CPU 오버헤드 분석을 통한 MariaDB와 PostgreSQL 성능 비교

이동호, 송민창, 조영태, 김승원
성균관대학교 컴퓨터교육과
e-mail : danny911kr@skku.ac.kr

Comparison of performance between MariaDB and PostgreSQL in terms of CPU overhead

Dong-Ho Lee, Min-Chang Song, Young-Tae Cho, Seung-Won Kim
Dept. of Computer Education, Sungkyunkwan University

요 약

IT기업뿐만 아니라 다양한 기업들이 빅데이터, 인공지능, 블록체인 등 많은 양의 컴퓨터 자원(CPU, RAM 등)을 요구하는 기술들을 서비스화 하고 있다. 따라서 한정된 자원으로 효율적인 서비스를 운영하는 것도 주요 이슈가 되고 있다. 본 논문에서는 오픈소스 RDBMS인 MariaDB와 PostgreSQL을 프로파일링하여 CPU 자원 효율성 관점에서 비교한다. 연구 결과 인터넷 서비스 환경에서 MariaDB가 PostgreSQL보다 버퍼 풀로 인해 페이지 캐시 참조율이 낮고, page fault 수가 적어 CPU 오버헤드가 더 작다는 것을 입증하였다.

1. 서론

스마트폰 기반 교통서비스를 제공하는 미국의 대표적인 스타트업 우버는 2016년 자사 DBMS를 기존 PostgreSQL에서 MySQL로 전환하였다. Uber Engineering은 자사 블로그를 통해 관련 Article을 게재하여 이에 대한 입장을 밝혔다.[1] 스토리지 엔진으로 InnoDB를 사용하는 MySQL이 PostgreSQL보다 성능이 더 좋으며 오버헤드가 작다는 내용이었다. 그러나 Article을 통해 제시한 몇몇 이슈에 대해 구체적인 증명을 밝히지 않아 PostgreSQL 사용자들 사이에 많은 반박이 있었다.

현재 IT기업뿐만 아니라 다양한 분야 기업들이 빅데이터, 인공지능, 블록체인 등 많은 양의 컴퓨터 자원(CPU, RAM 등)을 요구하는 기술들을 서비스화하고 있다. 이에 따라 한정된 자원으로 효율적인 서비스를 운영하기 위한 서버 자원 관리의 중요성이 커지고 있다. 위 기술들을 서비스로 제공하기 위해서는 서버 자원을 사용하는 주요 서버 구성요소로서 데이터베이스를 성능뿐만 아니라 CPU 이용률과 같은 서버 자원 활용도 관점에서 바라볼 필요가 있다.

위와 같은 관점에서 볼 때, Uber Engineering의 Article은 현재 가장 많이 활용되고 있는 오픈소스 데이터베이스인 MySQL과 PostgreSQL을 CPU의 자원 활용의 효율성 측면에서 비교함으로써 추가적인 연구 가치가 있다고 판단하였다. 하지만 MySQL은 유료인 Enterprise Edition과 무료인 Community Server 간 성능 차이가 있어, 무료 오픈소스 데이터베이스인 PostgreSQL과 비교하기에는 적절하지 않다. 따라서 MySQL과 동일 소스코드 기반인 MariaDB에 스토리지 엔진으로 InnoDB를 선택하여 비교를 진행하고자 한다.

이에 본 연구는 인터넷 서비스 환경을 조성 후 MariaDB가 PostgreSQL보다 CPU 자원 활용 효율성 측면에서 우수하다는 점을 두 데이터베이스의 context switching, page fault, 그리고 task-clock을 비교하여 입증하였다.

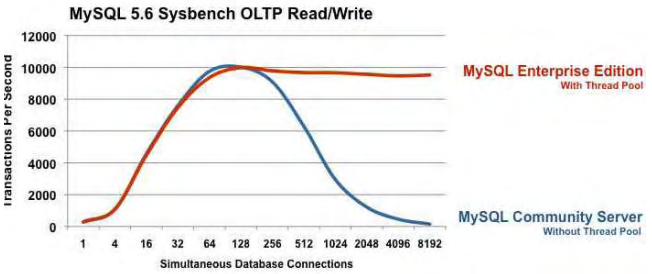
본 연구 구성은 다음과 같다. 우선 서론에서 연구 목적을 밝히고, 다음으로 선행연구를 이론적으로 고찰한다. 3장에서는 연구 절차를, 4장에서는 연구 결과를 제시하며, 마지막 5장에서는 연구결론을 요약하고 본 연구 의의와 한계를 논한다.

2. MariaDB와 PostgreSQL의 구조적 차이

MariaDB와 PostgreSQL은 여러 면에서 구조적인 차이가 있는데 대표적으로 연결 관리(Connection management)와 캐싱(Caching) 방식에서 큰 차이가 있다.

2.1. 연결 관리(Connection Management)

MySQL은 클라이언트에서 접속을 요청할 때마다 해당 요청을 수행하는 스레드를 생성한다. 더 나아가 MySQL Enterprise Edition과 MariaDB는 스레드 풀로 동시 접속을 관리하고 있다. (그림 1)의 MySQL 공식 홈페이지 벤치마크 결과는 스레드 풀이 접속 당 스레드보다 성능이 더 좋다는 것을 보여주고 있다.

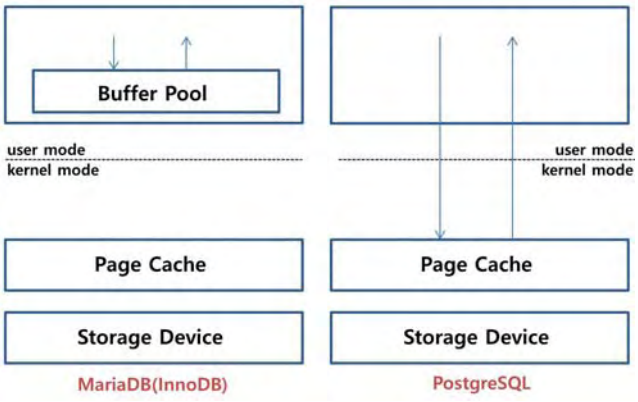


(그림 1) MySQL benchmark

반면 PostgreSQL은 클라이언트에서 접속을 요청할 때마다 해당 요청을 수행하는 프로세스를 생성한다. 연결 요청이 올 때마다 PostgreSQL Postmaster 프로세스가 child 프로세스를 fork하여 접속을 관리한다.

2.2. 캐싱(Caching)

디스크 캐시는 디스크의 일부 데이터를 RAM이 보유하는 소프트웨어 매커니즘이다. 이는 같은 데이터를 읽어야 할 시, 디스크에 접근하지 않고 RAM에서 바로 읽어올 수 있어 속도가 빠르다. 페이지 캐시는 리눅스 커널이 사용하는 가장 핵심적인 디스크 캐시로, 모든 데이터 페이지를 대상으로 동작한다. 또한, 리눅스 내 거의 모든 애플리케이션Read/Write 연산은 커널 내 페이지 캐시에 의존한다.[2] 따라서 MariaDB와 PostgreSQL 모두 데이터에 대한 Read/Write 연산을 할 시 페이지 캐시를 거치게 되는데 스토리지 엔진인 InnoDB는 (그림 2)와 같이 페이지 캐시에만 의존하는 것이 아닌 자체적으로 운용하는 버퍼 풀(buffer pool)을 통해 데이터와 인덱스를 캐싱한다.[2]



(그림 2) MariaDB와 PostgreSQL의 캐싱

3. 프로파일링

본 연구는 인터넷 서비스 환경을 가정하고 진행하였다. 이에 본 연구에서 프로파일링 하는데 사용한 시스템 환경은 <표 1>과 같다. 프로파일링 도구는 perf를 사용하였고, 서버 성능 테스트 도구로는 Apache Jmeter를 사용하였다.

CPU	I3-4150 3.5GHz
RAM	16GB
OS	Ubuntu 16.04.3
MariaDB	10.0.34
PostgreSQL	9.5.12

<표 1> 시스템 환경

3.1 프로파일링 과정

서버 내 RDBMS에 다수 사용자가 짧은 시간 동안 동시에 연결을 요청하는 상황을 가정하였다. 이에 Apache Jmeter를 <표2>와 같이 설정하여 서버에 SELECT 쿼리문을 요청하게 하였다. 이는 150초 동안 3000명의 사용자들이 서버에 SELECT 쿼리문을 요청하는 것과 같다.

Threads	3000
Duration	150(s)
Ramp-up	150(s)
Query	SELECT * FROM DB LIMIT 20

<표 2> Apache Jmeter 설정

이와 동시에 서버에서는 perf 명령어를 사용하여 Apache Jmeter를 통해 요청이 들어오는 150초 동안 샘플링을 진행하였다.

4. 연구 결과

아래 <표 3>은 150초 동안 각 RDBMS에서 발생한 context switching, page fault, 그리고 CPU 사용량 통계를 나타낸다. 여기서 task-clock은 해당 프로세스에 CPU가 배정되어 일을 한 시간을 뜻하며, CPU 사용량은 150초 동안 발생한 task-clock 비율을 뜻한다.

<표 3>을 토대로 PostgreSQL page fault 수가 MariaDB에 비해 매우 높으며, CPU 사용량 측면에서 PostgreSQL이 MariaDB에 비해 약 10배 높은 것을 알 수 있다. 여기서 발생한 page fault는 모두 디스크 접근이 필요 없는 minor page fault이었다.

	context-switches	page-fault	task-clock(msec) / CPU Utilized
MariaDB	19,011	518	712.456548 / 0.005
PostgreSQL	23,654	1,820,204	13388.487943 / 0.089

<표 3> 프로파일링 통계

RDBMS에서 context switching을 일으킨 함수들은 아래 <표 4>와 같다. MariaDB는 약 96%를 thread 스케줄링 하는데 사용하며, PostgreSQL은 약 87%를 process 스케줄링을 하는데 사용한다. 스케줄링을 할 때 context switching이 발생하지만, 이에 사용된 task-clock은 굉장히 작다.

	MariaDB	PostgreSQL
Context switching	96.61% schedule - 28.93% start_thread - 6.43% handle_connection_sockets 3.39% preempt_schedule_common	87.64% schedule - 42.82% PostmasterMain - 34.15% poll, select 12.36% preempt_schedule_common
Task clock	0.00% start_thread 0.00% preempt_schedule_common	0.00% PostmasterMain 0.00% preempt_schedule_common 0.2% poll

<표 4> context switching과 이에 대한 task-clock

PostgreSQL에서 page fault를 발생시킨 함수들과 page fault 처리 함수로 인해 발생한 task-clock은 <표 5>와 같다. 이는 page fault 빈도수를 기준으로 가장 비중이 높은 함수들이며, task-clock의 __do_page_fault 함수와 handle_mm_fault 함수는 page fault 처리 함수이다. 이를 토대로 PostgreSQL에서 발생한 page fault로 인해 발생한 task-clock은 약 1784.685msec이다.

PostgreSQL		
Shared Object	Task Clock	Page Fault
Postgres	2.94% __do_page_fault 0.63% handle_mm_fault 0.98% hash_search_with_hash_value 0.12% CreateTupleDescCopyConstr 0.08% createTemplageTupleDesc 0.07% heap_page_prune_opt 0.07% _bt_checkpage 0.03% pgstat_bestart	5.15% hash_search_with_hash_value 4.77% createTemplageTupleDesc 1.95% InitXLOGAccess 1.69% CreateTupleDescCopyConstr 1.62% _bt_checkpage 1.52% heap_page_prune_opt 1.20% pgstat_bestart
Libc-2.23.so Ld-2.23.so Libtasnl.so.6.5.1	5.33% 2.79% 0.29%	31.87% 7.25% 15.21%
	Postgres : 4.92% Libraries : 8.41% 총 : 13.33%	

<표 5> PostgreSQL page fault와 이에 대한 task-clock

RDBMS task-clock에서 페이지 캐시 관련 함수 비율은 <표 6>과 같다. 이를 토대로 RDBMS가 페이지 캐시를 참조하는데 사용한 시간은 MariaDB가 약 2.56msec, PostgreSQL이 약 262.414msec이다.

MariaDB	PostgreSQL	페이지 캐시 관련 함수	함수 설명
0.00% 0.07% 0.00%	0.05% 0.07% 0.02%	Find_get_entry Find_next_bit.part.0 Find_next_bit	페이지 캐시 엔트리를 탐색한다.
0.00% 0.29%	0.01% 0.88%	Pagecache_get_page Get_page_from_freelist	페이지캐시에서 페이지를 가져온다. 할당되지 않은 페이지 프레임을 찾아 다닌다.
0.00% 0.00%	0.03% 0.57%	Radix_tree_lookup Radix_tree_next_chunk	페이지 캐시는 Radix Tree 형태이므로, 탐색 과정에서 Radix Tree를 탐색한다.
0.00% 0.00%	0.14% 0.19%	Mark_page_accessed Copy_user_enhanced_fast_string	페이지 캐시에서 해당 데이터를 찾음을 의미한다. 유저 영역에서 페이지 캐시로 데이터를 복사한다.
총 0.36%	총 1.96%		

<표 6> 페이지 캐시 관련 함수들의 task-clock

5. 결론

본 연구는 인터넷 서비스 환경에서의 프로파일링을 통해 MariaDB가 PostgreSQL보다 CPU 자원 활용 효율성 측면에서 우수하다는 점을 입증하였다. 본 연구 결론은 다음과 같다.

첫째, MariaDB와 PostgreSQL의 context switching은 스레드와 프로세스 스케줄링을 할 때 발생한다. 하지만 context switching에 의한 task-clock은 현저히 낮아 이로 인한 오버헤드는 거의 없다.

둘째, MariaDB와 PostgreSQL에서 발생한 page fault는 모두 minor page fault였으며, PostgreSQL에서 발생한 대량의 page fault는 MariaDB 전체 task-clock의 두 배 이상에 해당하는 큰 오버헤드를 발생시켰다. 본 연구에서는 PostgreSQL에서 발생한 대량의 page fault 원인을 프로세스 기반 연결 관리 때문으로 보았다. Intel x86 등 대부분 CPU 아키텍처에서는 프로세스 스케쥴링으로 인한 context switching이 발생할 때마다 TLB flush가 일어나는 반면, 스레드 스케쥴링 시에는 TLB flush가 일어나지 않는다.[4] 이를 본 연구에 적용해보면, MariaDB는 하나의 프로세스 메모리를 공유하는 스레드들로 연결을 관리하는 반면, PostgreSQL은 자식 프로세스들을 이용하여 연결을 관리하기 때문에 프로세스들이 스케줄링 될 때마다 TLB flush가 일어난다. 이로 인해 TLB miss로 인한 minor page fault가 대량으로 발생한다고 보았으나 이를 입증하지는 못하였다.

셋째, MariaDB(InnoDB)가 제공하는 버퍼 풀 기능은 데이터베이스 페이지 캐시 참조 비율을 낮추어 페이지 캐시로 인한 오버헤드가 거의 없었다. 반면, PostgreSQL은 페이지 캐시를 참조하는데 262.414msec 가 소요되었는데, 이는 MariaDB 전체 task-clock인 712.456msec의 36%에 해당하는 상당한 오버헤드이다.

본 연구는 인터넷 서비스 환경에서 MariaDB와 PostgreSQL을 프로파일링하여 데이터베이스 자원 효율성에 대한 유의미한 검증을 제시한다. 연구 한계는 PostgreSQL에서 발생한 대량의 page fault에 대한 정확한 원인을 입증하지 못한 점이다. 향후 추가적인 연구를 통해 이를 입증하여 본 연구를 보완하고자 한다.

참고문헌

[1] UBER Engineering, https://eng.uber.com/mysql-migration

[2] 다니엘 보베이, 마르코 체사티. 『리눅스 커널의 이해』. 박장수 역. 서울: 한빛미디어, 2006.

[3] 송용주, 이민호, 엄영익. “Multiple Buffer Pool이 MySQL 성능에 미치는 영향 분석.” 2016년도 한국통신학회 동계종합학술발표회 논문집 (2016): 204-205.

[4] Yamada, Satoshi, Shigeru Kusakabe. "Effect of context aware scheduler on TLB." Parallel and Distributed Processing, 2008. IPDPS 2008.