LSM-tree기반 데이터베이스에서 발생하는 Stall 현상 및 원인 분석

김재은, 강수용
한양대학교 컴퓨터소프트웨어학과
wrlawodms@gmail.com, sykang@hanyang.ac.kr

The analysis for stalls of LSM-tree on LevelDB

Jaeeun Kim, Sooyong Kang
Department of Computer Science, Hanyang University

요 약

LSM-tree란 디스크 기반의 색인을 위한 자료구조 중 하나로 데이터베이스 시스템에서 고성능의 트랜잭션(transaction)처리를 제공하기 위한 실시간 색인구조이다. 데이터를 주기억장치와 보조기억장치에 계층적(multi-level)으로 구성하고 데이터베이스의 변화를 부분적으로 배치(batch)-병합(merge) 함으로써 쓰기의 성능을 향상시킨다. 하지만 이러한 구조를 실제로 구현하게 되면 일정 수준 이상의 쓰기요청이 계속 들어올 경우 병합과정으로 인해 순간적으로 처리량이 급격히 떨어지는 구간이 존재하는데, 이를 스톨(stall)이라고 한다. 본 논문에서는 구현체인 levelDB를 통해 실제 스톨이 일어나는 현상과 원인을 분석하여 제시한다.

1. 서 론

데이터베이스(Database System)에서는 데이터를 일관성(consistency)있고 지속성(durability)있게 유지해야 하며, 이러한 데이터에 빠르게 접근할 수 있는 것이 중요하다.

색인(index)이란 데이터베이스 내의 레코드(record)를 특정한 구조적 형태로 저장하여 빠른 속도로 임의접근할 수 있게 하는 기법을 말한다. 특히 OLTP(Online transaction processing)의 경우 이러한 색인의 구조를 실시간으로 변형, 유지해야 하며 이를 위해 B+tree[1]나 hash indexing[2]기법이 주로 사용되고 있다.

LSM-tree(Log structured merge-tree)[3]란 색인을 위한 자료구조 중의 하나로 많은 수의 쓰기 요청이 있을 때 낮은 비용으로 실시간 색인을 유지할 수 있도록 하기 위해 등장하였다. LSM-tree는 [그림 1]과 같이 크게 메모리부분과 디스크부분으로 나뉘는데 최신 입력 데이터는 먼저 메모리부분에 정렬된 상태로 유지되게 된다. 이때, 전체 색인구조와는 무관하게 고성능의 메모리내에서 비교적 소량의 데이터만으로 정렬된 상태를 유지하므로 빠른 속도로 쓰기 요청을 처리 할 수 있다. 이렇게 부분적으로 정렬되어 유지되던 데이터는 메모리버퍼를 모두 사용하게 될 때까지 디스크내의 데이터와의 병합(merge)이 미루어지다가 배치(batch)처리된다. 디스크의 데이터와 병합될 때에도 일부의 데이터와 병합을 진행하여 빠른 속도로 디스크 구조를 변형하고 유지할 수 있도록

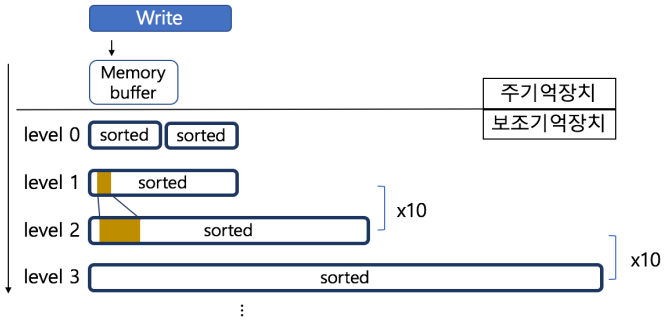

그림 1 LevelDB의 LSM-tree 구조도

하는데, 이를 위해 디스크 내의 데이터를 멀티레벨(multi-level)로 구성하여 가장 낮은 레벨의 데이터와의 병합만을 진행한다. 마찬가지로 낮은 레벨에 데이터가 가득 차게 되면 한 단계 더 높은 레벨과의 병합이 일어난다. 이러한 전파병합(cascading merge)방식을 통해 최소한의 구조변화로 색인구조를 유지하여 쓰기의 성능을 향상 시킨다. 특히 level 0의 경우에는 기존의 디스크데이터와의 병합을 진행하지 않고 메모리에 있는 데이터를 바로 디스크에 적을 수 있고, level0보다 상위 레벨들의 크기를 10배씩 증가시켜나감으로서 트랜잭션의 가용성을 높힌 구조이다. 이러한 구조는 일정시간동안 굉장히 높은 쓰기성능을 보여준다. 하지만 뒤로 미루어진 색인구조 변화를 진행하는 동안에도 쓰기요청이 계속해서 빠른속도로 일어날 경우에는 더 이상 요청을 처리 할 수 없어 잠시동안 처리가 지연되고 이를 **스톨(stall)**이라 한다.

표 1 실험 머신 하드웨어 환경

Component	Specification
Processor	Intel Xeon E5-2630 v4
Memory	16GB
Disk	Samsung 850 Evo 500G

스톨이 일어나는 경우는 메모리버퍼의 데이터를 level0로 내리는 비교적 적은 비용이 요구되는 것부터 다른 상위레벨의 공간도 가득 차서 전체적인 구조를 바꿔야 하는 높은 비용이 요구되는 경우까지 다양하다. 본 논문에서는 실제 구현체인 LevelDB[4]라는 open source key-value 데이터베이스를 통하여 스톨이 일어나는 현상과 그 원인들에 대해 분석하고 개선방향을 제시한다. LevelDB에서는 합병을 위한 스레드(thread)와 사용자의 요청을 위한 스레드가 동시에 실행되는데, 사용자의 요청을 처리하던 스레드가 특정 기준을 넘어서면 백그라운드(background) 스레드에게 데이터의 합병을 요청하여 색인의 구조를 바꾸게 된다.

2. 스톨에 대한 실험 및 분석

본 장에서는 스톨에 대한 현상 및 원인을 분석하여 제시한다.

2.1. 실험 및 분석 환경

스톨에 대한 분석을 진행하기 위해 levelDB와 levelDB의 벤치마킹 툴의 소스를 일부 변경하였다. 결과적으로 크게 두가지 측정기준이 추가 되었는데, 기존 벤치마킹툴이 제공해주는 정보에 추가적으로 각 시간마다 처리한 쓰기요청량을 카운트하여 각 시간별로 처리량을 변화를 측정하였다. 또한, 각 쓰기요청들의 처리시간(latency)을 측정하여 스톨이 일어난 시간들을 기준으로 요청들을 분류하였다. 추가된 측정기준과 기존의 벤치마킹툴이 제공하는 정보들, 소스코드를 통하여 분석을 진행하였다. 스톨이 최대로 일어나는 경우를 실험하기 위해 계속해서 쓰기요청이 들어오는 경우를 가정하여 random-write workload(100% write)로 실험을 진행하였다. 키와 값의 크기는 각각 16bytes와 100bytes이며 1억개의 연산을 실행하였다. 본 실험이 수행된 자세한 컴퓨터사양은 [표 1]에 명시되어 있다. 실험이 수행된 OS는 커널버전 4.8.7의 Ubuntu 16.04이며, ext4 file system을 사용하였다.

2.2. 스톨의 원인 및 실험 분석

먼저 스톨의 원인을 분석, 정리하고 실험결과를 바탕으로 실제 이러한 스톨들이 어떠한 형태로 발생하는지를 살펴본다.

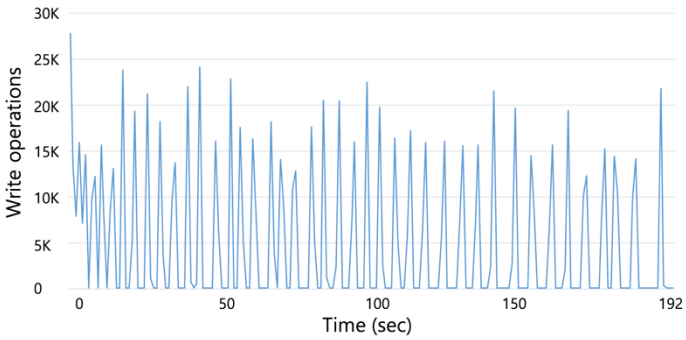

그림 2 실행 시간에 따른 임의쓰기연산들에 대한 스톨의 발생

Latency	Number of write op
~ 100ns	0
100ns ~ 1us	0
1us ~ 10us	96,906,345
10us ~ 100us	274,656
100us ~ 1ms	3,318
1ms ~ 10ms	2,809,056
10ms ~ 100ms	6,625
100ms ~ .	0
Total	100,000,000

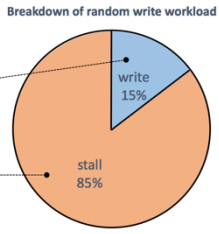

그림 3 쓰기처리시간(latency) 프로파일링

2.2.1. 스톨의 원인

LevelDB 에서 일어나는 스톨의 원인은 메모리버퍼가 디스크에 써지는 Level0 으로의 쓰기를 기준으로 하며 이를 4 가지로 정리하면 다음과 같다.

1) *Memory buffer 가 가득찬 경우*
   LevelDB 에서는 사용자의 쓰기요청이 저장되는 메모리버퍼가 가득차면 Level0 로 쓰여지기 위한 메모리버퍼로 전환된다. 이렇게 총 두개의 메모리버퍼가 존재하는데 Level0 에 공간이 없거나 처리가 너무 빨라 두 메모리버퍼가 모두 가득차면 전환된 테이블이 Level0 으로 쓰여질 때까지 스톨이 발생한다.
2) *Level0 의 크기가 일정 수준을 넘어서 상위레벨로의 합병을 기다리는 경우 (hard limit)*
   Level0 의 크기가 너무 커진 경우에는 메모리의 버퍼를 더 이상 level0 로 쓸 수 없으므로 처리를 멈추고 Level0 의 일부와 Level1 의 일부가 합병되어 Level0 의 공간이 만들어지기를 기다려야 한다. 합병이 완료되어 공간이 만들어지기까지 스톨이 발생한다.
3) *Level0 의 크기가 일정 수준을 넘어서 처리량이 줄어든 경우 (soft limit)*
   하나의 쓰기가 1)의 hard limit 으로 멈추어 있으면 쓰기요청의 처리시간의 변화량이 너무 급격해진다. 이를 여러 요청이 분산하여 latency 의 변화량을 줄이기 위해 Level0 의 크기가 일정 수준을

넘어서면 잠시동안(1ms) 멈추어 백그라운드 스레드가 합병을 할 수 있는 시간을 만들어 준다.

4) *Level0 의 크기가 일정 수준을 넘어서 기다리던 합병이 상위레벨들로 전파되는 경우*

Level0 에 공간이 부족해져서 Level1 과의 합병을 진행하는 과정에서 level1 의 공간이 부족해 질 수 있다. 이 경우 level1 과 level2 의 합병이 일어나야 한다. 이러한 과정은 저장공간의 여유가 있는 level N 까지 전파되어 일어나며 이런 경우 상대적으로 긴 스톨이 발생하게 된다.

2.2.2. 실행 시간에 따른 쓰기연산들의 스톨현상 분석

[그림 2]의 그래프를 보면 전체 실행시간 사이에 처리량이 급격하게 감소되는 부분들을 볼 수 있다. 초기에는 메모리 영역에 쓰기연산을 진행하므로 빠른 속도로 연산을 처리해 나간다. 하지만 일정 수준의 쓰기를 진행한 후부터 급격하게 처리량이 감소하게 되는 것을 볼 수 있다. 이러한 부분들이 stall 이 일어나는 시점이다. Stall 이 일어나고 일정시간이 지나면 다시 급격하게 처리량이 증가하는데 이 시점이 메모리버퍼가 비워지고 다시 빠른 속도로 쓰기처리가 가능해진 시점이다. 그래프의 초기 스톨지점을 보면 처리가 멈추는 것이 아닌 처리량이 감소되는 것이 잘 드러나 있는데 이 부분이 2.2.1.1)에서 병합이 상위레벨로 전파되지 않는 경우와 2.2.1.3) 부분이다. 이 후 처리 중간부분에서 지속적으로 처리가 멈추는 부분들이 2.2.1.2)와 2.2.1.4)의 스톨이 발생하는 부분이며, 스톨에서 복구되기까지가 시간이 지날 수록 늘어나는 것을 볼 수 있다. 스톨시간이 점점 늘어나는 것은 데이터의 양이 증가함에 따라 여러 레벨로 데이터가 퍼져나가고 합병이 상위레벨로 전파되어 합병의 시간이 오래 걸리는 경우이다.

2.2.3. 쓰기처리 처리시간 상세 분석(breakdown)

[그림 3]을 보면 각 쓰기연산들의 처리시간(latency)을 볼 수 있는데, 1ms 이하의 시간에 처리된 연산을 일반적인 연산이라 보고 1ms 를 초과한 연산을 stall 이 일어난 연산이라고 분류하였다. 엄밀히 분류하자면 10μs 초과, 1ms 이하의 연산 중 일부는 stall 로 분류할 수 있는 연산들이 있으나, 결과에 끼치는 영향은 미미하다.

일반적인 쓰기연산(약 96.9%)들은 1μs~10μs 사이에서 처리가 완료된다. 처리시간이 1ms(일반처리시간의 1000 배)를 넘는 연산들이 약 3% 존재하는데, 이 연산들이 앞서 언급한 hard limit 혹은 soft limit 에 의해 스톨이 발생한 연산들이다. 특히 10ms 가 넘어가는 연산은 병합과정에서 상위레벨들로 전파가 일어나 긴 시간동안 스톨이 발생한 연산들이다.

결과적으로 총 1 억개의 쓰기연산 중 약 3%의 연산만이 stall 로 인해 처리가 지연되었지만 처리시간으로는 약 85%를 차지하는 것을 확인할 수 있다.

3. 결론 및 개선방향

LSM-tree 를 기반으로 하는 데이터베이스 시스템은 일정시간동안 쓰기연산을 빠른 속도로 처리할 수 있으나 이에 대한 트레이드오프(trade-off)로 계속해서 들어오는 쓰기요청에 대해 일시적으로 스톨이 발생하게 된다. 스톨지점에서 소수의 요청들(3%)이 불공정하게 늦게 처리 되며, 전체적인 성능에 지대한 영향(85%)을 끼치는 것을 확인할 수 있다. 이러한 병목지점은 시스템 성능향상에 있어서 중요한 지점이며 본 논문의 구체적인 분석을 바탕으로 여러방면으로 개선이 가능할 것이다.

크게 소프트웨어적인 방법과 하드웨어적인 방법 두가지로 개선이 이루어 질 수 있다. 소프트웨어적인 개선에는 구조적인 변경이나 기술적인 최적화가 가능한데, 대표적으로 levelDB 에서 파생된 RocksDB[5]에서는 병렬성을 극대화 시켜 이를 개선하고 있으며, 하드웨어적으로는 NDP(Near Data Processing)[6]등을 이용하여 개선이 가능할 것이다.

4. 사사의 글

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음. [2014-0-00670, ICT 장비용 SW 플랫폼 구축]

5. 참고문헌

[1] Bayer, Rudolf, and Edward McCreight. "Organization and maintenance of large ordered indexes." *Software pioneers*. Springer Berlin Heidelberg, 2002. 245-262.

[2] Yuan, Pei-Sen, and De-Chang Pi. "Design and Implementation of Hash Index Used in Main Memory Database." *Jisuanji Gongcheng/ Computer Engineering* 33.18 (2007): 69-71.

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

[4] Google Inc., LevelDB, GitHub repository, https://github.com/google/leveldb

[5] Facebook Inc., RocskDB, GitHub repository, https://github.com/google/leveldb

[6] Gu, B., Yoon, A. S., Bae, D. H., Jo, I., Lee, J., Yoon, J., … Chang, D. (2016). Biscuit: A Framework for Near-Data Processing of Big Data Workloads. *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, 153-165. https://doi.org/10.1109/ISCA.2016.23