공학석사학위논문

멀티코어 CPU와 고성능 저장장치를 위한 데이터베이스 최적화

Database Optimization for Multi-core CPU and high performance storage

2018년 2월

서울대학교 대학원

컴퓨터공학부

민철기

멀티코어 CPU와 고성능 저장장치를 위한 데이터베이스 최적화

Database Optimization for Multi-core CPU and high performance storage

지도교수 염헌영

이 논문을 공학석사학위논문으로 제출함

2017년 11월

서울대학교 대학원

컴퓨터 공학부

민철기

민철기의 석사학위논문을 인준함

2018년 1월

위 원 장 **엄현상** (인)

부위원장 **염헌영** (인)

위 원 **전병곤** (인)

초록

Multi-core CPU와 고성능 저장장치가 발전하는 지금 시대에 전통적인 RDBMS(Relational database management system) 중 하나인 mysql에서는 발전하는 장비들을 제대로 활용하지 못 하고 있다. 먼저 Multi-core CPU의 활용에 대하여 기존의 mysql에서는 client의 수만큼 thread를 생성하며 하나의 client에 대해서는 기본적으로 하나의 thread 밖에 사용할 수 없다. 고성능 저장장치에 대해서는 장치의 최대 bandwidth를 모두 사용하지 못하고 있고 병렬적인 요청처리능력 또한 사용하지 못하고 있다. 네트워크 또한 CPU bottleneck에 의해 최대 bandwidth를 내지 못하고 있다.

저장장치나 네트워크의 bandwidth 활용 문제들은 CPU bottleneck에 의해서 발생하며 이것은 근본적으로 mysql의 single thread 프로그래밍에 의한 core의 활용 부족을 원인으로 볼 수 있다. 본 논문에서는 기존 mysql 코드를 수정하여 multi thread 프로그래밍을 통해 disk 읽기, record처리, 네트워크 처리 부분에서의 multi-core의 활용성을 높이고 저장장치에서 데이터를 더 효율적으로 읽어오며 네트워크 처리 또한 병렬적으로 수행 가능하도록 개선한다.

주요어 : 데이터베이스, multi-core, storage

학번 : 2016-21200

목차

그림 목차

제 1 장 서론

오늘날 multi-core CPU 와 NAND 플래시(SSD)가 발전하고 있다. 먼저 CPU 는 지금까지 CPU frequency 를 높여서 연산 능력을 발전시켜왔지만 더이상 frequency 가 높아지기 어려워지면서 frequency 보다는 코어의 개수가 많아지는 multi-core 의 방향으로 발전하게 되었다. CPU 의 코어 개수가 많아지면서 한 CPU 가 여러 thread 를 동시에 수행할 수 있게 되었고 그에 따라 멀티 thread 프로그래밍을 통해 늘어난 코어를 잘 활용하는 것이 중요해졌다.

NAND 플래시가 사용되는 SSD 는 기존에 주된 저장장치로 사용되던 하드 disk 를 대체하고 있다. SSD 는 하드 disk 에 비해 비교적 적은 용량을 가지지만 빠른 읽기와 빠른 쓰기 속도를 가지며 하드 disk 와 달리 random, sequential 접근에 따른 영향을 적게 받는다.

위 두 가지 요소에 대하여 기존의 전통적인 RDBMS 인 mysql 을 대상으로 최적화를 수행하려고 한다. mysql 은 client 마다 하나의 thread 를 배정하고 client 가 보낸 요청에 대해 배정된 하나의 thread 를 통해 처리한다. 다시 말해 한 client 가 보내는 하나의 요청에 대해 기본적으로 하나의 thread 만을 이용해서 처리한다. 또한 기존의 mysql 에서 SSD 를 사용하며 OLAP workload 인 TPC-H 를 수행하거나 간단한 select query 를 수행할 경우, SSD 가 가지고 있는 빠른 읽기 bandwidth 와 병렬적인 요청처리능력을 충분히 활용하지 못 한다[7,8].

이 두 가지 요소를 중심으로 mysql 에서 select query 를 수행할 때 multi-coreCPU 와 고성능 저장장치를 효과적으로 사용할 수 있도록 기존의 코드를 분석하고 최적화된 코드를 설계, 구현한다.

제 2 장 배경

2.1 Multi-core CPU

보통 CPU 는 frequency 가 높을수록 같은 시간 동안 더 많은 양의 일을 처리한다. CPU 의 frequency 는 기술이 발전하면서 지금까지 빠르게 증가해왔지만 전력과 열의 관계에 의해 더 이상의 frequency 증가는 어려워 보이는 추세이다. 하지만 그 대신에 CPU 의 core 수를 늘리는 방향으로 발전하고 있다. 현재 개인용 PC 에서도 보통 CPU core 수가 2 개 또는 4 개 정도 되며 여러 CPU 기술을 통해서 실제 core 보다 더 많은 core 의 효과를 내기도 한다. 이러한 CPU 의 발전 방향에 따라 core 를 얼마나 잘 활용하느냐에 따라 프로그램의 성능이 영향을 받으며 이것은 다시 thread 를 어떻게 구성하고 어떻게 동기화시켜 수행할 것인지가 중요하게 되었다. 오늘날 많은 프로그램들이 멀티 코어를 지원하는지 안 하는지에 따라 CPU 코어를 활용할 수 있는지 없는지가 결정되며 그것은 장비의 사용률과 프로그램의 성능에 영향을 끼친다.

2.2 NAND 플래시

예전에는 주로 하드 disk 를 저장장치로 사용하였지만 플래시 메모리 기술이 발전하면서 NAND 플래시를 사용하는 SSD 가 하드 disk 를 대체하고 있다. 기존의 하드 disk 는 기계식으로 disk 를 회전시키고 적절한 위치로 arm 을 이동시켜 데이터를 읽고 썼지만 플래시 메모리는 반도체를 이용하여 전기적으로 데이터를 읽고 쓴다. 이에 따라 플래시 메모리는 하드 disk 에 비해 기본적으로 훨씬 빠른

읽기, 쓰기 속도를 가지며 그 외에 여러 특성들을 가진다. NAND 플래시에서는 page 단위의 읽기, 쓰기와 block 단위의 erase 를 수행한다. 또한 erase 비용이 비교적 크게 들어 erase 를 즉시 수행하지 않고 invalidation 과 garbage collection 을 하게 되었다. 또 장치의 수명이 짧아 여러 block 을 골고루 사용하기 위한 wear leveling 기술을 사용한다. request 에 있어서 block size 가 클수록 효율적이며 parallelism 이 존재해서 다수의 request 를 동시에 처리할 수 있다.

2.3 Mysql

Mysql 은 전통적인 RDBMS(Relational Database Management System)이며 오픈소스로 소스코드가 공개되어 있다. 데이터는 기본적으로 disk 같은 보조 저장장치에 저장되며 작업 요청이 있을 때 disk 에서 메모리로 올려 사용한다. 데이터의 조작은 SQL 문 형태의 Query 를 사용하여 수행한다. 현재 mysql 은 server 에 접속하는 client 수만큼 thread 가 생성되며 기본적으로 client 하나당 core 하나를 사용하게 된다.[9] Mysql 에서는 여러 종류의 storage engine 을 사용할 수 있는데 이 논문에서는 storage engine 으로 innodb 를 사용한 경우를 다룬다[1].

2.4 TPC-H

TPC-H 는 의사결정 지원 벤치마크로 데이터베이스 트랜젝션 처리가 아닌 데이터 분석을 위한 워크로드이다. TPC-H 의 데이터 부분은 customer, lineitem, nation, orders, part, partsupp, region, supplier 의 테이블로 구성되어 있으며 query 부분은 22 개의

query 문으로 구성되어 있다. 여기서 각각의 query 문은 모두 SELECT 문으로 이루어져 있어 데이터베이스에서는 쓰기는 일어나지 않고 읽기만 일어나게 된다[3,5].

제 3 장 데이터베이스 처리 과정

Mysql 의 작업은 보통 client 의 요청을 server 가 처리하는 것으로 볼 수 있다. 대부분의 과정에서는 네트워크 연결을 제외하면 client 가 요청을 보내는 것 다시 말해 SQL query 문을 보내는 것으로 시작된다. client 가 SQL query 문을 보내면 그것이 server 에 도착하게 되고 server 는 받은 SQL query 문을 parse 한다. 이 과정에서 SQL query 문이 어떤 종류의 query 인지 확인하고 분류하게 된다. 이 논문에서는 select 문에 대해 살펴본다.

3.1 Mysql Select

select 문의 경우, join 과 연관이 있는데 select 가 여러 table 에 걸쳐 일어날 수 있고 여기에 join 이 필요하기 때문이다. 일반적인 경우 nested loop join 을 사용하게 되는데 이것을 통하여 table 별로 중첩된 join 이 수행된다. 이것은 다중 for 문과 같이 table 에서 record 하나를 정하고 다음 table 로 넘어가는 것을 중첩하여 반복한다. 여기서 코드를 크게 두 부분으로 나눌 수 있다. 하나는 table 에서 결과값의 대상이 될 record 를 정하는(읽는) 것이며 이것은 read record 함수를 통해 수행된다. 다른 하나는 nested loop 를 진행하기 위해 다음 table 로 진행하는 것이며 이것은 evaluate join 함수를 통하여 수행된다. evaluate join 마지막에는 end_select 를 통하여 결국 client 에 결과 값(record)을 보내게 된다.

3.2 Read Record

Read record 함수는 table 에서 record 하나를 읽는 작업이다. 이 논문에서는 innodb storage engine 을 사용하는 경우에 대해 설명한다. Table 은 보통 disk 에 파일 형태로 존재한다. 여기서 record 를 읽는다는 것은 파일에 접근하여 record 가 존재하는 부분을 읽어 memory 에 올리는 것을 말한다. table 내부의 구조는 b-tree 로 관리되며 mysql 은 read record 작업을 하기 위하여 b-tree 내부를 탐색한다. 각각의 데이터는 b-tree 의 leaf node 에서 읽을 수 있으며 mysql 은 b-tree 를 여러 가지 방법으로 탐색할 수 있다. 이 논문에서는 full_scan 에 사용되는 rr_sequential 방법에 대해서 중점적으로 다루었다. 데이터는 b-tree 의 leaf node 에서 데이터를 page 단위로 읽을 수 있고 page 는 여러 record 로 구성되어 있다. b-tree 에 접근하여 disk 에서 데이터를 읽으면 읽힌 데이터는 memory 에 존재하는 mysql buffer 에 저장되게 된다. 데이터는 buffer 에 존재하면서 mysql 에 의해 관리되며 evict 되기 전까지 다음에 같은 데이터에 대한 요청이 오면 disk 에서 다시 읽을 없이 메모리에서 바로 읽을 수 있게 된다. innodb stroage engine 에서는 disk 와 buffer 에 데이터를 innodb format 으로 저장하고 실제 사용할 수 있는 데이터는 mysql format 이기 때문에 buffer 에 올려진 데이터를 사용하려면 추가적인 변환과정이 필요하다. 이러한 format 간의 변환 과정 또한 read record 함수 내에서 수행한다.

```
sub_select // 재귀 호출을 통해 table의 개수만큼 호출
{
        ...
        while ( table의 record 개수만큼 반복 )
        {
                read_record
                ...
                evaluate_join → sub_select 호출
        }
        ...
}
```

그림 3.1 nested loop의 구현

3.3 Evaluate Join

Evaluate join 함수는 nested loop join 에서의 하나의 loop 에서 다음 loop 로 넘어가는 함수로 table 에서 하나의 record 가 read record 함수를 통하여 읽힌 뒤 다음 table 에서의 record 를 처리하기 위하여 수행한다. 보통의 구조는 sub_select 함수 안에 while 문이 있고 while 문 안에 read record 와 evaluate join 이 있어서 이 evaluate join 이 다시 sub_select 를 호출하는 방식으로 nested loop 를 처리하게 된다. 여기서 마지막에 table 처리 중에 불리는 evaluate join 은 end_select 를 호출하게 되는데 end_select 는 end_send 로 이어져 지금까지 nested loop 를 돌면서 얻은 결과를 client 에게 네트워크를 통하여 보내는 일을 한다.

3.4 End Send

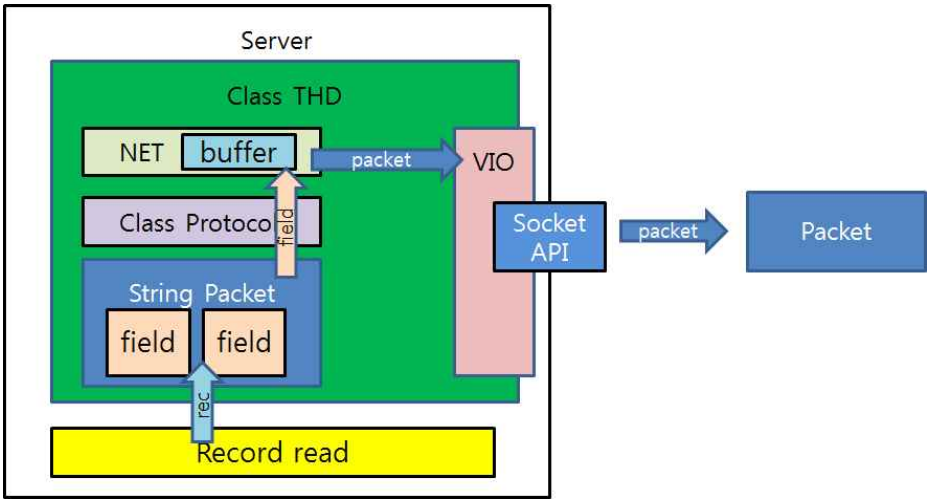

그림 3.2 네트워크 처리 흐름

end select 함수는 결국 end send 함수를 호출하여 네트워크 처리를 수행한다. end send 에서는 이전의 evaluate join 에서 nested loop 를 거쳐 지나온 데이터들을 가지고 client 가 요청한 query 에 맞는 결과 데이터들을 보내준다. Client 에게 query 에 대한 결과값을 보내기 위해 우선 보낼 query 의 형식에 대한 metadata 정보를 client 에게 보낸다. 그 후 Protocol class 를 통하여 모은 데이터를 field(데이터베이스 table 의 column 의 속성) 별로 형식에 맞게 packet 형태로 변환한다. 이 과정은 record 단위로 처리되며 처리된 데이터는 한 record 당 packet header 4byte 와 가변적인 packet data 를 구성하게 된다. packt 형태로 변환된 mysql 데이터는 NET 구조체의 packet 이라는 이름의 buffer 에 쓰여지는데 NET 구조체에서는 하나의 record 정보가 담긴 packet 들을 길이가

MAX_PACKET_SIZE 가 될 때까지 서로 이어서 저장한다. NET buffer 에 record 가 모여 데이터의 길이가 MAX_PACKET_SIZE 가 넘으려 하면 socket 입출력을 담당하는 VIO 구조체에서 NET 구조체에게 buffer 에 저장된 packet 데이터를 받아 TCP socket 을 통해 네트워크로 보낸다. 보내진 packet 은 client 가 받게 된다.

제 4 장 설계 및 구현

4.1 disk 읽기 개선

기존의 mysql 에서는 table 전체를 읽는 full scan 을 수행할 때 disk 에서 page 를 읽기 위해 b-tree 를 이용하여 b-tree 의 가장 왼쪽 leaf 까지 탐색한 다음 leaf 끼리 연결된 list 를 따라가며 sequential 한 page 들을 읽는다. 이 과정에서 mysql 은 table 별로 nested loop 를 사용하기 때문에 각각의 table 별로 b-tree 에서 어디까지 읽었는지 저장할 필요가 있고 읽은 node 와 page 를 표시하기 위해 cursor 를 사용한다.

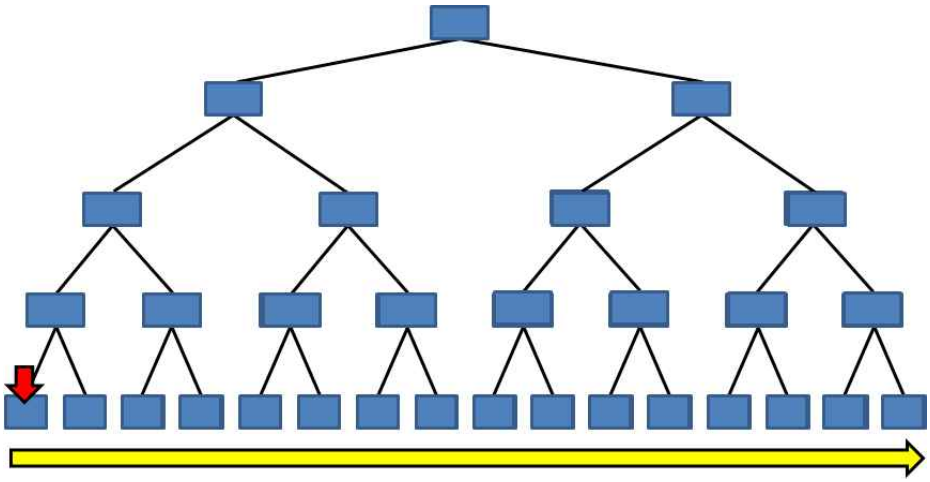

그림 4.1 원본 코드에서의 b-tree 탐색 과정

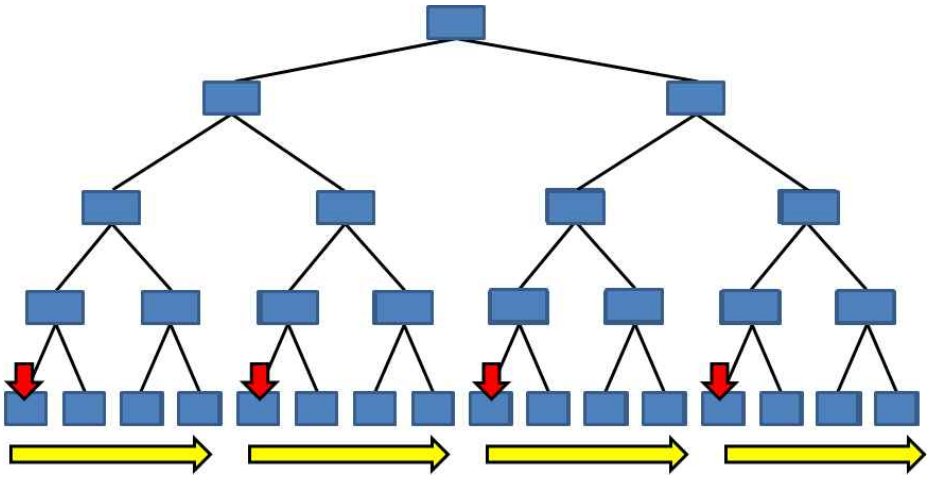

그림 4.2 수정된 코드에서의 b-tree 탐색 과정

기존의 읽기 방식에서는 하나의 cursor 에 의해 연속적인 주소를 순차적으로 읽었지만 수정된 코드에서는 여러 thread 를 이용해 여러 cursor 들에 의해 동시에 여러 곳을 읽을 수 있도록 수정하였다. b-tree 의 root 부터 leaf 쪽으로 한 level 에서의 node 의 수가 사용 가능한 thread 의 개수가 넘을 때까지 cursor 를 복제하며 tree 를 내려가고 사용 가능한 thread 가 모두 cursor 를 가질 때 각각의 cursor 를 해당 노드의 가장 왼쪽 자식 leaf 노드로 이동시킨다. 이렇게 분배된 cursor 들을 여러 thread 를 통하여 이어지는 leaf node 를 탐색하며 동시에 page read 요청을 disk 에 보내어 disk 읽기를 수행한다. 그림에서는 편의를 위해 이진트리 형태로 트리를 그렸고 4 개의 thread 를 사용할 때 빨간 화살표가 cursor 를, 노란 화살표가 cursor 의 진행방향을 나타낸다.

여기서 두 가지 장점이 있는데 첫 번째 장점은 기존의 하드 disk 에서 연속적으로 읽을 때 좋은 성능이 나오던 것과 달리 SSD 에서는 연속 읽기에 대한 이점이 크게 없으며 여러 읽기 요청이 동시에 수행될 수 있다는 점을 이용하는 것으로 기존의 긴 시간동안 반복적으로 수행되는 disk 요청 대신 짧은 시간동안 많은

양의 병렬적인 disk 요청을 수행하여 결과적으로 disk bandwidth 를 활용하고 disk 처리시간을 줄이는 것이다. 특히 memory 가 충분한 상황에서 그리고 table 전체를 읽는 full scan 을 수행하는 상황에서 가장 좋은 결과를 얻을 수 있다.

두 번째 장점으로는 기존의 코드에서는 nested loop 마다 바뀌는 table 과 cursor 를 보존하기 위하여 매 번의 disk 요청마다 cursor 의 불러오기와 저장을 수행해야 했는데 수정된 코드에서는 읽기가 시작될 때, 각각의 thread 가 cursor 를 가지고 하나의 table 의 모든 page 를 미리 읽게 되어 memory 에 올려놓기 때문에 cursor 를 임시적으로 보관할 필요가 없어졌고 cursor 의 불러오기와 저장을 매 disk 읽기마다 수행하지 않게 되었다.

여기에 추가적으로 새로 구현된 읽기 thread 를 주된 thread 와 비동기적으로 수행하므로써 disk 읽기 연산과 그 외의 함수 연산을 동시에 수행할 수 있다.

4.2 format 변환 개선

disk 에서 읽혀진 record 는 memory 상에 존재하는 buffer 에 올라가게 된다. 이 때 disk 그리고 memory 의 데이터는 innodb format 으로 저장되어 있는데 이것을 mysql 에서 사용하기 위해서는 mysql format 으로 변환하는 과정이 필요하다.

기존의 방식에서는 read record 함수 내에서 page read 가 완료되어 buffer 에 데이터가 올라가면 format 변환 함수가 수행되어 해당 데이터의 format 을 변환하였다. 위 과정은 모두 single thread 환경에서 수행된다.

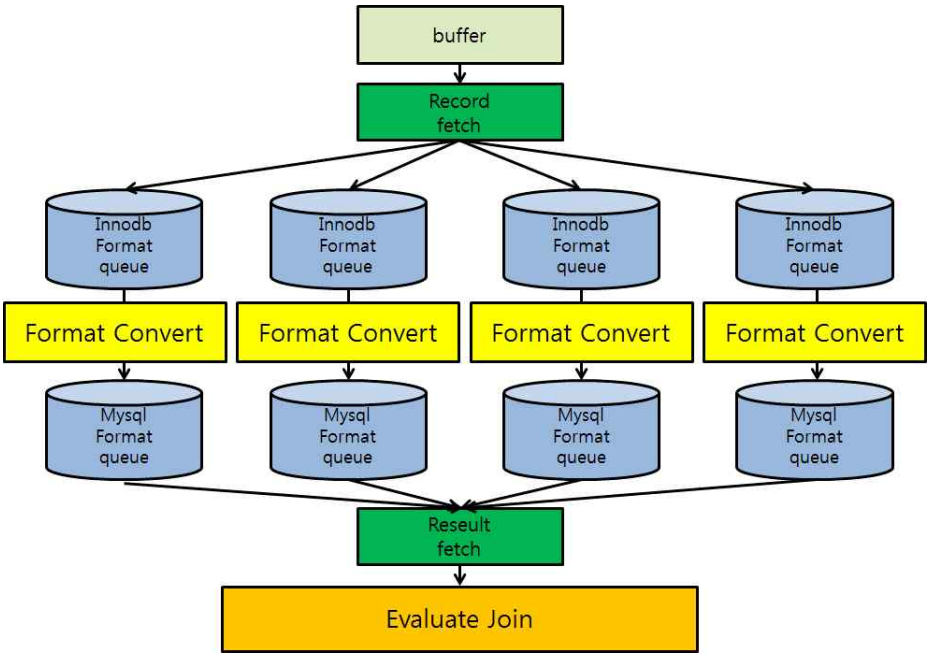

그림 4.3 개선된 format 변환 구조

개선된 방식에서는 핵심적인 데이터 format 과 format 변환 함수는 모두 그대로 사용하지만 single thread 를 multi thread 로 바꾸는 작업을 하였다. 이를 위해 여러 thread 에 데이터를 보내기 위한 fetch 함수, 변경 전 데이터를 저장하기 위한 queue 여러 개, 변경 후 데이터를 저장하기 위한 queue 여러 개, 완료된 데이터를 다시 모으는 함수를 추가하였다. 전체적인 흐름은 변환 전의 데이터를 fetch 함수를 통해 변환 전 queue 로 넣고 여러 개의 변환 함수들이 변환 전 queue 에서 데이터를 읽어 변환 후 queue 로 넘긴다. 그러면 데이터를 모으는 함수가 변환된 데이터를 변환 후 queue 에서 모아 결과값으로 낸다.

위 과정에서 하나의 변환함수 thread 마다 하나의 변환 전 queue 와 변환 후 queue 를 할당 받는다. 사용할 thread 의 수에 따라 queue 에 대한 추가적인 memory 할당과 추가적인 CPU core

사용이 필요하지만 multi thread 프로그래밍을 통하여 전체 format 변환 시간을 줄일 수 있다.

4.3 네트워크 처리 개선

기존의 mysql 에서는 nested loop join 을 통하여 생성된 record 에 대하여 해당 record 를 구성하는 각각의 field 에 단위로 protocol class 를 이용해 packet 으로 변환하고 NET 구조체에 packet 을 저장하여 모아진 packet 데이터로 VIO 함수를 사용해 socket 통신을 수행한다.

개선된 코드에서는 전체적인 네트워크 처리의 기본 흐름은 같지만 기존의 하나의 thread 와 달리 여러 thread 를 사용하고 thread 마다 protocol class, NET 구조체를 복제하여 record 를 packet 으로 변환하는 과정과 packet 으로 socket 통신을 수행하는 과정을 병렬화 한다.

위 과정을 통하여 전체적인 core 사용과 네트워크 처리의 속도가 상승될 것으로 기대했지만 병렬화를 위한 데이터 복사와 thread 의 동기화 overhead 가 예상보다 높아 성능 개선이 잘 이루어지지 않았다.

문제를 해결하기 위하여 thread 수의 조정, thread 가 사용할 record queue 의 크기 조정, field 단위의 병렬화, packet flush 의 조건 조정 등을 수행하였으나 근본적으로 innodb storage engine 에서의 nested loop 의 특성상 여러 table 을 조합하여 하나의 record 가 나오기 때문에 한 번에 한 record 만을 얻을 수 있었고 이것은 병렬화하는 데에 매우 제한적인 요소가 된다. 그러므로 병렬화를 효율적으로 수행하기 위해서는 storage 근본적인 nested loop

구조를 변경할 필요가 있어 보이며 이는 개선의 정도를 벗어난 것으로 본 논문에서 수행하기에는 적절하지 않은 것으로 보인다.

제 5 장 실험 및 분석

하나의 머신을 사용해 실험한다. CPU 는 2 * Xeon E5606 2.13GHz 4core 이고 memory 는 12G 이며 disk 는 Intel SSDPECME016T4 P3608 1.6TB NVMe PCIe 3.0 SSD 를 사용하였다.

성능 비교에서는 기존의 mysql 코드와 read-ahead 를 사용하지 않는 mysql, 수정된 mysql 코드에 대해서 disk 읽기와 query 문 수행 속도를 비교하였다. 사용된 데이터는 TPC-H 의 lineitem table 1.3GB 이고 사용된 query 는 SELECT * from lineitem 이다.

수정된 코드에서는 각각의 병렬 처리 단계마다 4 개의 thread 를 추가적으로 사용하였다.

5.1 disk 읽기 속도 실험

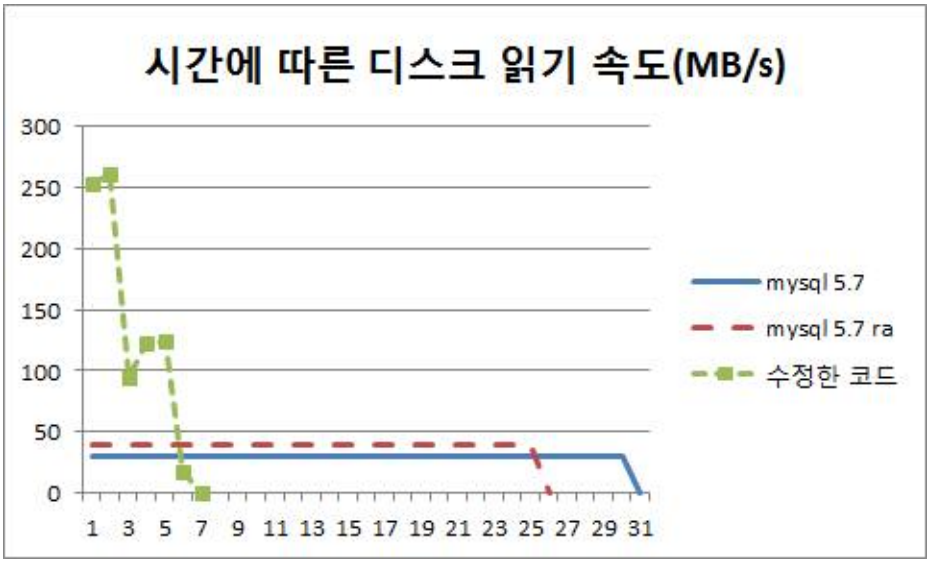

그림 5.1 disk 읽기 속도

mysql 5.7 원본의 경우 30MB/s 정도의 속도로 지속적으로 읽는다. 이것은 record 단위로 처리하는 mysql 의 특성에 따른 것으로 한 record 를 처리할 때마다 다시 read record 함수를 호출하고 disk 읽기가 일어난다.

mysql 5.7 read-ahead 의 경우, linear read-ahead 가 작동하는데 연속적인 읽기가 감지될 때 read-ahead thread 에서 연속적인 읽기를 연장해서 미리 읽는다. read-ahead thread 가 주된 thread 와 따로 동작하기 때문에 원본보다 약간 더 빠른 39MB/s 의 속도로 읽는다.

수정된 코드의 경우 읽기 요청이 시작될 때부터 여러 thread 가 b-tree 의 적절히 나눠진 leaf node 에서 병렬적으로 읽기를 시작하여 더 빠른 속도로 읽을 수 있고 데이터를 prefetch 하기 때문에 반복적인 요청 없이도 데이터를 buffer 에 미리 올려놓을 수 있다. 앞쪽의 속도는 빠르지만 뒤쪽의 속도는 불규칙적으로 떨어지는 것을

볼 수 있는데 이것은 thread 가 제각각 수행되고 종료되면서 일어나는 현상이다.

5.2 query 수행 속도 실험

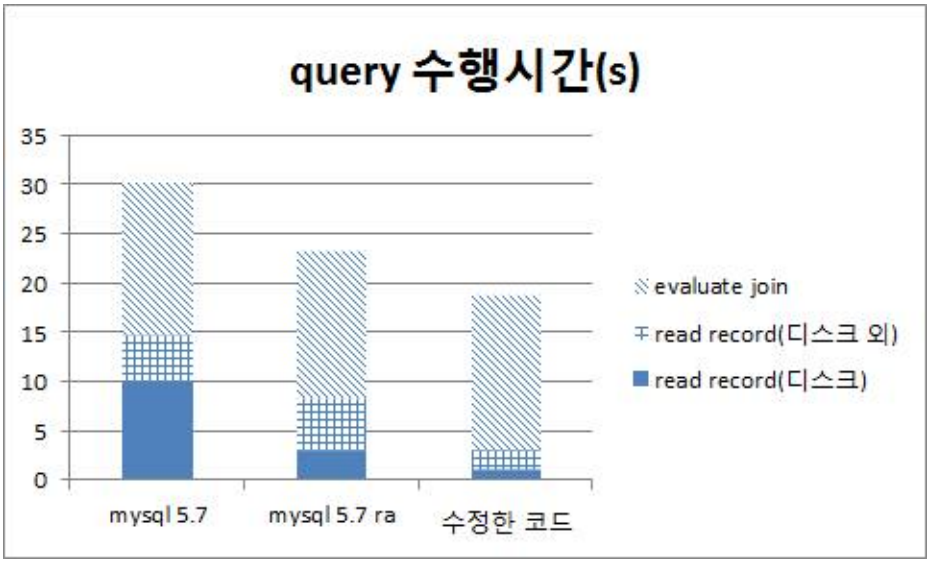

그림 5.2 mysql 함수 수행부분에 따라 걸리는 query 수행시간

query 수행시간은 다음과 같다. select query 를 수행하는데 mysql 5.7 원본은 총 30 초, read-ahead 는 23 초, 수정된 코드는 18 초가 걸렸다. 각각의 경우에 대해 크게 read record 에 걸린 시간과 evaluate join 에 걸린 시간으로 나눌 수 있고 read record 에 걸린 시간은 disk 처리에 걸린 시간과 그 외의 처리에 걸린 시간으로 나눌 수 있다.

read record(disk)를 보면 mysql 5.7 과 mysql 5.7 read-ahead, 수정한 코드가 각각 10 초, 3 초, 1 초의 성능 차이가 있다. read-ahead 의 경우, 주된 thread 와 따로 수행되는 read-ahead thread 가 disk 읽기를 수행했기 때문에 원본 코드보다 더 빠르게 측정되었고 수정한

코드의 경우 여러 thread 가 병렬적으로 그리고 주된 thread 와 비동기적으로 수행되었기 때문에 더 빠른 결과가 측정되었다.

read record(disk 외)를 보면 mysql 5.7 과 mysql 5.7 read-ahead, 수정한 코드가 각각 4.8 초, 5.2 초, 2.1 초의 결과가 나왔는데 이것은 mysql 5.7 과 read-ahead 는 이 부분에서 변화가 없으므로 비슷한 결과가 나왔고 수정한 코드는 format 변환과정에서 여러 thread 를 통하여 병렬적으로 처리했기 때문에 더 빠른 결과가 측정되었다.

evaluate join 부분은 네트워크 처리 부분인데 수정된 코드에서는 이 부분에서도 병렬 처리를 시도하였으나 데이터 복사 overhead 와 thread 동기화 overhead 가 너무 커서 성능을 향상시키지 못 하였고 수정된 코드를 적용시키지 못 하였다. 이 부분에서는 수정된 것이 없기 때문에 15.5 초, 15.1 초, 15.7 초로 세 코드 모두 수행시간이 비슷하게 측정되었다.

제 6 장 관련연구

Mysql 병렬화와 관련하여 query 를 나누어 병렬화시키는 기법이 있는데 이것을 query sharding 이라고 한다. 이것은 기존의 multi-thread 프로그래밍 되지 않은 mysql 을 프로그램 변경 없이 병렬화시킬 수 있는 방법으로 하나의 query 에 대하여 여러 query 로 쪼개어 mysql 에 여러 client 를 접속시켜 서로 독립적인 query 인 것처럼 수행하는 방법이다. 이렇게 수행하면 대부분의 경우에서 하나의 커다란 query 를 수행하는 것보다 좋은 결과를 얻을 수 있다. 하지만 모든 query 를 이런 식으로 나눌 수 있는 것은 아니며 복잡한 query 일수록 여러 query 로 쪼개는 것이 힘들 것이고 query sharding 을 수행하기도 어려울 것이다[2,4].

제 7 장 결론

이 논문에서는 흔하게 사용되는 오픈소스 RDBMS 인 mysql 를 가지고 multi-core 와 고성능 저장장치를 활용하기 위한 최적화 작업을 수행하였다. 고성능 저장장치를 활용하기 위하여 기존의 b-tree 탐색 및 읽기 요청 코드를 빠르게 수행할 수 있도록 수정하였고 mysql 과 innodb 사이의 format 변환에 multi-core 를 활용하기 위해 format 변환 부분에서 fetch thread, innodb format queue, mysql format queue, fetch result 함수 등을 통하여 병렬 처리 코드를 구현하여 여러 thread 를 통하여 여러 core 를 사용할 수 있도록 수정하였다.

위 내용으로 수정된 코드를 사용하여 full scan 의 select query 를 수행하였고 그 결과 기존의 mysql 5.7 코드보다 28%의 성능 향상을 측정하였다.

참고문헌

[1] “Mysql”, https://dev.mysql.com/

[2] https://mariadb.com/kb/en/library/shard-query/

[3] http://www.tpc.org/tpch/

[4] A Krasuski, MS Szczuka “Knowledge Driven Query Sharding“ CS&P, 2012

[5] http://www.olapcouncil.org

[6] Surajit Chaudhuri, Umeshwar Dayal, “An Overview of Data Warehousing and OLAP Technology” ACM SIGMOD Record Volume 26 Issue 1, March 1997 Pages 65-74

[7] Rini T. Kaushik, “FlashQueryFile: Flash-Optimized Layout and Algorithms for Interactive Ad Hoc SQL on Big Data ” HotStorage 2014

[8] Mitzi McCarthy Zhen He, “Efficient Updates for OLAP Range Queries on Flash Memory ” The Computer Journal, Volume 54, Issue 11, 1 November 2011, Pages 1773 - 1789

[9] Ioannis Alagiannis, Manos Athanassoulis, Anastasia Ailamaki, “Scaling up analytical queries with column-stores” Proceeding DBTest '13 Proceedings of the Sixth International Workshop on Testing Database Systems Article No. 8

ABSTRACT

In the current era of multi-core CPUs and high-performance storage devices, one of the traditional relational database management systems (RDBMSs), mysql is not making good use of evolving equipment. First of all, in the case of using multi-core CPU, existing mysql creates as many threads as the number of clients, and basically one thread can be used for one client. For high performance storage devices, the maximum bandwidth of the device can not be used at all and parallel processing capability is not available. The network is also not getting the maximum bandwidth by the CPU bottleneck.

The bandwidth utilization problems of the storage device and the network are caused by the CPU bottleneck, which is fundamentally caused by the lack of utilization of the core by the single thread programming of mysql. In this paper, we modify the existing mysql code to improve disk reading, record processing, and network processing by multi-thread programming.