플래시 내 I/O 분리 처리를 통한 LSM-tree 기반 데이터베이스 성능 최적화

임희락O, 염헌영, 손용석*
서울대학교 컴퓨터공학부, *중앙대학교 소프트웨어학부
rockylim@snu.ac.kr, yeom@snu.ac.kr, *sysganda@cau.ac.kr

LSM-tree based Database System Optimization through Read/Write Isolation

Heerak LimO , Heon Young Yeom, Yongseok Son*
School of Computer Science and Engineering, Seoul National University
*School of Software, Chung-Ang University

요 약

고성능 저장장치의 역할은 웹-스케일 인프라에서 점차 더 중요해 지고 있다. 특히 NVMe(Non-Volatile Memory Express) 기반 SSD(Solid State Device)로 대표되는 차세대 저장장치는 데이터센터에 적극적으로 도입되고 있다. 하지만 데이터센터에서 실행되는 응용프로그램은 이러한 고성능 저장장치의 특징을 고려하지 않는다. 본 논문에서는 NVMe SSD와 같은 고성능 저장장치를 사용하는 LSM-tree(Log-Structured Merge tree) 기반의 데이터베이스 시스템에서, 읽기와 쓰기 요청이 혼재되어 있는 워크로드에서의 성능 저하 현상에 집중한다. 이 문제를 해결하기 위해 본 연구에서는 I/O 작업을 분리하여 처리하도록 어플리케이션 계층에서의 플래시 관리 기법을 제시한다.

1. INTRODUTION

최근 클라우드 시장의 급격한 성장에 따라 데이터센터에서 차세대 고성능 저장 장치를 적극적으로 도입하고 있다. 데이터베이스 시스템은 데이터 센터에서 안정적이고 높은 성능의 서비스를 제공해야 하는 대표적인 애플리케이션이다. 특히 RocksDB[4]와 Apache Cassandra[1]를 포함한 많은 데이터베이스 시스템 애플리케이션들은 고성능의 I/O 처리를 위해 LSM-tree(Log-Structured Merge tree)[9] 자료구조를 사용한다. 하지만 이러한 데이터베이스 시스템은 NVMe SSD와 같은 차세대 저장 장치의 특성을 반영하고 있지 않기 때문에 저장 장치의 I/O 성능을 완전하게 활용하지 못하고 있다.

NAND 기반 NVMe SSD와 같은 차세대 저장 장치는 고성능의 읽기 처리량을 갖는다. 하지만, 읽기 요청이 쓰기 요청과 섞이면 그 성능이 급격하게 떨어지는 특성을 갖는다. 그 이유는 읽기와 쓰기 요청 사이에 응답시간 차이와 GC(Garbage Collection)와 같은 FTL(Flash Translation Layer) 내부 작업으로 인한 지연 때문이다. 또한 LSM-tree 기반의 데이터베이스 시스템에서는 하위 레벨의 SST(Sorted String Table) 파일들을 합쳐 상위 레벨의 SST 파일을 만들어 내는 compaction 작업으로 인한 추가적인 I/O 요청이 발생하고, 이로 인해 저장 장치 내에서 읽기와 쓰기 작업이 더 빈번하게 섞일 수 있다. 본 논문에서는 이러한 읽기와 쓰기 요청이 혼재된 워크로드에서 고성능 저장 장치의 성능 저하 현상에 대한 실험 결과를 소개하며, 응용프로그램 기반의 플래시 관리 기법을 통해 플래시 내 I/O 분리가 성능 개선에 영향을 미칠 수 있음을 보인다.

본 논문에서는 4.의 성능평가 실험을 통해 읽기 및 쓰기가 혼재된 워크로드에서 NVMe SSD 디바이스와 데이터베이스 애플리케이션의 성능 감소를 실험을 통해 보인다. 또한 3.에서 제시한 응용프로그램 기반 플래시 관리 기법의 기대효과 및 타당성을 검증하기 위해 데이터베이스 애플리케이션의 스토리지 백엔드를 모델링 한 I/O 벤치마크 성능을 보인다.

2. RELATED WORK

읽기와 쓰기가 혼재하는 워크로드에서 데이터베이스 시스템의 성능 저하를 해결하기 위한 솔루션 중 하나로 데이터 복제를 사용한 기법이 제시되었다 [10]. 이 연구에서는 쓰기와 읽기 요청을 처리하는 물리적인 SSD 자체를 분리하여 쓰기 요청이 있더라도 읽기 요청은 독립적으로 예측 가능한 성능을 보이도록 하는 Rails 라는 디자인을 제시하였다. Rails에서는 데이터 복제를 통해서 이러한 접근 방법을 통해 성능 개선을 이루어 낼 수 있다는 것을 증명하였다. 하지만, 플래시 저장장치에서 동일한 데이터를 복제하여 서로 다른

이 성과는 2018년도 정부 (미래창조과학부)의 재원으로 한국연구재단의 차세대정보·컴퓨팅기술개발사업과 (NRF-2015M3C4A7065646, NRF-2016M3C4A7952587) 한국연구재단의 생애 첫 연구사업의(No. NRF-2018R1C1B5085640) 지원을 받아 수행됨. (교신저자: 손용석)

물리적위치에서 가지고 있어야 한다. 즉, 저장장치의 capacity를 전부 활용하지 못한다는 단점이 있다.

여러 연구에서 응용프로그램 동작 환경을 고려한 스토리지 디바이스의 데이터 관리 기법을 제시 해왔다. 멀티-스트림[7]에서는 동일한 스트림을 통해 유사한 접근 패턴을 갖는 데이터를 SSD 내부적으로 함께 배치하도록 호스트에서 스트림 정보를 주고 저장 장치는 데이터 저장에 이를 활용한다. 하지만 물리적인 데이터 배치와 GC 등은 여전히 SSD 내부 FTL에 의해서 제어되기 때문에, 응용프로그램이 요청하는 I/O 요청과 GC와 wear-leveling 같은 SSD 내부의 작업이 조화롭게 작동하지 않을 수 있다.

Open-Channel SSD는 SSD 내부의 정보 및 제어를 호스트에 공개하여 호스트 애플리케이션 또는 커널에서 직접 I/O와 플래시 디바이스를 관리하는 I/O 메커니즘이다. 데이터의 물리적인 저장 위치나, I/O 스케줄링, GC 등과 같은 기존 SSD의 컨트롤러에서 수행하던 FTL의 여러 기능이 호스트에서 수행된다. 따라서 호스트의 애플리케이션 또는 커널의 동작 환경을 활용한 Flash 최적화가 가능하다.

Open-Channel SSD를 사용한 애플리케이션 계층의 플래시 관리 기법 연구가 제시되었다 [6]. 이 연구에서 제시하는 아키텍처에서는 Open-Channel SSD를 사용하여 응용 프로그램이 직접 플래시 저장 장치를 관리한다. 타겟 데이터베이스 시스템으로 RocksDB를 사용하였으며, LSM-tree의 동작 메커니즘에 따라 I/O를 스케줄링 하였으며, 플래시 저장장치 내 물리적인 저장 위치를 결정하였다. 또한 응용프로그램 기반의 FTL을 사용하기 위해 Open-Channel SSD를 위한 유저스페이스 I/O 라이브러리인 liblightnvm을 도입하였다 [2]. 이를 통해 애플리케이션 계층에서 읽기를 수행하는 스레드와 쓰기를 수행하는 스레드가 플래시 디바이스의 서로 다른 물리적 위치에서 입출력을 수행하여 서로 간섭하지 않도록 하는 것이 가능함을 보였다.

3. APP-DRIVEN FLASH MANAGEMENT

RocksDB를 비롯한 많은 I/O 중심의 응용 프로그램은 쓰기 시 append-only 방식으로 동작하는 LSM-tree 데이터 구조를 사용한다. LSM-tree는 논리적으로 FTL과 유사한 내부 구현을 가지고 있기 때문에 응용프로그램에서 FTL을 구현할 때 큰 오버헤드 없이 구현할 수 있다. 본 논문에서는 LSM-tree 기반 데이터베이스 시스템에 최적화 된 애플리케이션 기반 플래시 관리 방법을 제안한다. RocksDB에서 들어오는 모든 데이터는 memtable에 버퍼링 된 다음 버퍼가 플러시 될 때 레벨 0의 SST(Sorted String Table) 파일로 유지된다. LSM-tree의 compaction이 실행되면 이러한 SST 파일은 다른 SST 파일과 병합되어 상위 레벨의 SST 파일을 형성하며 기존 SST 파일은 더는 사용되지 않게 된다. 이 프로세스는 FTL에서 유효하지 않은 블록을 관리하는 것과 유사하다. 따라서 이러한 오래된 SST 파일은 결국 GC의 대상이 되고 저장 장치에서 지워진다.

SSD는 내부적으로 I/O를 병렬적으로 처리할 수 있는 단위를 가지고 있다. 이러한 단위는 SSD의 Channel 마다 존재하는 Die(Chip, Way)의 수 만큼 존재한다. 예를 들어 16개 채널을 갖는 SSD에 채널 당 8개의 Die(Chip, Way)가 있으면 총 128개의 병렬 I/O 처리 단위가 있다. 이러한 병렬 처리 단위는 서로 독립적이며 서로의 작업에 영향을 미치지 않는다.

I/O 스레드가 각 SST 파일의 모든 데이터를 SSD의 병렬 처리 단위 안에 저장하면, SST 파일마다 독립적으로 병렬처리 단위에서만 I/O 처리가 가능하므로 다른 I/O 스레드의 방해 없이 SST 파일을 읽거나 쓸 수 있다. 또한 compaction을 수행하는 스레드가 SST 파일을 읽을 때, 다른 I/O 스레드의 쓰기 요청과 충돌하지 않으므로 읽기와 쓰기 사이의 대기 시간 차이로 인한 불필요한 지연이 사라진다. 즉, 읽기와 쓰기가 분리되어 불필요한 대기 시간을 줄이고 예측 가능한 성능을 달성할 수 있다. [6].

하지만 SST 파일을 하나의 병렬 처리 단위 안에서만 접근할 수 있도록 하는 디자인은 결국 병렬 처리 성능을 저하하고 전반적인 성능 저하를 가져오는 것으로 나타났다. SST 파일을 병렬 처리 단위에 정적으로 매핑하면, SST 파일의 크기 나 레벨이 고려되지 않으므로 응용 프로그램이 저장 장치의 병렬성을 충분하게 활용하지 못할 수 있다. 본 논문에서는 **4. RELATED WORK**의 기존 연구들의 문제점을 해결하기 위해 저장장치의 capacity나 병렬성의 희생 없이 플래시 내 I/O 분리를 통해 얻을 수 있는 성능 개선에 대한 실험 결과를 밝힌다. 이를 위해 Open-Channel SSD 및 liblightnvm 라이브러리를 사용하여 어플리케이션 기반의 플래시 관리 기법을 통한 성능 평가를 진행하였다.

4. PRELIMINARY EXPERIMENTS AND EVALUATION

본 연구에서는 512GiB의 Samsung 960 Pro NVMe SSD를 사용하여 성능 평가를 진행하였다. 저장 장치 자체 성능 및 데이터베이스 응용 프로그램의 성능을 평가하기 위해 fio(flexible I/O tester) 벤치마크 및 RocksDB 벤치마크 도구 (db_bench)를 사용하였다. Figure 1은 fio 벤치마크 프로그램을 사용하여 디바이스의 입출력 성능이 읽기와 쓰기 요청이 섞일 때 크게 저하되는 것을 보여 준다. 특히 읽기 요청의 비율이 50% 일 때, 읽기 성능은 읽기 요청만 수행할 때의 성능과 비교하여 약 25.6%의 성능을 보여준다. 또한 쓰기 역시 쓰기 요청만 수행할 때와 비교하여 26.8%의 성능을 보여준다. 이처럼 읽기와 쓰기가 혼합된 워크로드는 응용 프로그램에서도 저장장치 자체

성능 평가 실험 결과와 유사한 성능 저하 현상을 야기한다. Figure 2는 key-value 스토리지 시스템 애플리케이션인 RocksDB에서 워크로드의 쓰기 요청 비율이 증가함에 따라 읽기 성능이 심각하게 저하됨을 보여준다. 특히 쓰기가 워크로드의 10%를 차지하더라도 읽기 성능은 읽기 요청만 수행했을 때의 약 ¼가량으로 떨어진다.

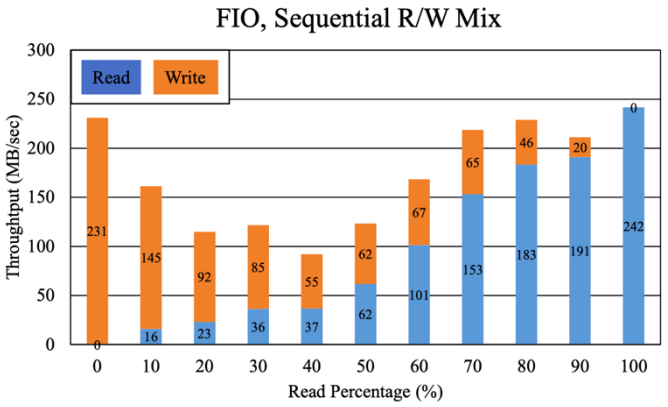

Figure 1. FIO benchmark – 워크로드에서 읽기가 차지하는 비율에 따른 읽기 및 쓰기 처리량 비교: 읽기 및 쓰기 요청이 섞일수록 성능이 비선형적으로 급격히 떨어진다.

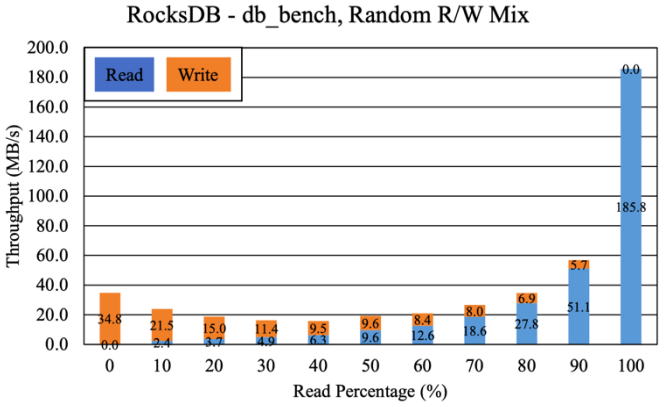

Figure 2. RocksDB's benchmark tool – 워크로드에서 읽기가 차지하는 비율에 따른 읽기 및 쓰기 처리량 비교: 읽기 및 쓰기 요청이 섞일수록 성능이 비선형적으로 급격히 떨어진다.

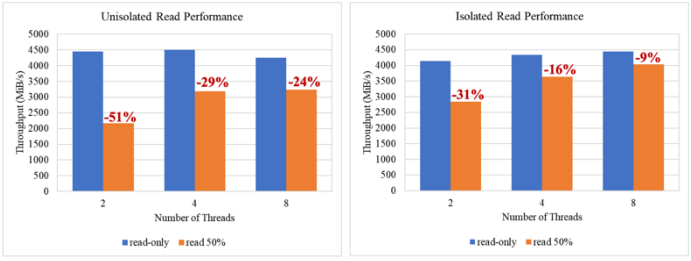

Figure 3. 읽기/쓰기 가 혼재된 워크로드에서 플래시 내 I/O 분리가 미치는 영향 비교

Figure 3 은 Open-Channel SSD를 사용하여 read-only 워크로드에서의 디바이스 성능 대비 읽기와 쓰기가 절반으로 혼재된 워크로드의 성능 감소를 나타낸 그래프이다. 읽기와 쓰기가 처리되는 플래시 내 병렬 처리 단위가 분리되지 않았을 때는, read-only 대비 최대 51%까지 성능 하락이 발생하였다. 하지만 Open-Channel SSD의 병렬처리 단위를 절반씩 읽기와 쓰기 요청을 위해 분리했을 때에는 최대 31%, 최소 9%의 성능 하락을 보였다. 즉, 읽기와 쓰기가 절반씩 혼재된 워크로드 에서 I/O가 처리되는 물리적 위치를 분리해줌으로써 최대 20%까지의 성능 향상을 보였다.

5. CONCLUSION

본 연구를 통해 플래시 내 I/O 분리 처리 및 어플리케이션 기반의 플래시 관리 기법이 읽기 및 쓰기가 혼합된 워크로드 상에서 성능개선 효과가 있음을 보였다. 차후 진행될 연구에서는 스토리지 백엔드를 모사한 벤치마크 애플리케이션이 아닌, RocksDB를 비롯한 실제 어플리케이션 환경에 이러한 I/O 분리 처리 기법을 적용하여 성능 개선 사항을 확인 해 본다.

5. REFERENCES

[1] Apache Cassandra, http://cassandra.apache.org/.

[2] liblightnvm – User space I/O library for Open-Channel SSDs, http://lightnvm.io/liblightnvm.

[3] Open-Channel Solid State Drive Interface Specification, https://openchannelssd.readthedocs.io/en/latest/specification.

[4] RocksdB, https://rocksdb.org.

[5] Feng Chen, Rubao Lee, and Xiaodong Zhang. 2011. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on. IEEE, 266-277.

[6] Javier González, Matias Bjørling, Seongno Lee, Charlie Dong, and Yiren Ronnie Huang. 2016. Application-Driven Flash Translation Layers on Open-Channel SSDs. (03 2016).

[7] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The Multi-streamed Solid-State Drive.. In HotStorage.

[8] Woon-Hak Kang, Sang-Won Lee, Bongki Moon, Yang-Suk Kee, and Moonwook Oh. 2014. Durable write cache in flash memory SSD for relational and NoSQL databases. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM, 529-540.

[9] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). Acta Informatica 33, 4 (1996), 351-385.

[10] Dimitris Skourtis, Dimitris Achlioptas, Noah Watkins, Carlos Maltzahn, and Scott A Brandt. 2014. Flash on Rails: Consistent Flash Performance through Redundancy.. In USENIX Annual Technical Conference. 463-474.