매니코어 환경에서 스킵 리스트에 적용된 논블로킹 동기화 기법의 성능비교

Performance Comparison of Non-Blocking Synchronization Mechanisms Applied to Skip List in Manycore Environment

요 약

매니코어 시스템에서 전통적인 동기화 기법이 갖는 성능 향상의 한계를 해결하기 위해, 논블로킹 방식의 알고리즘들이 제안되었다. RCU는 대표적인 논블로킹 방식 동기화 기법으로 널리 사용되고 있지만 프로그래밍 복잡성과 쓰기연산의 성능 저하 문제가 있다. 이와 같은 한계를 극복하기 위해 최근 Multi Version Concurrency Control (MVCC) 기반으로 동작하며 트렌잭션 방식의 보다 쉬운 API를 제공하는 MV-RLU 동기화 기법이 소개되었다. 본 논문에서는 LSM 트리 기반 Key-Value Store에서 각광 받고 있는 Skip list 자료구조에 RCU와 MV-RLU 동기화 기법을 적용하여 매니코어 환경에서 성능 평가 및 분석을 하였다. Zipf 분포의 랜덤 키 생성 워크로드에서 MV-RLU가 RCU에 비해 최대 2.5배 이상의 스레드 동시성 향상 결과를 보였다.

1. 서 론

매니코어 프로세서의 도입으로 컴퓨터 시스템의 병렬성은 증가되었지만 전통적 동기화 기법의 한계로 코어 수에 비례하는 성능 향상을 보이지 못한다. Spinlock, Mutex, Readers-writer lock 등의 전통적 동기화 기법을 사용하면 암달의 법칙에 따라 코어가 증가하여도 Lock에 의한 임계영역으로 작업들이 블록 되기 때문에 성능 향상에 한계가 생긴다[1].

매니코어 시스템에서 성능 한계가 발생하는 또 다른 이유는 매니코어 하드웨어의 캐시 동기화로 인한 부하이다. 전통적 동기화 기법이 스레드 간에 공유하는 동기화 객체는 스레드가 임계영역에 진입할 때마다 그 값이 변한다. 이는 임계영역에 진입하기 위해 블록된 스레드들의 캐시를 모두 갱신해야 함을 뜻한다. 코어수가 많은 시스템에서 이러한 부하는 성능에 치명적이다[2].

이와 같은 전통적 동기화 기법의 문제를 해결하기 위해 논블로킹 방식의 동기화 기법이 제안되었다[3, 4]. 대표적 논블로킹 알고리즘인 Read-Copy Update(RCU)는 한 스레드가 공유 자료구조를 변경하는 중에도 읽기 연산을 수행하는 스레드들의 동시 실행을 가능하게 한다. 이러한 특성으로 인해 RCU는 쓰기 연산의 비율이 적은 상황에서는 성능 확장성을 보이지만 쓰기 연산을 Mutex로 동기화하기 때문에 쓰기 연산의 비율이 증가하면 성능이 감소하는 문제가 있다. 또한, 수정 연산 시에 단일 포인터에 대한 원자적 연산만 지원하므로 복잡한 자료구조의 수정 시 프로그래밍이 복잡해지는 단점이 있다. MV-RLU (Multi Version Read Log Update)는 다중 읽기뿐만 아니라 다중 쓰기 연산도 블로킹 없이 수행할 수 있는 논블로킹 기법이다. 다중 버전을 생성하여 여러 스레드가 각자 해당 버전을 동시에 접근하는 방식으로 스레드 동시성을 높이는 구조이다. 그리고 트랜잭션 방식의 쉬운 프로그래밍 인터페이스를 제공한다[4, 5].

Skip list는 탐색, 삭제, 삽입이 확률적으로 O(log(N))이면서도 B+Tree에서 발생하는 자료구조 재조정의 부하가 없기 때문에 LSM 트리 기반 Key-Value Store를 포함하여 응용프로그램에서 각광받고 있다[6]. 매니코어 환경에서 전통적인 동기화 기법에 기반한 Key-Value Store를 사용하는 경우, Skip list는 성능 확장성이 없다.

대표적인 논블로킹 기법인 RCU, MV-RLU를 Skip list에 적용하는 것으로 동기화 성능 향상을 기대할 수 있음에도 관련 선행 연구가 적다. 본 논문에서는 Skip list에 논블로킹 동기화 기법인 RCU, MV-RLU를 적용하여 매니코어 환경에서 삽입, 삭제, 탐색 연산의 성능을 평가하였다. MV-RLU를 적용한 Skip list는 Zipfian 분포의 랜덤키 생성 워크로드에서 RCU보다 최대 2.5배 이상의 성능 향상을 보였다.

2. 관련 연구

본 연구 이전에도 매니코어 환경에서 확장성 있게 동작하는 Skip list를 구현, 실험한 연구들이 있다[7,8]. JellyFish는 Key-Value Store인 MongoDB의 Skip list를 MVCC (Multi Version Concurrency Control) 방식의 기법

을 적용해 성능을 개선했다[7]. Jiffy 역시 MVCC 기반의 Skip list로써 배치 연산과 자료구조의 스냅샷을 이용해 최적화된 범위 탐색을 구현하였다[8]. 논블로킹 방식의 동기화 기법인 MV-RLU와 RCU도 Skip list 자료구조의 동시성을 향상시킬 수 있는 방안임에도 불구하고 선행 연구가 많지 않아 본 논문에서 이를 평가 및 분석한다.

3. 구현

3.1 RCU Skip list

Skip list는 한 노드에 접근할 수 있는 경로가 하나 이상이므로 RCU가 요구하는 단일 포인터 연산으로는 Skip list의 수정을 마칠 수 없다. 대신 노드별 Mutex와 Garbage Collector를 이용한 알고리즘[9]을 응용해 구현하였다. RCU는 읽기 연산을 위해 임계영역에 진입하는 쓰레드에게 RCU_READ_LOCK을 호출하게 한다. RCU는 쓰레드가 RCU_READ_UNLOCK을 호출하기 전에 참조한 객체가 제거되지 않는 것을 보장한다. 참고한 알고리즘[9]은 Java 언어를 대상으로 개발한 것으로 참조가 존재하면 그 객체가 제거되지 않는 것을 전제로 한다. 이를 RCU의 RCU_READ_LOCK과 RCU_READ_UNLOCK API를 이용해 Java의 Garbage Collector를 대체하였다.

3.2 MV-RLU Skip list

MV-RLU는 API와 구현 가이드라인이 명확히 정의되어 있다. 객체를 읽기 위해서는 MVRLU_READ_LOCK을 호출해 스레드 자신이 자료구조에 접근한 상태임을 알리고 자료구조에 접근한 시점을 측정한다. 이후 객체를 읽기 위해서는 MVRLU_DEREF를 이용해 자신이 자료구조에 접근한 시점에 부합하는 객체의 버전을 탐색한다. 객체를 수정하는 경우에는 MVRLU_TRY_LOCK을 이용해 객체의 수정 권한을 얻고 해당 스레드의 로그에 복사본을 생성한다. 스레드는 복사본을 수정 완료한 뒤 다른 스레드에게 공개(commit)한다.

4. 성능 평가

실험 환경은 Intel® Xeon® Gold CPU 두 개를 이용해 구성하였다. 자세한 사항은 표 1에 나타내었다.

실험은 탐색 연산의 비율을 전체 연산의 50%로 Skip list에 삽입, 삭제, 탐색 세 가지 연산을 10초 동안 수행한 횟수를 측정하였다. 연산에 사용하는 랜덤 키값을 Uniform 분포, Zipfian 분포(지수값 s: 0.3, 0.6, 0.9)로 하여 총 네 번의 실험을 진행했다.

CPU	Intel® Xeon® Gold 6258RCPU@2.70GHz
코어수	동일 CPU 2개 (112 logical cores)
Memory	376GB
OS	5.15.0-67-generic #74~20.04.1-Ubuntu
Compiler	clang version 10.0.0-4ubuntu1

표 1 실험 환경

그림 1, 2, 3, 4는 탐색, 삽입, 삭제 연산의 초당 평균 수를 모두 더한 값이다. 벤치마크 결과에 대한 증거를 제시하기 위해 Perf를 이용해 CPU Cycle을 측정하였다. 그 결과는 표 2, 3에 제시되어 있다. 표 3에 기재된 find는 Skip list에 삽입, 삭제할 때 수정할 노드들을 찾는 함수이다. mutex_trylock은 MVRLU_TRY_LOCK 함수가 노드의 수정권한을 얻기 위해 사용하는 함수이다.

RCU는 그림1, 2, 3에서 모두 확장성 있는 성능을 보여준다. 이러한 결과는 RCU를 Skip list에 적용하기 위해 사용한 알고리즘[9]이 노드별 Mutex (Fine-grained lock 방식)를 사용 해 최적화하였기 때문이다. 수정이 필요한 노드만 Mutex를 얻으므로 경합이 낮으면 RCU의 수정 연산을 병렬적으로 처리할 수 있어 확장성 있는 성능을 낼 수 있다. 표 2를 보면 RCU를 적용한 Skip list 구현은 Mutex를 사용함에도 불구하고 Mutex에 소모되는 CPU Cycle이 높게 측정되지 않았다. MV-RLU는 그림 1, 2, 3에서 60개 스레드를 기점으로 MV-RLU의 성능이 RCU에 비해 근소하게 낮다. 이는 MV-RLU가 자료구조를 읽기 위해 버전 체인을 탐색하는 부하가 있지만 RCU는 읽는 연산에 추가적인 부하가 없기 때문에 발생하는 성능 격차이다. 표 3을 보면 모든 경우에 MV-RLU 벤치마크가 소모하는 CPU Cycle의 가장 상위에 MVRLU_DEREF 함수가 나타난다. 이는 MV-RLU 벤치마크가 버전 탐색에 많은 CPU Cycle을 소모함을 의미한다.

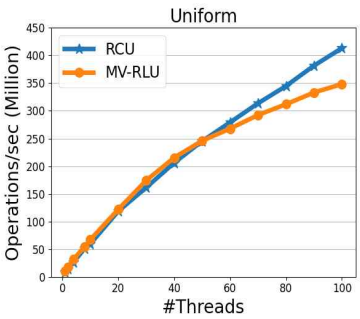

그림 1: Read 50%, Uniform

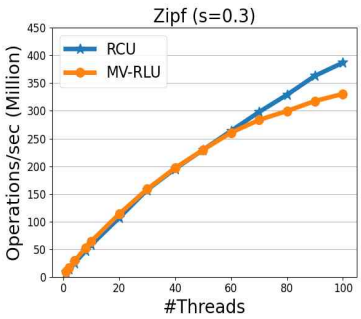

그림 2: Read 50%, Zipf(s=0.3)

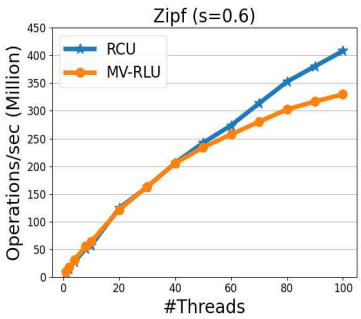

그림 3: Read 50%, Zipf(s=0.6)

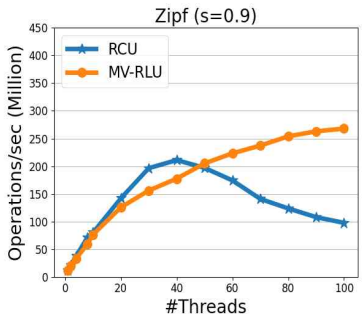

그림 4: Read 50%, Zipf(s=0.9)

가장 경합이 심한 Zipfian (s=0.9)인 그림 4에서 RCU의 성능은 40개 스레드를 기점으로 감소한다. Skip list에 특정 부분에 높은 경합이 발생하면 RCU의 수정 작업 동기화를 위해 사용한 Mutex의 영향으로 스레드 수가 증가함에도 오히려 성능이 감소하게 된다. 이러한 현상은 표 2에 Zipf (s=0.9)의 70개 스레드 항목에 확인할 수 있다. 다른 경우에 상위에 나타나지 않는 Mutex에 대기하는 연산이 34.11%로 가장 많은 CPU Cycle을 소모하였다. 반면에 MV-RLU는 그림 4의 실험에서도 확장성 있는 성능을 낸다. MV-RLU는 읽기 연산뿐만 아니라 삽입, 삭제 연산도 객체의 여러 버전을 생성하여 각 스레드가 해당 버전을 읽기, 수정하므로 Skip list의 특정한 구역에 연산이 집중되어도 스레드 수에 비례해 성능이 증가한다. 그림 4의 실험 100개 스레드인 결과에서 MV-RLU는 RCU보다 2.5배 더 높은 성능을 보였다.

	40 Threads	70 Threads
Uniform	탐색(38.33%) 삽입(23.01%) 삭제(21.82%)	탐색(37.48%) 삽입(22.56%) 삭제(22.41%)
Zipf (s=0.3)	탐색(35.44%) 삽입(21.45%) 삭제(20.16%)	탐색(34.83%) 삽입(21.16%) 삭제(19.83%)
Zipf (s=0.6)	탐색(34.49%) 삽입(20.91%) 삭제(19.58%)	탐색(33.74%) 삽입(20.67%) 삭제(19.42%)
Zipf (s=0.9)	탐색(26.76%) 삽입(18.46%) 삭제(15.91%)	Mutex(34.11%) 삽입(13.46%) 탐색(12.48%)

표 2 RCU Perf 상위 세 가지 CPU 소모 항목

	40 Threads	70 Threads
Uniform	MVRLU_DEREF(42.83%) find(10.36%) mutex_trylock(5.32%)	MVRLU_DEREF(40.40%) find(9.52%) mutex_trylock(4.90%)
Zipf (s=0.3)	MVRLU_DEREF(39.54%) find(9.55%) mutex_trylock(4.95%)	MVRLU_DEREF(37.75%) find(8.83%) mutex_trylock(4.58%)
Zipf (s=0.6)	MVRLU_DEREF(38.94%) find(9.26%) mutex_trylock(5.00%)	MVRLU_DEREF(37.82%) find(8.46%) mutex_trylock(4.60%)
Zipf (s=0.9)	MVRLU_DEREF(43.23%) find(7.60%) mutex_trylock(5.95%)	MVRLU_DEREF(44.57%) find(7.57%) mutex_trylock(6.82%)

표 3 MV-RLU Perf 상위 세 가지 CPU 소모 항목

5. 결론

본 논문에서는 널리 사용되는 자료구조인 Skip list에 대표적인 논블로킹 동기화 알고리즘인 RCU, MV-RLU를 적용하고 그 성능을 비교하였다. Skip list에 접근하는 키 값의 분포가 균일하거나 적게 편차된 경우(Zipfian s=0.3, 0.6), 스레드 수가 증가함에 따라 RCU의 성능이 MV-RLU보다 좋게 나타났다. 반면에 Zipfian (s=0.9) 분포를 따르는 키값으로 Skip list에 접근한 실험에서는 MV-RLU가 RCU보다 최대 2.5배 더 높은 성능을 보였다.

참고문헌

[1] Eyerman, S., & Eeckhout, L. Modeling Critical Sections in Amdahl's Law and Its Implications for Multicore Design. SIGARCH Comput. Archit. News, 38(3). doi:10.1145/1816038.1816011

[2] Boyd-Wickizer, S., Kaashoek, M. F., Morris, R., & Zeldovich, N. Non-scalable locks are dangerous. In Proceedings of the Linux Symposium.

[3] McKenney, P. E., & Walpole, J. What is RCU, fundamentally?. Linux Weekly News (LWN.net).

[4] Kim, J., Mathew, A., Kashyap, S., Ramanathan, M.K., & Min, C. Mv-rlu: Scaling read-log-update with multi-versioning. In Proceedings of the Twenty-Fourth International Conference on ASPLOS.

[5] Kim, C., Choi, E., Han, M., Lee, S., & Kim, J. Performance Analysis of RCU-Style Non-Blocking Synchronization Mechanisms on a Manycore-Based Operating System. Applied Sciences, 12(7), 3458.

[6] Luo. Chen, "LSM-based storage techniques: a survey", The VLDB Journal, 2020.

[7] Yeon, J., Kim, L., Han, Y., Lee, H. G., Lee, E., & Kim, B. S. JellyFish: A Fast Skip List with MVCC. In Proceedings of the 21st International Middleware Conference.

[8] Kobus, T., Kokociński, M., & Wojciechowski, P. T. (2022, April). Jiffy: a lock-free skip list with batch updates and snapshots. In Proceedings of the 27th ACM SIGPLAN.

[9] Herlihy, M., Lev, Y., Luchangco, V., & Shavit, N. A simple optimistic skiplist algorithm. 14th International Colloquium, SIROCCO 2007, Castiglioncello, Italy, June 5-8, 2007. Proceedings 14. Springer Berlin Heidelberg.