연속 컴팩션 기법을 이용한 LSM-tree의 FPGA 컴팩션 가속기

임민제°, 신동군
성균관대학교 전자전기컴퓨터공학과
scvhero9607@gmail.com, dongkun@skku.edu

LSM-Tree Compaction FPGA Accelerator using Cascading Compaction

Minje Lim°, Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

Log-structured merge Tree(LSM-tree)는 Key-value store를 구현하기 위해 사용되는 주요한 자료구조 중 하나이다. LSM-tree의 구조를 유지하기 위해서는 기존의 파일들을 병합하여 중복된 Key를 가진 레코드를 제거하고, 새로운 파일을 생성하는 컴팩션 작업이 수행될 필요가 있는데, 컴팩션 작업은 DB의 스토리지 쓰기 지연을 유발하여 성능에 악영향을 미친다. 이 논문에서는 FPGA 기반 컴팩션 가속기를 설계 및 구현하였고, 제한된 decoder의 수로 Level 0 - Level 1 컴팩션을 처리하기 위한 연속 컴팩션 기법을 제안한다. Host-only version과 비교한 결과, 컴팩션 성능을 최대 79%, end-to-end 성능을 최대 28.2% 개선하였다.

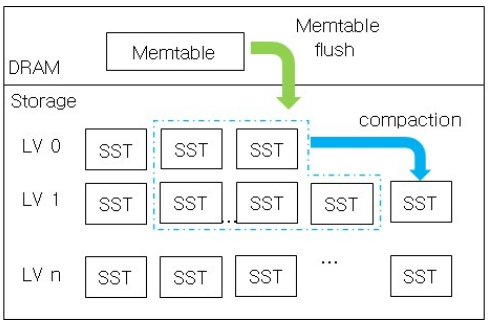
1. 서론

Log-structured Merge Tree(LSM-tree)[1] 기반의 Key-value Store는 Big Data Processing 등의 분야에서 기존의 RDBMS를 대체하여 널리 사용되고 있으며, LevelDB[2], Cassandra[3], RocksDB[4] 등에서 사용되고 있다. LSM-tree는 사용자가 삽입한 (key, value) 레코드를 In-memory 구조체인 Memtable과, Key 기준으로 정렬된 레코드가 연속되어 기록된 SSTable(Sorted String Table) 형식으로 나누어 관리한다. 그림 1은 LSM-tree의 구조를 나타낸 것으로, 사용자가 삽입한 (key, value) 레코드는 Memtable에 저장되며, Memtable의 크기가 임계값에 도달할 경우 Memtable을 SSTable로 변환한 뒤 스토리지에 저장하는 Memtable Flush 작업이 수행된다. SSTable은 여러 레벨로 분할되어 관리되며, SSTable 간 중복된 레코드를 제거하기 위하여 기존 SSTable들을 병합하여 새로운 SSTable을 만드는 컴팩션(Compaction) 작업이 수행된다.

컴팩션 작업은 사용자의 레코드 삽입 작업을 지연시킬 수 있으며, 이는 쓰기 성능 하락의 주요한 원인이 된다. 우리가 LevelDB에서 db_bench의 fillrandom workload 실행 시 쓰기 작업의 지연을 분석해본 결과, 쓰기 작업을 수행하는 도중 발생하는 지연의 84.3%가 컴팩션으로 인한 지연인 것을 확인할 수 있었다.

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2020R1F1A1073758).

본 논문에서는 컴팩션으로 인한 쓰기 지연 문제를 완화하기 위하여 다음과 같은 작업을 수행하였다.



[그림 1] LSM-tree의 구조

- FPGA를 이용하여 LSM-Tree의 컴팩션 작업을 가속할 수 있는 IP를 설계 및 구현하였다.
- Level0-Level1 컴팩션을 FPGA offloading하기 위해 필요한 연속 컴팩션 기법을 설계 및 구현하였다.
- LevelDB에서 FPGA 컴팩션 가속 IP를 사용할 수 있도록 수정하고, 컴팩션 성능을 최대 79%, end-to-end 성능을 최대 28.2% 개선하였다.

2. 배경 및 관련 연구

2.1 LSM-tree SSTable 구조 및 컴팩션

SSTable은 LSM-tree에서 스토리지에 레코드들을 파일로 저장할 때 이용되는 형식이며, 실제 레코드가 기록되어 있는 데이터 블록, 각 데이터 블록의 크기와 파일 내부 위치 정보가 기록된 인덱스 블록, 그리고 메타데이

터 정보를 담고 있는 푸터(footer)로 구성되어 있다.

Memtable flush를 통하여 DRAM에서 스토리지의 레벨 0에 SSTable이 기록될 경우, 새롭게 기록된 SSTable과 기존의 SSTable 간에 중복되는 Key를 가지는 레코드가 발생할 수 있으며, 중복된 Key range를 가진 SSTable의 수가 증가할수록 특정 Key를 찾기 위하여 여러 SSTable을 확인해야 하는 문제가 발생한다. 이러한 문제를 해결하기 위해 컴팩션 작업이 수행되어야 한다.

컴팩션 작업은 레벨 i에 존재하는 SSTable 수가 임계값에 도달할 경우 수행되며, 레벨 i와 레벨 i+1에 존재하는 중복되는 Key range를 가진 SSTable들을 병합하여 새로운 SSTable을 레벨 i + 1에 기록하게 된다.

2.2 LSM-Tree 작업 FPGA 오프로딩

Teng Zhang 등이 제안한 논문[5]은 LSM-tree 기반의 Key-value store인 PolarDB에서 DB 쿼리를 FPGA를 탑재한 SSD에 오프로딩한 논문으로, 별도의 메타데이터 처리 없이 데이터 블록을 처리할 수 있도록 데이터 블록에 별도의 헤더를 추가하였다. 반면 본 논문에서는 기존의 LevelDB 포맷을 그대로 이용하였다.

Wei Cao 등이 제안한 논문[6]과 X. Sun 등이 제안한 논문[7]은 FPGA PCIe 가속기에 LSM-tree Key-value store의 컴팩션을 오프로딩한 논문으로, FPGA에서 메타데이터를 처리할 경우 DRAM 임의 접근에 의한 비효율성으로 인하여 오프로딩으로 얻을 수 있는 이득이 적어 Host의 CPU를 이용하여 SSTable의 메타데이터를 처리하였고, Level 0-Level 1 컴팩션에 필요한 디코더의 수가 부족할 경우 컴팩션을 오프로딩 하는 대신 Host에서 컴팩션을 진행한다. 반면, 본 논문에서는 연속 컴팩션 기법을 이용하여 Level 0-Level 1 컴팩션 오프로딩이 가능하도록 하였고, 이를 위하여 메타데이터의 처리까지 FPGA를 이용하여 구현하였으며, 메타데이터의 처리와 데이터의 처리가 병렬적으로 수행될 수 있도록 하였다.

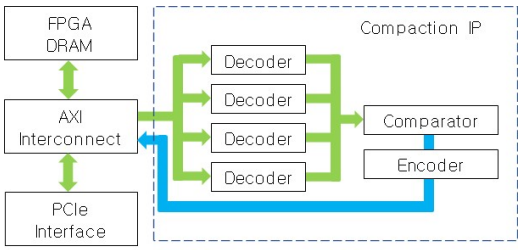
3. 설계 및 구현

이 절에서는 컴팩션 IP의 구조를 설명한다. 먼저, IP의 내부 구조에 대해 소개하며, 그 다음, Level0-Level1 컴팩션을 처리하기 위한 연속 컴팩션 기법을 설명한다.

3.1 IP 설계

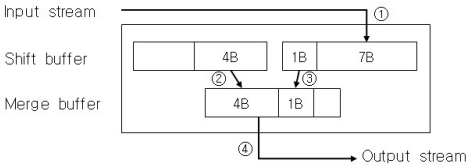
컴팩션 가속기는 컴팩션 IP, AXI interconnect, FPGA DRAM, PCIe 인터페이스로 구성되어 있다. PCIe 인터페이스는 FPGA DRAM과 Host DRAM 간 데이터의 DMA transfer 제어 및 컴팩션 IP의 제어 레지스터를 PCIe BAR(Base Address Register)에 맵핑하는 역할을 담당하며, AXI interconnect는 PCIe 인터페이스와 컴팩션 IP가 FPGA DRAM에 접근할 수 있도록 연결한다.

컴팩션 IP는 SSTable을 Key-Value 스트림 형태로 변환하는 디코더, 각 디코더에서 Key를 수신하여 비교하고 최소 key부터 인코더에 전달하는 컴퍼레이터(Comparator), 그리고 Key-Value 스트림을 SSTable로 변환하는 인코더로 나뉘어진다. 각 구성요소는 AXI4-Stream 인터페이스를 통해 연결되어 서로 비동기적으로 동작할 수 있도록 설계되었으며, Bus width는 증가할 경우 데이터 전송 속도는 증가하는 반면 레지스터의 연산 오버헤드 등이 증가하는 문제가 있으므로 트레이드 오프를 고려하여 8 Byte로 결정하였다.



[그림 2] 컴팩션 IP 구조

디코더는 푸터 및 인덱스 블록을 파싱하여 데이터 블록 정보를 추출하는 인덱스 디코더와, 데이터 블록 내부 레코드를 분할하여 스트림 형태로 컴퍼레이터에 전달하는 데이터 디코더로 구성되며, 인덱스 디코더와 데이터 디코더가 서로 비동기적으로 동작함으로써 메타데이터 처리에 걸리는 시간과 데이터 블록 블록의 처리에 걸리는 시간이 서로 오버랩 된다. 데이터 블록 내부에 연속적으로 기록되어 있는 (Key, Value) 레코드들은 임의 길이를 가진 반면, 스트림에서는 데이터가 8 Byte씩 전송되므로, 레코드를 분할하기 위해서는 데이터를 버퍼링하고 임의 길이로 분할하여 전송할 수 있는 기능이 필요하다. 스트림 세퍼레이터(separator)는 입력 스트림에서 수신한 바이트 중 일부만 출력 스트림에 전송하고, 남은 바이트는 임시로 버퍼링하여 다음 출력에 전송하는 역할을 수행하며, 읽기와 쓰기를 동시에 진행할 수 있도록 2개로 구성된 8 Byte Shift buffer와 1개의 8 Byte Merge buffer로 구성된다. 그림 3은 스트림 세퍼레이터 내부에 4B의 데이터가 남아있고, 5B의 데이터가 요청되었을 경우의 동작을 보여준다. 스트림 세퍼레이터는 ① Input stream에서 수신한 내용을 비어 있는 Shift buffer에 기록하는 동시에 ② Shift buffer에 남아있던 4B를 merge buffer에 기록하고, ③ Shift buffer의 1B를 Merge buffer에 병합한 뒤 ④ 출력 스트림에 전송하는 과정을 거쳐 동작한다.



[그림 3] 스트림 세퍼레이터의 동작

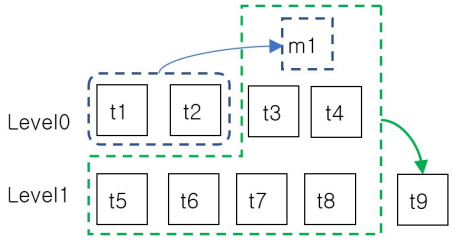
컴퍼레이터는 각 디코더로부터 수신한 Key들을 비교하여 가장 작은 Key를 선정한 다음, 해당 Key가 중복되지 않는 Key일 경우 인코더에 해당 레코드를 전달하며, 중복된 key일 경우 최신 레코드를 제외한 레코드를 폐기한다.

인코더는 컴퍼레이터로부터 수신한 레코드를 SSTable로 변환하고, 결과를 FPGA DRAM에 기록하는 역할을

수행한다. SSTable의 데이터 블록은 별도의 버퍼링 없이 DRAM에 기록되며, 인덱스 블록과 푸터는 BRAM 버퍼에 임시 작성된 뒤, SSTable의 크기가 임계값을 넘어 파일을 분리해야 할 경우 현재까지 기록된 데이터를 기반으로 DRAM에 기록된다.

3.2 연속 컴팩션

Level 1 이상에 존재하는 SSTable들의 경우에는 서로 Key range가 겹치지 않으므로 같은 레벨의 SSTable들을 하나의 디코더가 순차적으로 처리하여도 문제가 없지만, Level 0의 SSTable들은 서로 Key range가 겹칠 수 있기 때문에 각 SSTable들을 동시에 비교해야 올바른 최소 Key를 얻을 수 있고, 이를 하드웨어적으로 구현하기 위해서는 Level 0의 SSTable의 수만큼 디코더가 필요하다는 문제가 있다. 연속 컴팩션은 Level 0-Level1 컴팩션 수행 시 디코더의 개수가 n개일 경우, Level 0의 SSTable이 n-1개 이하가 될 때까지 Level 0의 일부 SSTable에 대해서만 컴팩션을 수행하여 임시 SSTable을 작성한 뒤. Level 0의 임시 SSTable에 n-1개, Level 1의 SSTable들에 1개의 디코더를 할당하여 컴팩션을 수행하는 기법이다. 이 때, 임시 SSTable의 크기가 커질 경우 다음 컴팩션의 수행 시간이 증가하므로, 임시 SSTable 생성 과정에서는 디코더의 수를 확보하는데 필요한 최소한의 SSTable만이 컴팩션의 대상이 된다. 그림 4는 4개의 디코더를 이용하여 Level 0의 SSTable t1~t4와 Level 1의 t5~t8에 연속 컴팩션을 수행하는 예시를 보여준다. 먼저, 2개의 디코더를 이용하여 t1과 t2에 컴팩션을 수행하여 임시 SSTable m1을 생성한 뒤, m1, t3, t4에 각각 디코더 1개씩를 할당하고, Level 1의 t5~t8는 정렬되어 있으므로 디코더 1개를 통해 4개의 SSTable을 순차적으로 읽어들이며 컴팩션을 수행하여 최종 결과 t9를 생성한다



[그림 4] 연속 컴팩션

3.3 LevelDB 수정

LevelDB에서 컴팩션을 수행하는 백그라운드 스레드에서 기존 컴팩션 함수인 *DoCompactionWork()* 를 호출하는 대신 PCIe FPGA 가속기에 컴팩션 작업을 오프로딩 시키는 함수를 호출하도록 수정하였다. 컴팩션 스레드는 컴팩션의 대상이 되는 SSTable들을 Host DRAM 버퍼에 적재한 후 DMA transfer를 통해 FPGA DRAM에 복사한 뒤 컴팩션 IP를 동작 시키며, 컴팩션이 전부 완료되었을 경우 다시 FPGA DRAM에서 Host DRAM으로 SSTable을 복사한 뒤 파일에 기록한다.

4. 실 험

4.1 실험 환경

Host PC는 i7 4790 CPU, 12GB DRAM, 500GB SSD를 사용하였으며, Ubuntu 16.04 환경에서 실험을 진행하였다. FPGA는 Xilinx ZCU106[8]를 사용하였고, PCIE 3.0 x4 인터페이스를 통해 Host와 연결되며 250Mhz의 동작 속도를 가진다. 벤치마크에는 LevelDB db_bench의 fill-random workload를 이용하여 500만번의 레코드 삽입을 테스트 하였다.

4.2 성능 평가

Level 0-1 컴팩션의 경우, SW-only 버전의 경우 126.4MB/s, FPGA 버전의 경우 156.7MB/s로 연속 컴팩션으로 인하여 상대적으로 적은 23%의 성능 향상을 보였지만, 그 외의 경우에는 SW-only 버전의 경우 109.1MB/s, FPGA 버전의 경우 195.6MB/s로 최대 약 79%의 성능 향상을 확인할 수 있다. End-to-end performance의 경우, original LevelDB의 경우 19.1MB/s, FPGA offloading 적용 시 24.5 MB/s로, 약 28.2% 성능 개선이 있었다.

5. 결론

본 논문에서는 LSM-tree 기반의 Key-value store에서의 컴팩션 동작을 FPGA를 이용하여 가속하는 IP를 구현하였다. 그리고, 제한된 decoder를 이용하여 Level0-Level1 컴팩션을 수행할 수 있는 연속 컴팩션을 구현하였다. CPU 버전 대비 순수 컴팩션 성능을 1.79배, 전체 DB 성능을 최대 28.2% 개선하였다.

참고문헌

[1] P. O'Neil et al, "The log-structured merge-tree (LSM-tree)". Acta Informatica, 33(4):351-385, 1996.

[2] Google, LevelDB, https://github.com/google/leveldb

[3] Apache, Cassandra, cassandra.apache.org

[4] Facebook, RocksDB, https://github.com/facebook/rocksdb

[5] Teng Zhang et al, "FPGA-Accelerated Compactions for LSM-based Key-Value Store", *In: 18th USENIX Conference on File and Storage Technologies (FAST 20)*. 2020, pp. 225-237

[6] Wei Cao et al, "POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database", *In: 18th USENIX Conference on File and Storage Technologies (FAST 20)*. 2020, pp. 29-41

[7] X. Sun et al, "FPGA-based Compaction Engine for Accelerating LSM-tree Key-Value Stores", In: *2020 IEE 36th International Conference on Data Engineering (ICDE)*. 2020, pp. 1261-1272

[8] https://www.xilinx.com/products/boards-and-kits/zcu106.html