ISSN 2383-6318(Print) / ISSN 2383-6326(Online)
KIISE Transactions on Computing Practices, Vol. 26, No. 3, pp. 161-166, 2020. 3
https://doi.org/10.5626/KTCP.2020.26.3.161

GPU 가속화 필터를 적용한 디스크 기반 키-값 데이터베이스

(A Disk-based Key-Value Store with GPU-accelerated Value Filter)

김 도 영 † (Doyoung Kim)　최 원 기 † (Won Gi Choi)　노 홍 찬 †† (Hongchan Roh)　박 상 현 ††† (Sanghyun Park)

요 약 LSM-Tree 자료구조와 SST 파일로 구성된 RocksDB는 디스크 기반 키-값 데이터베이스로써 RDBMS의 스토리지 엔진으로 활용되고 있으며, 사이즈가 작은 데이터가 폭발적으로 생성되는 스마트 시티 및 IoT에서 많이 사용되고 있다. RocksDB가 스토리지 엔진으로써 사용될 때, RocksDB는 레코드를 키-값 데이터로 변환하여 저장한다. 대부분의 컬럼 데이터들은 모두 값으로 저장되므로, 스토리지 레벨에서의 필터 질의 최적화를 위해서는 값 기반 필터 연산인 value filter 연산이 필요하다. RocksDB는 키에 대한 Bloom Filter를 적용함으로써 단일 키 기반의 데이터 조회 성능은 개선하였지만, value filter 연산은 값 기반의 데이터 조회이므로 전체 키-값 데이터에 대한 조회가 발생하여 성능이 극심히 저하된다. 본 논문에서는 전체 SST 파일 조회를 GPU로 병렬처리 함으로써, value filter 연산의 성능을 최대 25% 개선하는 방법을 제시한다.

키워드: 디스크 기반 키-값 데이터베이스, GPU, Nvidia CUDA, RocksDB

Abstract RocksDB, a Disk-based Key-Value Store comprising the LSM-Tree structure and SST file, is widely used for Smart City or IoT components. When RocksDB is used as the storage engine for relational databases, RocksDB stores relational records into key-value data. Since most column data of relational record are stored in a value, a value filter operation is required for filter query optimization at the storage level. RocksDB improves the performance of key-based data retrieval by storing Bloom Filter in SST file. But the value filter operation, a value-based data retrieval operation, inevitably leads to retrieval of all key-value data resulting in poor performance. We propose a method that improves performance of all key-value data retrieval operation by using GPU accelerating. The proposed method demonstrates up to 25% more performance than the naïve RocksDB value filter operation.

Keywords: disk-based key-value store, GPU, Nvidia CUDA, RocksDB

· 이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(IITP-2017-0-00477, SW컴퓨팅산업원천기술개발사업(SW스타랩)), 2019년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원(NRF-2015M3C4A7065522)과 국토교통부의 스마트시티 혁신인재육성사업의 지원을 받아 수행된 연구임
· 이 논문은 2019 한국컴퓨터종합학술대회에서 '디스크 기반 키-값 데이터베이스의 SST 파일에 대한 GPU 가속화 필터 기법'의 제목으로 발표된 논문을 확장한 것임

† 비 회 원 : 연세대학교 컴퓨터과학과 학생
kem2182@yonsei.ac.kr
cwk1412@yonsei.ac.kr
†† 비 회 원 : SKTelecom ICT R&D Center 매니저
hongchan.roh@sk.com
††† 종신회원 : 연세대학교 컴퓨터과학과 교수(Yonsei Univ.)
sanghyun@yonsei.ac.kr
(Corresponding author임)

논문접수 : 2019년 10월 1일
(Received 1 October 2019)
논문수정 : 2020년 1월 22일
(Revised 22 January 2020)
심사완료 : 2020년 1월 27일
(Accepted 27 January 2020)

정보과학회 컴퓨팅의 실제 논문지 제26권 제3호(2020. 3)

1. 서 론

Nvidia 계열 GPU는 Compute Unified Device Architecture(CUDA)[1] API 기반 플랫폼을 통해 설계한 알고리즘을 실행할 수 있다. CUDA는 C언어를 비롯하여 다양한 프로그래밍 언어로 알고리즘을 설계할 수 있으며, GPU를 사용한 병렬처리 알고리즘이 구현된 라이브러리들을 제공하고 있다. 최근 연구로, GPU와 CUDA를 활용하여 기존 데이터베이스의 성능을 높이는 [2-5] 같은 연구들이 활발하게 진행되고 있다.

RocksDB[6]는 Facebook에서 주도적으로 개발한 디스크 기반 오픈소스 키-값 데이터베이스이며, log structured merge tree(LSM-Tree)[7] 자료구조를 기반으로 하여 SSD에 특화된 성능을 가지고 있다. RocksDB는 관계형 데이터베이스의 스토리지 엔진[8]으로써 많이 활용되고 있는데, 특히 데이터 사이즈는 작지만 데이터 양이 많이 발생하는 IoT나 스마트시티 등의 환경에서 주로 사용되고 있다. RocksDB가 관계형 데이터베이스의 스토리지 엔진으로 사용될 때, 사용자가 입력한 관계형 레코드는 키-값 데이터 형태로 변환되어 LSM-Tree로 관리되는 블록 기반 sorted static table(Block-based SST) 파일에 저장된다. 이때 관계형 레코드의 컬럼 데이터들은 값으로 저장되는데, 컬럼 데이터에 대한 질의 처리를 위해서는 값에 대한 필터 연산인 value filter 연산이 필요하다. RocksDB는 SST 파일에 키에 대한 Bloom Filter[9]를 같이 저장함으로써 키-값 데이터의 키에 대한 조회 연산 성능을 개선하였다. 하지만 값에 대한 필터 연산인 value filter 연산은 필연적으로 모든 키-값 데이터들을 순차적으로 조회하게 되며, 이로 인하여 RocksDB에 급격한 성능 저하가 발생하게 된다.

본 논문에서는 여러 Block-based SST 파일에 나누어 저장되어 있는 키-값 데이터를 해석하는 block decode 작업과 value filter 작업을 GPU에서 병렬적으로 처리하여 성능을 개선한 GPU-accelerated Value Filter 기법을 제안한다. 제안한 기법은 SST 파일의 키-값 데이터를 그대로 GPU로 복사하고, value block decode 및 value filter 작업을 GPU에서 병렬적으로 처리하기 때문에 CPU의 작업 부하가 적으며 기존 대비 최대 25%의 성능 향상을 보였다.

본 논문은 다음 섹션들로 구성된다. 섹션 2에서는 RocksDB의 Block-based SST 파일 구조와 Nvidia GPU에서 제공하는 병렬 기법들을 설명한다. 섹션 3에서는 본 논문이 제안한 방법인 GPU-accelerated Value Filter를 설명하고, 섹션 4에서는 제안한 방법과 기존 방법 간의 value filter 작업 성능을 비교한다. 마지막으로 섹션 5에서는 GPU-accelerated Value Filter의 방법에 대한 결론을 기술하고, 향후 연구에 대하여 서술한다.

2. 배 경

2.1 Block-based SST File

RocksDB는 입력된 키-값 데이터들을 디스크 상에서 LSM-Tree로 관리되고 있는 여러 Block-based SST 파일에 저장한다. 그림 1과 같이 하나의 Block-based SST 파일은 크게 5가지 종류의 block으로 구성된다. 첫번째 block 종류로, DataBlock은 인코딩 된 키-값 데이터들이 저장되는 block으로써, 키-값 데이터들은 N개의 DataBlock에 키 순서대로 저장된다. DataBlock은 내부에 일부 키-값 데이터들을 가르키는 offset 정보가 저장된 DataBlock index(seek offset)를 같이 저장한다. 두번째 block 종류로, MetaBlock은 SST 파일에 대한 메타 정보들이 저장되는 block이다. 그 중에 특히 MetaBlock 1에는 파일에 저장된 키-값에 대한 Bloom Filter가 저장되는데, 이 Bloom Filter를 통해 키-값 데이터를 키 기반으로 빠르게 조회할 수 있다. 세번째 block 종류로 MetaIndexBlock은 K개의 MetaBlock을 가르키는 index handle이 저장되어 있다. 네번째 block 종류는 IndexBlock인데, MetaIndexBlock과 비슷하게 N개의 DataBlock을 가르키는 index handle이 저장되어 있다. 이때 index handle에는 키가 같이 저장되어 있는데, 이는 DataBlock의 첫번째 키가 저장된다. 마지막 block 종류로 Footer block은 MetaIndexBlock과 IndexBlock의 handle이 저장되어 있어, SST 파일을 처음 읽을 때 사용된다.

2.2 Nvidia CUDA

CUDA는 GPU의 코어들을 논리적 구조로 활용하여

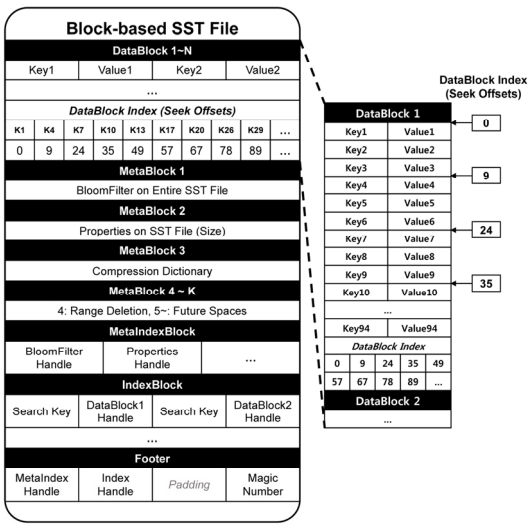

그림 1 Block-based SST 파일 구조
Fig. 1 Block-based SST File Structure

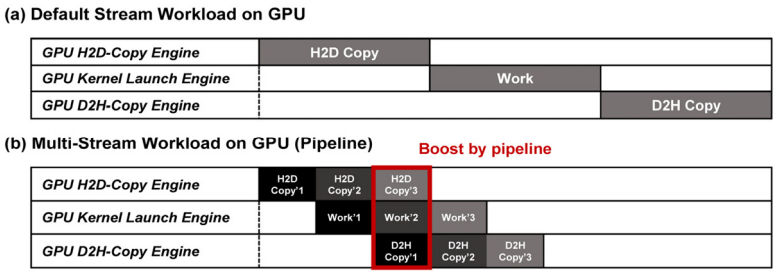

그림 2 CUDA Stream으로 구성한 pipeline
Fig. 2 Pipeline composed of CUDA Stream

사용한다. 각 GPU 코어를 thread로 표현하며, thread의 집합은 block, block들의 집합을 grid로 표현한다. GPU에 워크로드를 실행하는 방법은 3 종류의 엔진을 사용하여 진행된다. 첫번째 엔진은 Host2Device engine이라고 하며, 워크로드에 필요한 데이터들을 CPU-DRAM에서 GPU-VRAM으로 복사한다. 두번째 엔진은 Kernel engine라고 하며, 각 GPU thread에서 실행될 커널 함수를 GPU block으로 설정할 thread 개수를 설정하여 호출함으로써 GPU에서 작업을 실행한다. 마지막 엔진은 Device2Host engine이라고 하며, 작업 결과를 저장된 GPU-VRAM에서 CPU-DRAM으로 복사하여 도출해낸다. 각 엔진은 이전 엔진 작업이 모두 완료되어야 작업을 실행할 수 있으므로, 전체 작업을 한 개의 워크로드로 실행하게 되면 성능이 굉장히 저하되는 문제점이 있다.

CUDA Stream은 CUDA에서 제공하는 pipeline API로써, CUDA Stream을 활용하면 전체 작업을 다중 워크로드로 구성하여 실행하므로, 엔진 간의 병목 현상을 제거하여 성능을 높일 수 있다.

그림 2는 3개의 CUDA Stream을 사용하여 구성한 pipeline의 예시이다. 기본으로 제공하는 default Stream만 사용하여 워크로드를 구성하면, Nvidia GPU의 엔진에 할당된 작업이 종료되어야 다음 엔진의 작업이 실행될 수 있으므로, 엔진 간의 병목 현상이 발생한다. 하지만 CUDA Stream을 사용하여 워크로드를 3개의 서브 워크로드로 나누면, 엔진 간의 병목 현상을 나눌 수 있으며, 워크로드가 커질수록 pipeline으로 인한 성능 향상 효과는 증가한다.

3. 시스템 설계

3.1 동기

RocksDB는 LSM-Tree의 부족한 조회 성능을 키 기반 Bloom Filter를 적용함으로써 단일 키에 대한 조회 성능을 개선하였다. RocksDB가 스토리지 엔진으로 활용될 때, index로 지정된 column의 값은 키-값 엔트리의 키에 저장된다. index column에 대한 조건이 포함된 쿼리는 키 기반 Bloom Filter를 사용하여 뛰어난 성능을 보인다. 하지만 index가 아닌 column의 값들은 모두 키-값 엔트리의 값에 저장되는데, 이 때문에 키 기반 Bloom Filter는 index가 아닌 column에 대한 조건이 포함된 쿼리에 활용될 수 없다. 그리하여 키 기반 Bloom Filter를 사용하지 않고 값에 대한 value filter 작업을 수행하며, 이는 필연적으로 모든 SST 파일을 순차적으로 확인하게 되고, 쿼리의 성능이 크게 저하된다.

본 논문에서는 RocksDB의 value filter 작업을 GPU로 가속화하는 GPU-accelerated Value Filter를 제안한다. GPU-accelerated Value Filter는 Block-based SST 파일의 seek offset을 기준으로 GPU 병렬처리를 수행함으로써, value filter 작업의 속도를 개선한다. GPU-accelerated Value Filter의 구조는 총 3단계로, 데이터 전처리, GPU 병렬처리, 결과 데이터 수집으로 구성되어 있다.

3.2 데이터 전처리

GPU-accelerated Value Filter의 데이터 전처리 과정은 그림 3과 같이 작동한다. 먼저 GPU에서 사용하는 VRAM은 한번에 최대로 저장할 수 있는 용량이 정해져 있으므로, VRAM 용량에 적합한 양만큼 SST 파일들을 DISK로부터 읽는다. GPU 병렬처리의 기준은 DataBlock의 seek offset이므로, SST 파일들의 DataBlock들을 파싱하여 키-값 데이터와 seek offset들로 분류한다. 키-값 데이터들은 raw key-value(KV) data array에 저장하고, seek offset들은 SST 파일마다 누계하여 전체에 대한 seek offset으로 만들어 seek offset array에 저장한다. value filter 조건에 해당하는 연산자와 기준 값은 value filter context에 저장된다. 전처리되어 분류 및 생성된 raw KV data array, seek offset array는 GPU pipeline 구성을 위하여 4등분되며, 4개의 CUDA Stream에 의해 GPU로 복사된다. value filter context는 모든 CUDA Stream이 공유하는 값이므로 GPU로 한번만 복사되어 사용된다.

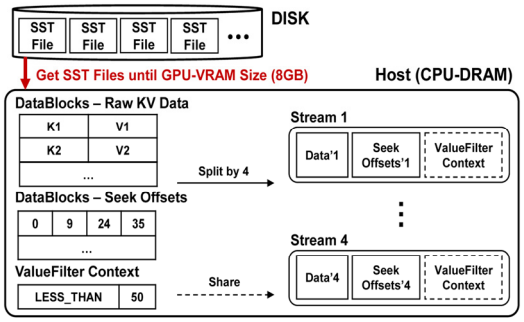

그림 3 GPU-accelerated Value Filter의 데이터 전처리
Fig. 3 Data preprocessing on GPU-accelerated Value Filter

3.3 GPU-accelerated Value Filter

value filter 작업에 필요한 데이터들이 전처리되어 모두 GPU로 복사되고 나면, GPU 커널 함수를 실행하여 value filter 작업을 수행한다. GPU 커널 함수의 동작 과정은 그림 4와 같이 작동한다. 먼저 전처리 과정에서 수집했던 seek offset마다 GPU-thread를 하나씩 할당하고, GPU-thread를 특정 개수 단위로 나누어 GPU-block을 구성하여 처리한다. 각 GPU-thread에서는 할당 받은 seek offset을 기준으로 value filter 작업을 수행하며, 수행 과정은 다음과 같다. 먼저 seek offset에 해당하는 raw KV data를 decode하여 KV pair로 변환한다. 그 다음, 변환된 KV pair를 iterate하며 value filter context를 적용하여 value filter 작업을 수행한다. 마지막으로 filter된 KV pair의 DataBlock offset 정보를 results array에 저장한다. KV pair를 바로 저장하지 않고, DataBlock offset 정보만을 저장함으로써 GPU-VRAM 용량을 효율적으로 사용할 수 있다.

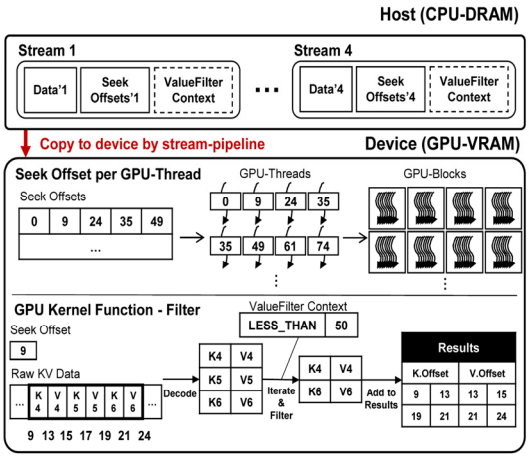

그림 4 GPU-accelerated Value Filter의 구조
Fig. 4 Structure of GPU-accelerated Value Filter

3.4 결과 데이터 수집

커널 함수의 실행이 끝나면, value filter 결과물이 저장된 results array를 GPU-VRAM으로부터 CPU-DRAM으로 복사한다. 결과 데이터를 수집하는 과정은 그림 5와 같이 작동한다. 먼저 results array에는 KV pair에 해당하는 DataBlock offset이 저장되어 있으므로, raw KV data array를 조회하여 KV pair로 변형한다. results array는 4개의 CUDA Stream을 통하여 pipeline화되어, 4개의 results array로 나누어졌기 때문에, 각 results array마다 1개의 CPU thread를 할당하여 KV pair로 변형한다. 변형하는 작업이 모두 완료되면, value filter의 작업이 종료된다.

기존의 Iterator 기반 value filter는 모든 키-값 데이터에 대하여 순차적으로 SST 파일을 조회하므로 성능이 크게 저하되지만, 제안한 방법은 GPU를 활용하여 여러 키-값 데이터를 병렬적으로 처리하기 때문에 조회 성능을 크게 개선할 수 있고, 키-값 데이터가 많이 저장되어 있는 환경에 적합하다.

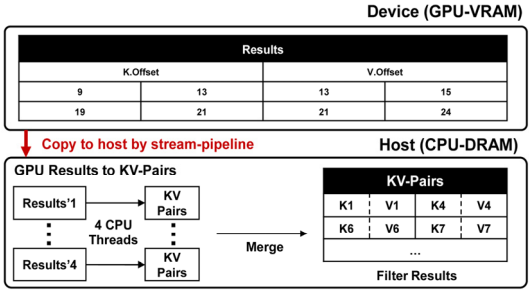

그림 5 GPU 실행 결과 데이터 수집
Fig. 5 Data collection from GPU execution

4. 실 험

실험 환경은 표 1과 같다. 이 중에 GPU는 VRAM의 크기가 8GB인 Nvidia Geforce GTX 1070ti를 사용하였는데, 이는 GPU-accelerated Value Filter에서 한번에 처리하는 최대 SST 파일 개수의 기준이 된다.

본 논문에서 비교한 실험 대상은 표 2와 같다. 표 2의 data get 방법은 RocksDB에 저장된 키-값 데이터를

표 1 실험 환경
Table 1 Test Environment

CPU	Intel i7 6700K (Quadcore, 4.0 GHz)
DRAM	64GB (2400MHz)
HDD	3TB (7200 RPM)
GPU	Nvidia Geforce GTX 1070ti, 8GB
CUDA	10.1.105

표 2 실험 대상
Table 2 Test Subjects

data get method	value filter method
Iterator	CPU (native)
	AVX
	GPU-Thrust
Block	GPU-Block (proposed method)

조회하는 방법을 의미하는데, 이중에 Iterator는 저장된 모든 키-값 데이터를 순서대로 하나씩 조회하는 방법을 의미하고, block은 SST 파일의 DataBlock 단위로 조회하는 방법을 의미한다. 표 2의 value filter 방법은 RocksDB에서 값 기반 필터 작업을 수행하는 방법을 의미하는데, 이중에 AVX는 Intel CPU의 Vector Processing Unit인 Advanced Vector Extension(AVX)[10]를 활용하여 구현한 필터 작업을 의미하고, GPU-Thrust는 Nvidia Thrust API[11]를 활용하여 구현한 필터 작업을 의미한다.

실험 set은 다음과 같이 설정하였다. 키는 0에서부터 1씩 증가하여 저장하였고, 값은 최소 0, 최대 99999중에서 임의의 값을 가지도록 저장하였다. 데이터 저장 개수는 1M개, 10M개, 100M개, 1000M개로 설정하였고, 필터 조건은 필터되는 데이터 개수가 전체의 0%, 20%, 40%, 60%, 80%, 100%로 설정하였다. 실험 결과는 CPU 방법의 성능 대비 throughput 성능 변화를 측정하였다.

실험 결과는 그림 6의 그래프와 같다. 모든 실험 set에서 CPU, AVX, GPU-Thrust의 성능이 근사하였다. 이는 Iterator로 키-값 데이터를 조회하는 비용이 value filter 비용보다 더 많은 비중을 차지하고 있기 때문이다. 한편, GPU-block의 성능은 실험 set에 따라 큰 차이를 보였는데, 그림 6의 (a)처럼 필터 조건이 40% 이하로 설정되었을 때, 데이터 저장 개수가 증가할수록 GPU-block의 성능이 다른 방법들보다 월등히 뛰어난 것을 확인할 수 있었다. 특히 필터 조건이 0%이고, 데이터 저장 개수가 1000M개인 상황에서 GPU-block의 성능이 기존 CPU 방법 대비 25%만큼 향상되었다. 하지만 그림 6의 (b)처럼 필터 조건이 60% 이상으로 설정되면, 전체적으로 GPU-block의 성능이 기존 CPU 방법보다 크게 감소되었다. 이는 섹션 3.4에서 설명했던 결과 데이터 수집 단계에서, results array가 증가함에 따라 CPU thread에서의 처리 비용이 증가했기 때문이다. 제안한 모델인 GPU-block은 데이터 저장 개수가 많을수록, 필터되는 데이터 개수가 적을수록 높은 성능을 가진다. RocksDB를 관계형 데이터베이스의 스토리지 엔진으로 활용할 때 질의 선택도(query selectivity)를 value filter에 적용하여, 질의에 따라 value filter를 적절히 선택한다면 성능을 보장할 수 있을 것이라 예상된다.

5. 결론 및 향후 연구

RocksDB는 LSM-Tree의 부족한 조회 성능을 키 기반 Bloom Filter를 적용함으로써 단일 키에 대한 조회 성능을 개선하였다. 하지만 값에 대한 데이터 조회 작업인 value filter 작업은 키 기반 Bloom Filter를 사용할 수 없으므로 모든 데이터를 순서대로 조회해야 하는 문제점이 있다. 본 논문은 모든 키-값 데이터 조회가 발생하는 value filter 작업을 GPU로 병렬처리하는 방법을 제안하였다. 그리고 실험을 통하여 데이터 개수가 많을수록 제안한 방법의 성능이 뛰어난 것을 확인하였다. 하지만 필터되는 데이터 개수가 증가할수록 결과 데이터 수집 단계에 부하가 발생하여 성능이 급격하게 저하되었다.

향후 연구로서 RocksDB를 관계형 데이터베이스의 스토리지 엔진으로 활용하고, 테이블의 질의 선택도에 따라 GPU를 활용하는 정책을 구현하고자 한다. 그리고 RocksDB의 LSM-Tree는 Level이 깊어질수록 조회해야하는 SST 파일의 개수가 기하급수적으로 증가하는데, 깊은 Level의 SST 파일에 대해서만 GPU value filter를 사용하는 정책을 구현하고자 한다. 상기한 두 정책을 활용한다면, 결과 데이터 수집 단계에서 발생하는 부하에 의한 성능 저하를 해결할 수 있을 것으로 기대한다.

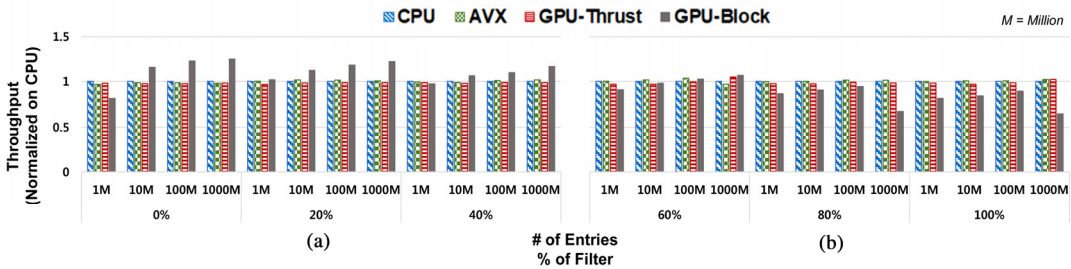

그림 6 데이터 개수, 필터 정도에 따른 throughput 성능
Fig. 6 Throughput performance based on number of entries and percent of filter

References

[1] Nvidia CUDA [Online]. Available: https://docs.nvidia.com/cuda/ (downloaded 2020. Jan. 29)

[2] M. A. Awad, et al., "Engineering a High-Performance GPU B-Tree," *Proc. of the 24th Symposium on Principles and Practice of Parallel Programming (PPoPP'19)*, pp. 145-157, Feb. 2019.

[3] S. Ashkiani, et al., "GPU LSM: A Dynamic Dictionary Data Structure for the GPU," *IEEE International Parallel and Distributed Processing Symposium (IPDPS'18)*, pp. 430-440, May 2018.

[4] P. Bakkum, et al., "Accelerating SQL Database Operations on a GPU with CUDA," *Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU-3)*, pp. 94-103, Mar 2010.

[5] K. Zhang, et al., "Mega-KV: A Case for GPUs to Maximize the Throughput of In-Memory Key-Value Stores," *Proc. of the VLDB Endowment 8.11*, pp. 1226-1237, Jul. 2015.

[6] RocksDB [Online]. Available: https://rocksdb.org/ (downloaded 2020. Jan. 29)

[7] P. O'Neil, et al., "The log-structured merge-tree (LSM-tree)," *Acta Informatica* 33.4, pp. 351-385, 1996.

[8] MyRocks [Online]. Avilable: http://myrocks.io/ (downloaded 2020. Jan. 29)

[9] B. Debnath, et al., "BloomFlash: Bloom filter on flash-based storage," *IEEE International Conference on Distributed Computing Systems*, pp. 635-644, Jun. 2011.

[10] C. Lomont, "Introduction to intel advanced vector extensions," *Intel White Paper*, pp. 1-21, 2011.

[11] Nvidia Thrust [Online]. Available: https://docs.nvidia.com/cuda/thrust/index.html (downloaded 2020. Jan. 29)

김 도 영

2018년 연세대학교 컴퓨터과학과(학사)
2020년 연세대학교 컴퓨터과학과(석사)
관심분야는 데이터베이스 시스템, 빅데이터, 분산처리 시스템

최 원 기

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

노 홍 찬

2006년 연세대학교 컴퓨터과학과(학사)
2008년 연세대학교 컴퓨터과학과(석사)
2014년 연세대학교 컴퓨터과학과(박사)
2014년~현재 SK Telecom AIX 기술센터 연구원. 관심분야는 데이터베이스 시스템, 플래시 SSD

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