 1991년~1996년 대우통신 연구원, 2001년~2002년 IBM T. J. Watson Research Center Post-Doctoral Fellow. 2002년~2003년 포항공과대학교 컴퓨터공학과 조교수. 2003년~2006년 연세대학교 컴퓨터과학과 조교수. 2006년~2011년 연세대학교 컴퓨터과학과 부교수. 2011년~현재 연세대학교 컴퓨터과학과 교수. 관심분야는 데이터베이스, 데이터 마이닝, 바이오인포매틱스, 빅데이터 마이닝 & 기계 학습