Lock분산을 통한 key-value store 성능 분석

강민구[0], 이승현, 최종무
단국대학교 소프트웨어학과
best6693@naver.com, tmd1237@gmail.com, choijm@dankook.ac.kr

Performance Analysis of Lock Distribution on Key-Value Store

Mingu Kang[0], Seunghyun Lee, Jongmoo Choi

Dankook University
Department of Software

요 약

여러 웹 서비스 회사에서는 방대한 양의 데이터를 처리하기 위해 In-memory기반 key-value store 방식의 데이터 입출력 시스템을 지원한다. key-value 형식의 데이터는 멀티 스레드 환경에서 데이터 입출력의 원자성이 보장되어야 한다. 원자성을 보장하기 위해서 다양한 lock spectrum이 존재하며 본 논문에서는 lock spectrum을 coarse-grained lock과 fine-grained lock으로 구현하고 분산 해시 테이블을 통해 추가적인 성능 개선을 도모하였다. 실제 실험 결과에 따르면 fine-grained lock으로 구현 시 항상 성능이 개선되는 것이 아니며 적절히 lock을 분산시켰을 경우 coarse-grained lock보다 7배 성능이 개선되고 추가적인 개선 방법인 분산 해시 테이블을 통해 5배 이상 성능 개선 가능하다.

1. 서 론

현재의 여러 웹 서비스는 사용자의 데이터를 빠른 속도로 저장하고 요청하는 값을 빠르게 전달한다. 방대한 양의 데이터를 관리하기 위해 페이스북, 유투브, 구글에서는 In-memeory 기반 key-value형식의 분산 데이터 스토어 시스템을 사용한다. key-value란 고유한 하나의 key에 하나의 value를 가지고 있는 형태를 의미한다[1].

key-value는 해시 테이블을 통해 매우 빠른 속도로 데이터 입출력이 가능하지만 멀티 스레드 환경에서 데이터 입출력의 병렬성 문제가 발생하기 때문에 원자성이 보장되어야 한다[2]. 원자성 보장은 lock을 통해 해결 가능하다. 다양한 방법으로 lock을 걸어줄 수 있고 lock의 분산과 위치에 따라 프로그램의 성능 차이를 확인할 수 있다. 따라서 원자성을 보장하면서 프로그램의 높은 성능을 유지하는 것이 중요하다[3].

이에 본 논문은 멀티 스레드 환경에서 key-value형식의 데이터를 해시 테이블을 사용하여 입출력했으며 원자성을 보장하는 전제하에 lock의 위치와 개수를 변경하며 성능을 비교하는 실험을 했다. 다양한 lock spectrum을 만들고 각 성능을 Intel VTune Amplifier을 이용해 분석하였다. 실험의 결과로 coarse-grained lock보다 lock을 여러 개로 분산시켜 병렬성을 증가시킨 fine-grained lock이 처리율이 높아지는 것을 확인했다.

본 논문의 구성은 다음과 같다. 2장에서는 프로그램 실험 환경에 대해서 설명하고, 3장에서는 coarse-grained lock을 사용할 때 스레드 수의 변화에 따른 수행 시간을 비교한다. 4장에서는 3장에서 소개한 coarse-grained 형태의 lock을 fine-grained lock의 형태로 분산시켜 수행 시간을 비교한다. 5장에서는 해시 테이블의 개수를 증가시켜 추가적인 성능 개선을 확인한다. 6장에서는 실험의 결과 분석 및 향후 계획으로 마무리한다.

2. 실험환경 및 프로그램 설정

본 논문에서는 key-value형식의 데이터를 해시 테이블을 이용해 관리한다. 초기 해시 테이블의 크기는 50으로 지정했고 저장된 데이터의 수가 해시 테이블 크기의 70%를 초과하면 해시 테이블의 크기를 늘린다. Key 값의 범위는 50000개이며, 하나의 value는 8KB로 설정했다. 해시 테이블에서 발생하는 key의 충돌 해결은 Chaining 기법을 통해 처리했다.

멀티 스레드 환경에서 lock의 분산에 따른 처리 시간을 비교하기 위해 총 네 개의 스레드를 생성했고 각 스레드마다 105만개의 요청을 검색 10, 입력 10, 삭제 1 의 비율로 분배했다. 성능 측정을 위한 클라이언트로는 Intel VTune Amplifier를 사용하였다.

OS	ubuntu - 16.04.4 64bit
Kernel	4.13.0-38-generic
CPU	Intel(R) Core(TM) i7-7700HQ
RAM	16GB
SSD	NVMe SAMSUNG MZVLW256

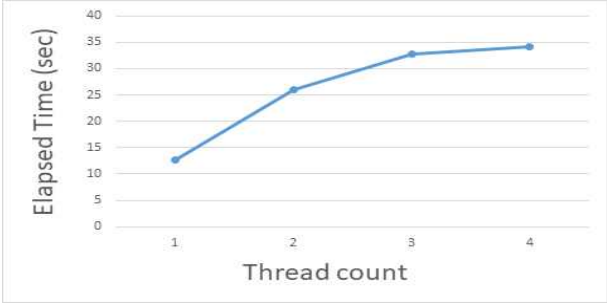
표 1 실험 환경

3. 스레드 개수 변화에 따른 수행시간 분석

Coarse grained single lock을 하나 선언해서 스레드 수 변화에 따른 수행 시간 변화를 측정하였다. 105만개의 요청을 보냈고 입력 50만개, 검색 50만개, 삭제 5만개의 비율로 설정하였다. [그림 1]에서 보이는 것처럼 single lock은 스레드가 두 개 이상인 경우 성능 병목이 되는 주 원인이다. 다수의 스레드가 하나의 lock을 사용해서 검색, 입력, 삭제의 원자성을 보장하기 때문에 lock 경합이

빈번하게 발생하게 되고 elapsed time이 증가하게 된다.

이미 많은 연구에서 증명된 것과 같이 lock 경합이 성능 저하를 일으키는 것을 알 수 있다[4][5]. 실제로 Intel VTune Amplifier를 사용해 프로파일링 해본 결과 대부분의 시간이 lock을 잡기 위해 block 상태에 빠져있는 것을 확인하게 되었다. 우리는 global single lock을 여러 개의 lock으로 분산시켜 경합을 피하면 병렬 성능 개선을 통한 애플리케이션 성능 확장성이 개선될 수 있음을 확인했다.



[그림 1] Single lock 하나에 대한 스레드 개수 변경 시 Elapse Time 비교

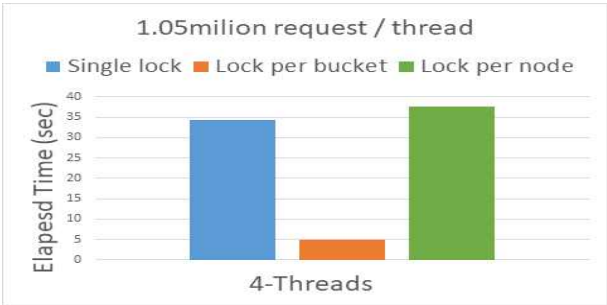
4. Lock의 분산에 따른 처리율 분석

lock의 분산에 따라 성능 향상에 변화가 있는지 확인하기 위해 아래 세 가지 방법으로 lock을 분산한다. 주 분산 방법일 뿐 이외에도 필요한 최소한의 lock을 사용한다.

(a) Coarse-grained (b) Fine-grained

single lock / lock per bucket / lock per node

(a)로 갈수록 적은 수의 lock을 사용하였고 (b)로 갈수록 많은 수의 lock을 사용하였다. 다수의 lock을 사용하면 병렬성이 보장되어 시간이 짧게 걸릴 것 같지만 실제로 수행 시간을 측정하였을 때 예상과 다른 것을 확인할 수 있다.



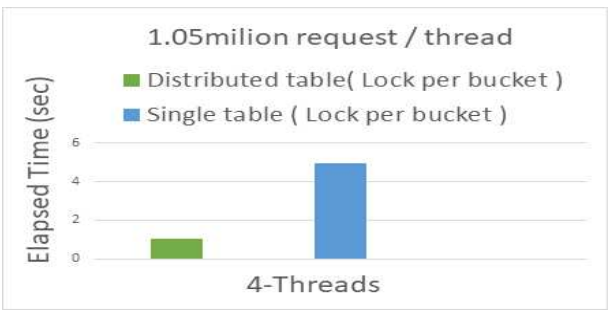
[그림 2-1] Lock분산에 따른 Elapse Time 비교

Lock per node의 경우 노드를 순회하는 동안 현재 위치한 노드에 lock을 걸어줌으로써 데이터 입출력의 원자성을 보장한다. 매 노드마다 lock을 잡아주기 때문에 lock, unlock의 호출이 잦아 lock 분산의 이점보다 lock 사용의 overhead가 크다[6]. [그림 2-2]에서 확인할 수 있듯 각 노드마다 lock을 걸어주게 되면 병렬성이 증가하여 사용자의 요청을 처리하는 스레드의 평균 wait time이 줄어들지만 전체 수행시간은 single lock을 사용했을 때보다 길어지는 것을 확인할 수 있다. 반면에 bucket 단위로 lock을 걸어주게 되면 lock 경합을 줄이는 동시에 lock 사용의 overhead 또한 줄어들어 앞서 두 lock 분산 방법에 비해 elapsed time이 약 7배 단축되었음을 알 수 있다.

[그림 2-2] 스레드의 평균 wait time 비교

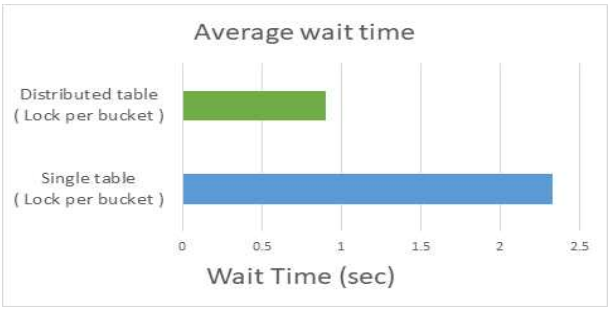
5. 분산 데이터 스토어를 통한 성능 개선

가장 높은 성능을 보인 lock per bucket의 성능을 개선하도록 방법을 모색한 결과 병렬 성능을 고도화 시킬 수 있도록 해시 테이블의 수를 두 개로 증가시켰다. 데이터의 범위에 따라 서로 다른 해시 테이블에서 데이터를 관리함으로써 lock 경합을 줄일 수 있다.



[그림 3-1] 해시 테이블 분산에 따른 Elapse Time 비교

2개의 해시 테이블을 생성해 각각의 해시 테이블에서 서로 다른 lock을 사용하여 원자성을 보장하며 데이터 입출력을 병렬적으로 수행한 결과 약 5배의 성능 개선을 확인할 수 있다. 방대한 양의 데이터를 나누어 관리하면서 경합이 줄어들 뿐만 아니라 해시 테이블을 더욱 효율적으로 사용하여 elapsed time의 확연한 차이를 볼 수 있다. [그림 3-2]에서 확인할 수 있듯 스레드가 wait 상태에 머무르는 평균 시간을 확인해본 결과 상대적으로 distributed table의 wait time이 약 2배 이상 짧다. 이를 통해 여러 개의 스레드가 병렬적으로 데이터 입출력을 수행한 결과 5배 이상의 성능 개선을 확인할 수 있다.



[그림 3-2] 스레드별 평균 wait time 비교

6. 결론

본 논문에서는 멀티 스레드 환경에서 key-value store의 원자성 보존을 위해 다양한 lock 기법을 사용했으며 global single lock을 사용하는 것 보다 lock을 분산시킴으로서 병렬성을 증가시키고 성능 향상을 할 수 있음을 확인했다.

이러한 결과들을 기반으로 memcached, redis와 같은 오픈소스 In-memeory 기반 key-value형식의 분산 데이터 스토어 시스템의 성능 개선에 기여할 수 있을 것이다.

* 본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 SW중심대학지원사업의 연구결과로 수행되었음(2017-0-00091)

참고문헌

[1] Kyung-Tae Song and Sang-Hyun Park, “A Recent Trend of Database for Big Data Handling using Key-value databse”, *The Journal of Korean Institute of Information Technology*, vol. 12, No. 1, pp. 48-50, 2017.

[2] Michael kerrisk. (2010). *THE LINUX PROGRAMMING INTERFACE*(pp.631-653). No Starch Press.

[3] A. Dudás, S. Juhász, and S. Kolumbán, “Performance analysis of multi-threaded locking in bucket hash tables,” ANNALES Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae Sectio Computatorica, vol. 36, pp. 63-74, 2012.

[4] Joo-Hyun Kyong and Sung-Soo Lim, “An Analysis of Lock Contention in Linux Memory Management for Many-core System”, *The Journal of Korean Institute of Information Scientists and Engineers*, pp. 1571-1573, 2015.

[5] Euihyeok Kim and Min-Soo Kim, “Performance Analysis of Modern Hashing Techniques for Multi-core CPUs”, *The Journal of Korean Institute of Information Scientists and Engineers*, pp. 189-201, 2013.

[6] Nathan R. Tallent, John M. Mellor-Crummey, Allan Porterfield, “Analyzing Lock Contention in Multithreaded Applications”, *Association for Computing Machinery*, pp. 1-4, 2010.