https://doi.org/10.5392/JKCA.2019.19.05.187

키-값 저장소를 위한 효율적인 로그 처리 기법 설계 및 평가

Design and Evaluation of an Efficient Flushing Scheme for key-value Store

한혁
동덕여자대학교 컴퓨터학과

Hyuck Han(hhyuck96@dongduk.ac.kr)

요약

키-값 스토리지 엔진은 소셜 네트워크, 온라인 전자 상거래 환경 및 클라우드 서비스를 포함한 많은 컴퓨팅 환경에서 점점 더 수요가 증가하고 있는 필수적인 구성 요소다. 최근 키-값 스토리지 엔진은 트랜잭션, 버전 관리 및 복제를 비롯한 많은 기능을 제공한다. 키-값 스토리지 엔진에서 트랜잭션 처리는 로그 선행 기입을 사용하여 원자성을 제공하며, 동기식 커밋 방식에서는 트랜잭션이 완료되기 전에 로그 데이터를 플러시한다. 그러나 로그 선행 기입에서 로그 데이터를 저장 장치로 플러시하는 것은 다양한 최적화 기법이 제안되었음에도 불구하고 여전히 fsync() 호출에 큰 오버헤드가 존재하고 있기 때문에 키-값 스토리지 엔진의 성능 병목이다. 이 논문에서는 기존 플러싱 체계를 최적화하기 위해 그룹 동기화 기법을 제안하여 키-값 스토리지 엔진의 성능을 개선한다. 또한, fsync()를 수행하는 동안에 다른 트랜잭션을 수행하는 트랜잭션 스케줄링 기법을 제안한다. 이 체계는 기존 시스템이 제공하는 동일한 트랜잭션 수준을 지원하면서 fsync() 호출의 수를 줄이는 효율적인 방법이다. 우리는 WiredTiger 스토리지 엔진에 제안하는 방법을 구현하였다. 실험 결과는 제안된 시스템이 기존 시스템에 비해 키-값 워크로드의 성능을 향상시킨다는 것을 보여준다.

■ 중심어 : | 키-값 스토리지 엔진 | 로그 선행 기입 | 동기화 연산 |

Abstract

Key-value storage engines are an essential component of growing demand in many computing environments, including social networks, online e-commerce, and cloud services. Recent key-value storage engines offer many features such as transaction, versioning, and replication. In a key-value storage engine, transaction processing provides atomicity through Write-Ahead-Logging (WAL), and a synchronous commit method for transaction processing flushes log data before the transaction completes. According to our observation, flushing log data to persistent storage is a performance bottleneck for key-value storage engines due to the significant overhead of fsync() calls despite the various optimizations of existing systems. In this article, we propose a group synchronization method to improve the performance of the key-value storage engine. We also design and implement a transaction scheduling method to perform other transactions while the system processes fsync() calls. The proposed method is an efficient way to reduce the number of frequent fsync() calls in the synchronous commit while supporting the same level of transaction provided by the existing system. We implement our scheme on the WiredTiger storage engine and our experimental results show that the proposed system improves the performance of key-value workloads over existing systems.

■ keyword : | Key-value Storage Engine | Write-ahead Logging | Synchronization |

* 이 논문은 2018년도 동덕여자대학교 학술연구비 지원에 의하여 수행된 것임.
접수일자 : 2019년 02월 08일
심사완료일 : 2019년 04월 30일
수정일자 : 2019년 04월 30일
교신저자 : 한혁, e-mail : hhyuck96@dongduk.ac.kr

I. 서 론

키-값 스토리지 엔진은 소셜 네트워크, 온라인 전자 상거래 및 클라우드 서비스를 포함한 많은 스케일 아웃 컴퓨팅 환경에서 중요한 역할을 한다[1]. 특히 최근 많은 클라우드 서비스 업체들은 키-값 스토리지 엔진을 (예: Amazon의 DynamoDB[2], GitHub의 Redis[3], Facebook의 Memcached[4], Linkedin의 Voldemort[5]) 주요 컴퓨팅 소프트웨어 컴포넌트로 채택하였다. 따라서 키-값 스토리지 엔진의 중요성은 점점 강조되었다. 이에 키-값 스토리지 엔진은 높은 성능, 확장성 및 신뢰성을 제공하기 위해 개선되었고, 복제, 버전 설정, 잠금 및 트랜잭션을 포함한 다양한 기능이 최신 키-값 시스템에서 수행될 수 있다.

현대의 키-값 스토리지 엔진은[6][7] 데이터 페이지가 원래 데이터 위치에 기록되기 전에 로그 영역에 로그 데이터를 쓰는 로그 선행 기입(WAL/Write-Ahead Logging)을[8] 사용하여 트랜잭션 속성(예: 원자성)을 지원한다. 트랜잭션은 커밋 절차 중에 fsync() 시스템 호출을 실행하여 로그 데이터를 저장 장치로 플러시한다. 그러나 fsync() 호출이 로그 데이터를 포함하는 파일 시스템의 변경된 페이지들을 플러시하면서 오버헤드가 발생한다. 일부 과거의 연구는 커밋 및 플러싱 작업과 관련된 오버헤드 문제를 다루었다. 예를 들어 그룹 커밋은[9] 여러 커밋 요청을 단일 작업으로 그룹화하여 처리하지만 로그 쓰기 및 동기화 연산으로 인한 오버헤드는 여전히 크다. 비동기식 커밋은[10] 로그 쓰기 요청이 완료될 때까지 기다리지 않고 트랜잭션을 완료할 수 있도록 하지만 시스템의 장애가 발생한다면 사용자에게 완료되었다고 보고한 트랜잭션이 손실될 수 있다.

이 논문는 트랜잭션 처리에서 기존의 플러싱 체계를 최적화하여 키-값 스토리지 엔진의 성능을 개선하기 위한 효율적인 방법을 제안한다. 본 논문의 분석은 현재의 방법들이 트랜잭션이 로그 데이터를 플러시하기 위해 그룹 커밋 및 통합-어레이 기법에 의해 이미 개선되었더라도 빈번한 fsync() 호출을 초래함을 보여준다. 본 논문의 핵심적인 기법은 i) 한 번의 fsync() 호출에 가능한 많은 트랜잭션의 로그 데이터를 플러시하는 것과 ii) 로그 데이터가 플러시 되는 동안에 스토리지 엔진 쓰레드가 다른 트랜잭션을 수행하여 키-값 스토리지 엔진의 성능을 향상시키는 것이다.

WiredTiger 스토리지 엔진은 ACID 트랜잭션을 포함하고 데이터 관리를 위한 고성능, 확장성이 있는 오픈 소스 플랫폼이다. WiredTiger는 성능과 확장성 덕분에 많은 시스템에서 활용되고 있으며 최근에 MongoDB의 기본 저장 엔진으로 채택되었다. WiredTiger는 그룹 커밋 및 통합-어레이를[11] 채택하여 기존 로그 처리의 성능 문제를 개선하였다. 본 논문은 fsync() 호출들을 그룹하여 처리하고, 로그 데이터가 플러시 되는 동안에 다른 트랜잭션을 처리하도록 개선하였다. 고성능 SSD를 탑재한 서버 시스템에서 수행한 실험 결과는 제안된 방법이 기존 방식에 비해 성능을 향상시킨다는 것을 보여준다.

본 논문의 기여는 다음과 같다.

- WiredTiger의 로그 처리 방법을 분석하였고 fsync() 호출의 오버헤드를 분석하였다.
- 트랜잭션의 로그 처리 성능을 개선하기 위한 효율적인 최적화 기법들을 제안하고 WiredTiger에 구현하였다.
- 성능 평가를 통해 개선된 WiredTiger가 기존 WiredTiger 보다 키-값 워크로드에서 최대 10배 정도 성능을 개선하였음을 보였다.

이 논문의 구성은 다음과 같다. 2장에서는 관련 연구와 연구 동기에 대해 설명하고 3장에서는 최적화 기법을 설명한다. 4장에서는 실험 결과를 보여주며 5장에서는 향후 연구 방향과 함께 논문의 결론을 맺는다.

II. 관련 연구 및 배경

1. 관련 연구

데이터베이스 및 키-값 스토리지 엔진의 로그 처리 오버헤드를 줄임으로써 성능을 개선하기 위한 몇 가지 연구가 있었다. 그룹 커밋은[9] 로그 처리 오버헤드를 줄이기 위해 널리 사용되는 기법이다. 이 기법은 적절한 커밋 대기 시간 동안의 여러 로그 처리 요청을 단일 I/O 작업으로 처리한다. 본 논문에서 제안하는 방법은

이 기법과 유사하지만 커밋 대기 시간이 없으며 트랜잭션 스케줄링을 통해 더 효율적으로 트랜잭션을 처리한다.

비동기식 커밋은[10] 로그 처리를 위한 I/O 요청이 완료될 때까지 기다리지 않고 트랜잭션을 완료하도록 허용함으로써 그룹 커밋을 확장한다. 이 방법은 상당한 성능 향상을 보여주며 상용 및 오픈 소스 데이터베이스 시스템에서 사용된다. 그러나 시스템의 장애가 발생할 경우 커밋된 작업을 잃을 수 있는 반면, 우리가 제안한 방법은 안전성을 희생하지 않고 성능을 개선할 수 있다.

Aether는[11] 빠른 커밋을 위해 여러 트랜잭션이 잠금으로 보호되는 로그 데이터를 로그 버퍼로 복사하는 것이 아니라 잠금 경합을 줄이는 통합-어레이라는 기법을 제안 및 구현하였다. 이 방법에서는 여러 트랜잭션 쓰레드들이 동시에 로그 데이터를 로그 버퍼에 복사하는 것이 가능하며 로그 데이터가 저장 장치에 써진 후에 트랜잭션이 종료된다. 본 논문은 Aether와는 달리 여러 로그 플러시 요청들을 하나의 fsync() 호출로 처리하고 트랜잭션 스케줄링을 통해 높은 성능을 보여준다.

상변화메모리(PCM) 및 플래시 메모리와 같은 비휘발성 메모리(NVM)를 사용하여 로그 처리 오버헤드를 줄이기 위한 여러 연구가 있다. NV-Logging은[12] 트랜잭션 시스템에서 NVM의 비용-효율적인 사용을 조사했다. 로그를 처리하는 하위 시스템에 PCM과 같은 메모리를 사용하면 모든 저장 장치를 교체하는 것보다 비용-효율적인 측면에서 더 성능이 좋아짐을 보였다. 또한 여러 트랜잭션의 동시 로깅을 지원하고 중앙 집중식 로그 버퍼로 인한 성능 병목 현상을 줄이기 위해 트랜잭션별 로깅을 제공한다. FlashLogging은[13] 동기식 로깅을 위해 여러 USB 플래시 드라이브를 활용하는 로깅 시스템 설계를 제안했다. 이는 USB 플래시 드라이브가 낮은 가격과 높은 효율성 때문에 동기식 로깅에 적합하다는 것을 보여준다. PCMLogging은[14] 디스크 기반 데이터베이스의 데이터 버퍼링과 로깅을 위해 PCM 장치를 이용하는 로깅 체계를 제안하였다. 이러한 연구들은 본 논문과 로그 처리 성능 개선이라는 측면에서 유사하지만, 본 논문과 달리 시스템 구축비용을 증가시킬 수 있는 추가적인 메모리 장치를 필요로 한다.

2. 연구 배경

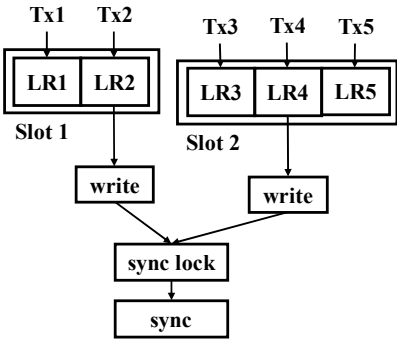

그림 1. WiredTiger의 통합 어레이 기법 (Tx : 트랜잭션, LR : 로그 레코드, sync lock : 동기화 잠금)

이번 장에서는 서버 시스템에서 WiredTiger 스토리지 엔진의 로그 플러시 체계에 대해 설명을 하고 fsync() 호출이 성능에 미치는 영향을 분석한다. 기존의 잠금으로 보호되는 중앙 로그 버퍼에 로그 레코드를 복사 처리하는 방식과는 달리 WiredTiger는 통합-어레이 방법을 이용한다. [그림 1]과 같이 통합-어레이 방법에서는 중앙 로그 버퍼가 아닌 로그 버퍼를 슬롯이라는 개념으로 분산화하여 하나의 슬롯에 여러 트랜잭션의 로그 데이터를 동시에 복사할 수 있다. 한 슬롯에 복사하고 있는 트랜잭션 쓰레드들 중에서 하나가 슬롯의 모든 로그 데이터를 저장 장치에 쓰고, 저장 장치와의 동기화를 위한 fsync() 호출은 잠금을 획득한 쓰레드만이 수행한다.

기존의 통합 어레이의 성능 평가를 위해 2개의 Intel Xeon CPU E5-2690 CPU를 장착한 서버를 사용하였다. 이 서버 시스템은 128GB 메인 메모리와 Intel의 NVMe 기반 플래시 메모리 SSD를 (DC 3700) 가지고 있다. 이 시스템에 Linux 커널 4.8.7 버전에 기반을 두는 Ubuntu 18.04 LTS를 구동하였다. 키-값 스토리지 엔진 평가에 많이 사용되고 있는 YCSB를[15] 이용하였으며, 읽기 및 쓰기 비율이 1:1인 워크로드로 설정하였다. 실험에서는 작업 쓰레드의 수를 1부터 32까지 변화시켜가면서 성능을 측정하였고 수행 시간을 분석하였다. [그림 2]는 트랜잭션 성능 결과이며 32 작업 쓰레드일 때 초당 14만개 정도의 트랜잭션을 수행할 있음을 알 수 있다.

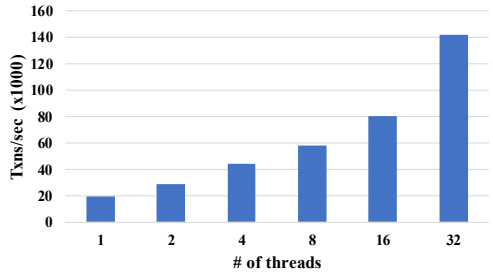

그림 2. YCSB 성능 결과

로그 처리 시 저장장치와의 fsync() 호출의 비중을 분석하기 위해 트랜잭션 수행 시간을 분석하였으며 그 결과는 [그림 3]에 표시했다. YCSB 워크로드에서 전체 트랜잭션 수행 시간에서 트랜잭션 커밋이 차지하는 비중은 90 ~ 97% 정도이며, 전체 트랜잭션 수행 시간에서 커밋시 발생하는 저장장치와의 fsync() 연산이 차지하는 비중은 78 ~ 88%로 측정되었다. 이 결과는 트랜잭션 커밋 시에 fsync() 호출이 전체 수행 시간의 가장 큰 비중을 차지하고 있고 fsync() 호출 체계의 최적화가 필수적임을 보여준다.

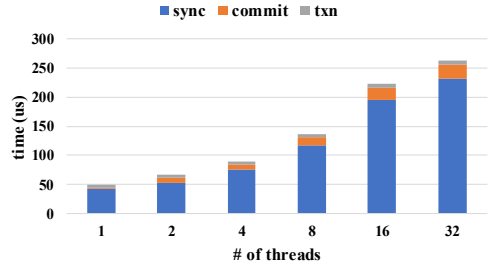

그림 3. YCSB 수행 시간 분석

III. 최적화 기법 및 성능 평가

1. 통합 동기화

이 장에서는 앞서 분석한 것과 같이 키-값 스토리지 엔진의 큰 오버헤드인 트랜잭션 커밋 시 발생하는 fsync() 호출 체계를 최적화하는 기법에 대해 설명한다. 기존의 통합-어레이 기법에서는 하나의 로그 슬롯을 처리할 때마다 잠금에 의해 보호되는 fsync() 연산을 수행하였지만 통합 동기화 기법에서는 가능한 많은 fsync() 연산을 한 번에 처리한다.

[그림 4]와 같이 세 개의 쓰레드가 (T1, T2, T3) 각각의 슬롯에 참여하고 있는 트랜잭션들의 로그 데이터를 디스크에 쓴 후에 플러시되지 않은 로그 일련 번호 LSN1, LSN2, LSN3으로 로그 레코드를 플러시한다. 이를 위해 각 쓰레드는 각 로그 일련번호를 포함하는 동기화 요청 객체를 만들어서 요청 대기열에 요청을 잠금 없이 삽입하고 동기화 잠금을 획득하려고 시도한다. 이 예제에서 T1 쓰레드가 동기화 잠금을 획득하고 동기화 요청을 집계한다. 동기화 잠금을 획득하지 못 한 다른 쓰레드(T2와 T3)들은 T1의 fsync() 수행 완료 신호를 기다린다.

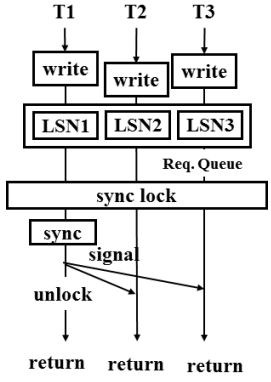

그림 4. 통합 동기화 기법 (T : 쓰레드)

이 단계가 끝나면 동기화 잠금을 획득한 쓰레드는 먼저 요청 대기열을 닫아 동기화 요청 입력을 차단한다. 이 때, 동시 트랜잭션의 요청은 대기열의 동기화 잠금을 획득한 쓰레드에 의해 닫힐 때까지 집계된다. 다음으로, 저장장치와 동기화를 위해 fsync() 연산을 수행하고 연산이 끝나면 대기하고 있는 다른 쓰레드들에게 완료 신호를 보낸다. 그리고 동기화 잠금을 해제하여 다음 통합 동기화 작업을 수행할 수 있도록 한다. 이러한 방법은 빈번한 로그 플러싱을 최대로 감소시켜 fsync() 호출 횟수와 동기화 잠금 획득의 수를 줄이기 때문에 트랜잭션 처리 성능이 향상된다.

2. 트랜잭션 스케줄링 최적화

이 장에서는 동기화 연산이 수행되는 동안에 쓰레드들이 동기화 연산이 끝나는 것을 기다리는 것이 아니라 다른 트랜잭션을 수행하도록 하는 트랜잭션 스케줄링에 대해 설명한다. 이 기법에서는 다음과 같은 흐름으로 트랜잭션 처리를 진행한다. 먼저 앞에서 설명했던 것과 동일하게 작업 쓰레드들이 동기화 요청 객체를 만들어서 요청 대기열에 요청을 삽입하고 동기화 잠금 획득을 시도한다. 동기화 잠금을 획득한 쓰레드가 동기화 요청을 집계한 후에 별도의 동기화 요청만 처리하는 쓰레드에 (동기화 쓰레드) 집계된 동기화 요청 객체들을 전달한다. 동기화 쓰레드는 전달받은 동기화 요청 객체들을 한 번에 처리한다. 이와 별도로 작업 쓰레드들은 현재 수행 중인 트랜잭션을 완료로 표시하지 않고 다른 트랜잭션을 수행하도록 한다. 즉, 기존에 수행 중이던 트랜잭션은 완료로 표시되지 않은 상태로 별도의 쓰레드 로컬 메모리 공간에 저장하고 해당 트랜잭션을 종료하지 않은 상태에서 다른 트랜잭션을 수행한다. 따라서 기존 트랜잭션의 수행 결과가 사용자에게 보고되지 않으므로 트랜잭션의 안전성을 훼손하지 않는다. 동기화 쓰레드가 집계된 동기화 요청을 처리한 후에 해당 트랜잭션들을 완료 상태로 표시하고 작업 쓰레드들은 완료 상태로 변경된 트랜잭션들을 종료하고 사용자에게 결과를 보고하여 트랜잭션 처리를 마친다.

[그림 5]는 최적화된 트랜잭션 스케줄링의 예를 보여준다. 작업 쓰레드 T1, T2는 Tx1, Tx2 트랜잭션을 수행하고 있으며, T1, T2 쓰레드들은 동시에 로그 버퍼에 있는 로그들을 저장장치에 쓴다. 쓰레드 중에 T1 쓰레드가 동기화 잠금을 얻고 T1 쓰레드가 T1, T2의 동기화 요청 객체를 집계하여 별도의 동기화 쓰레드에 전달한다. 동기화 쓰레드는 동기화 연산을 수행하며 동기화 연산이 수행하는 동안에 T1, T2 쓰레드는 다른 트랜잭션인 Tx3, Tx4를 수행한다. 동기화 쓰레드에 의해 동기화 연산이 완료된 후에 T1, T2는 완료 상태로 변경된 Tx1과 Tx2의 완료 처리를 끝낸다.

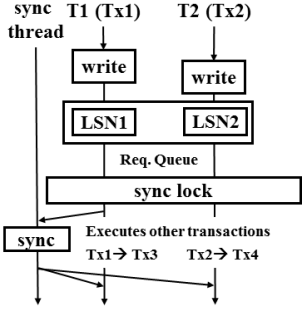

그림 5. 최적화된 스케줄링 기법

IV. 성능 평가

제안된 기법을 평가하기 위해 2장에서 사용하였던 서버 시스템을 사용하였다. 실험을 위하여 YCSB와 Sysbench의 OLTP 벤치마크를 사용하였다. WiredTiger의 트랜잭션 고립 수준을 read- uncommitted로, 스토리지 엔진의 캐시 크기를 16 GB로 설정하여 실험을 진행하였다. 작업 쓰레드의 수는 1부터 32까지 변화시켜가면서 성능을 측정하였다. 모든 그래프의 vanilla는 기존의 WiredTiger를, CS는 통합 동기화 기법이 구현된 WiredTiger를, CS+TS는 통합 동기화 기법과 트랜잭션 스케줄링 기법이 구현된 WiredTiger 버전을 의미한다.

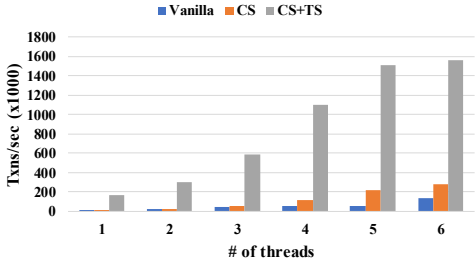

그림 6. YCSB 성능 평가 결과

[그림 6]은 YCSB의 성능 평가 결과이다. 세 가지 버전의 WiredTiger 모두 작업 쓰레드의 수가 32일 때 최고 성능을 보였으며 vanilla는 초당 14만개, 통합 동기화 기법을 구현한 CS는 초당 28.5만개, CS+TS는 초당 156만개의 트랜잭션을 수행하였다. 이는 작업 쓰레드의 수가 32개일 때 통합 동기화 기법이 2배 정도의 성능 향상을 그리고 트랜잭션 스케줄링 기법이 5.5배 정도 성능 향상을 가져옴을 보여준다.

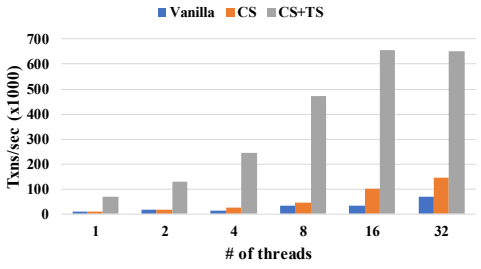

그림 7. OLTP 성능 평가 결과

[그림 7]은 Sysbench OLTP의 성능 평가 결과이다. 작업 쓰레드의 수가 16일 때 성능 향상 폭이 가장 컸다. 이 때, vanilla는 초당 3.2만개, 통합 동기화 기법을 구현한 CS는 초당 10만개, CS+TS는 초당 65만개의 트랜잭션을 수행하였다. 통합 동기화 기법이 3.1배 정도의 성능 향상을 그리고 트랜잭션 스케줄링 기법이 6.4배 정도 성능 향상을 가져왔다.

V. 결론

본 논문에서는 여러 IT 분야에서 많이 활용되고 있는 키-값 스토리지 엔진의 성능을 분석하였다. 분석된 결과에 의하면 스토리지 엔진의 트랜잭션 처리에서 트랜잭션 커밋 시 저장장치와의 동기화를 위한 fsync() 연산이 트랜잭션 처리 성능에 가장 큰 비중을 차지하고 있다. 이를 해결하기 위해 동기화 연산을 최대한 집계하여 처리하는 통합 동기화 기법과 동기화 연산이 끝나기를 기다리는 동안에 다른 트랜잭션을 수행하는 트랜잭션 스케줄링 기법을 제안하였고, 이러한 기법들을 WiredTiger 스토리지 엔진에 구현하였다. YCSB, Sysbench OLTP를 이용해서 성능을 평가한 결과를 통해 제안된 기법이 기존의 트랜잭션 처리 기법보다 더 나은 성능을 보여주는 것을 확인하였다. 향후 연구에서는 동기화 잠금을 완전히 제거하여 성능을 더욱 향상시킬 수 있는 기법을 제안하고자 한다.

참 고 문 헌

[1] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny, "Workload analysis of a large-scale key-value store," In Proceedings of the 12th ACM international conference on Measurement and Modeling of Computer Systems (SIGMETRICS '12), 2012.

[2] Swaminathan Sivasubramanian, "Amazon dynamodb: a seamlessly scalable non-relational database service," In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12), 2012.

[3] J. L. Carlson, *Redis in Action*, Manning Publications Co., 2013.

[4] Brad Fitzpatrick, *Distributed caching with memcached*, Linux Journal, 2004.

[5] Roshan Sumbaly, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, and Sam Shah, "Serving large-scale batch computed data with project Voldemort," In Proceedings of the 10th USENIX conference on File and Storage Technologies (FAST '12), 2012.

[6] Jing Han, E. Haihong, Guan Le, and Jian Du, "Survey on NoSQL database," In Proceedings of the 6th International Conference on Pervasive Computing and Applications, 2011.

[7] WiredTiger, http://www.wiredtiger.com, 2014.

[8] C. Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz, "ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging," ACM Trans. Database Syst., Vol.17, No.1, pp.94-162, 1992.

[9] P. Helland, H. Sammer, J. Lyon, R. Carr, P. Garrett, and A. Reuter, "Group commit timers and high volume transaction systems," In Proceedings of High Performance Transaction Systems (HPTS), 1987.

[10] R. Ramakrishnan and J. Gehrke, *Database management systems*, Osborne/McGraw- Hill, 2000.

[11] Ryan Johnson, Ippokratis Pandis, Radu Stoica,

Manos Athanassoulis, and Anastasia Ailamaki, “Aether: a scalable approach to logging,” In Proceedings of the 2010 international conference on very large database (VLDB '10), 2010.

[12] Jian Huang, Karsten Schwan, and Moinuddin K. Qureshi, “NVRAM-aware logging in transaction systems,” Proceedings of the 2014 international conference on very large database (VLDB '14), 2014.

[13] Shimin Chen, “FlashLogging: exploiting flash devices for synchronous logging performance,” In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (SIGMOD '09), 2009.

[14] S. Gao, J. Xu, T. Härder, B. He, B. Choi, and H. Hu, “PCMLogging: Optimizing Transaction Logging and Recovery Performance with PCM,” IEEE Transactions on Knowledge and Data Engineering, Vol.27, No.12, 2015.

[15] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears, “Benchmarking cloud serving systems with YCSB,” In Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10), 2010.

저 자 소 개

한 혁(Hyuck Han) 정회원

- 2003년 8월 : 서울대학교 컴퓨터공학부(공학사)
- 2006년 2월 : 서울대학교 컴퓨터공학부(공학석사)
- 2011년 2월 : 서울대학교 컴퓨터공학부(공학박사)
- 2011년 3월 ~ 2012년 8월 : 서울대학교 컴퓨터공학부 박사후 연구원
- 2012년 9월 ~ 2014년 2월 : 삼성전자 메모리 사업부 책임연구원
- 2014년 3월 ~ 현재 : 동덕여자대학교 컴퓨터학과 조교수

〈관심분야〉 : 데이터베이스 시스템, 병렬 프로그래밍, 분산 시스템