Practical and Efficient Lock-Free Garbage Collection Based on Reference Counting

본 논문은 Lock-free 동적 데이터구조와 함께 사용하기 위한 reference counting을 기반으로 garbage collection체계의 구현을 제시한다.

1장

동적 원자적 데이터 구조를 구현하기 위해 메모리 관리는 필수적이다. 일반적으로 상호배제를 기반으로 하는 알고리즘으로 메모리 관리를 하는데, 상호배제는 blocking을 유발하므로 교착상태, 우선순위 반전, 기아와 같은 심각한 문제가 발생할 수 있다. 반면에 lock-free알고리즘으로 이러한 문제를 해결할 수 있다. Non-blocking 알고리즘에서는 동시 작업의 효과가 뒤얽힌 프로세스에서 일관된 방식으로 관찰되어야 하고, 일반적으로 일관성 요구사항을 **선형화**라고 한다.

알고리즘 동작이 lock-free가 되려면, 모든 서브 동작이 lock-free여야 한다. 따라서, lock-free 동적 데이터 구조에는 일반적으로 lock-free 메모리 관리가 필요하다. 메모리 관리문제는 동적 메모리 할당과 garbage collection으로 구분된다.

다양한 lock-free garbage collection체계가 있다. Michael & Herlihy의 Double-width CAS를 기반으로 하는 체계는 일부 32비트 아키텍처에서 사용할 수 있지만 현재 64비트 아키텍처 중 일부에서만 사용할 수 있고, 스레드에서 로컬 포인터의 안전성만 보장하므로 개체에 대한 전역 참조를 항상 신뢰할 수 있어야 하는 lock-free알고리즘을 지원할 수 없다. 또, 동시성의 수준 증가로 인한 큰 성능저하와 큰 데이터구조의 순회를 재시도하는 데이터 구조 알고리즘을 야기할 수 있다. Valois et al.은 원자 프리미티브를 사용하여 lock-free reference counting을 구현했지만 메모리할당을 위한 알고리즘과 사용하도록 제한되어 있다. Detlefs et al.은 재생 메모리를 임의로 재사용 할 수 있는 체계를 제시했지만, 현재 아키텍처에서 사용할 수 없는 double-words CAS를 기반으로 한다. Valois et al.에서 Reference counting 기법은 잠재적으로 스레드로부터 참조가 임의의 노드를 재할당하는 것을 차단할 수 있음을 확인했다.

본 논문은 각각의 단점을 피하면서 관련 기술의 장점을 유지하는 목적으로, 일반적인 lock-free reference counting 방식으로, Hazard pointer의 효율성과 reference counting의 장점을 결합한다. 이는 ‘선형화’이며, 메모리할당에 대한 임의의 체계와 호환하고, 일반적으로 사용되는 원자 프리미티브를 사용하여 구현할 수 있으며, 지역뿐만 전역의 참조도 안전을 보장할 수 있다.

논문은 2장 구현의 목표로 하는 시스템의 유형 설명. 3장 garbage collection문제의 세부사항 설명. 4장 실제 알고리즘. 5장 구현에 대한 연산의 정확한 의미를 정의, lock-free와 선형화 증명, 정확성. 6장 lock-free데이터 구조와 관련하여 새로운 체계의 실험적 평가. 7장 마무리로 구성되어 있다.

2장

시스템의 각 노드는 로컬 메모리와 함께 프로세서를 포함하며, 모든 노드는 상호 연결 네트워크를 통해 공유메모리에 연결된다. 각 작업은 프로세서 중 하나에서 순차적으로 실행되는 반면 각 프로세서는 한번에 많은 작업을 수행할 수 있고, 다른 프로세서에서 실행중인 협력작업은 공유메모리에 내장된 공유 데이터 객체를 사용해 조정, 통신한다. 캐시 일관성 공유메모리에서 서브작업을 통해 공유 데이터 객체에서 작업을 동기화 하는데, 공유 메모리는 시스템의 모든 노드에 대해 균일하게 접근할 수 없고 메모리의 다른 부분에 대해 다른 접근시간을 갖는다. 공유메모리 시스템은 동기화를 위해 단일 메모리 단어의 원자적인 작업과 원자적 프리미티브를 지원해야 한다. 이 논문에서는 FAA와 CAS를 사용한다.

3장

이 논문에서는 동적 lock-free 데이터구조의 맥락에서 garbage collection문제를 해결을 목표로 한다. Lock-free데이터 구조는 데이터를 갖는 노드로 구성되고, 이런 노드는 임의의 패턴으로 포인터를 이용해 서로 참조하여 상호 연결된다. 각 노드는 다른 노드를 참조하는 **링크**라 하는 임의의 수의 포인터를 포함할 수 있다. 링크를 통해 참조된 노드를 **역참조**라고 한다. 동적 & 동시성 데이터 구조에서 노드는 데이터 구조에서 지속적으로 추가/제거될 수 있는데, 시스템의 메모리양이 제한되어 있으므로 노드의 점유된 메모리는 시스템에서 동적으로 할당/재생되어야 한다. 노드가 삭제될 때 시스템에 명시적으로 재생된다. 동시 환경에서는 재사용된 노드이 메모리에 대한 접근 가능성이 데이터구조와 시스템의 정확성에 치명적이기 때문에 쓰레드의 로컬 참조도 포함해야 한다.

재사용을 올바르게 결정하는 논리 단위를 garbage collection(GC)이라 하며 다음 특성이 있어야 한다.

1. GC는 데이터구조의 일부가 아니며, 쓰레드에 의해 향후에 접근이 불가능한 garbage만 회수해야 한다.
2. 언제라도, 데이터 구조의 일부는 아니지만 시스템으로 재사용되지 않은 노드의 수에는 제한이 있어야한다.

GC 구현은 일반적으로 관련 쓰레드와 상호작용하고 노드의 접근에 제한을 두어야 한다.

구현하려는 데이터 구조는 Lock-free이고 선형화 가능하므로, GC는 다음을 보장해야 한다.

1. 데이터 구조와 통신하기 위해 GC의 모든 동작은 lock-free여야하고 선형화 되어야 한다.

데이터구조에 대해 시스템의 총 점유 메모리를 최소화 하기위해 다음 속성을 달성해야 한다.

1. GC에서 회수한 메모리는 재사용을 위해 접근할 수 있어야 하기 때문에, 기본 메모리 할당자와 호환되어야 한다.

동시환경에서 스레드가 다른 쓰레드에 의해 삭제된 노드에 대한 로컬 참조를 보유하는 경우, 처음 쓰레드가 삭제된 노드의 링크를 사용할 수 있으면 매우 유용하다.

1. 노드에 대한 로컬 참조가 있는 쓰레드는 해당 노드에 포함된 모든 링크에 대해 역참조가 가능하다.

이 논문의 알고리즘은 위의 모든 특성과 현대에서 일반적으로 사용가능한 원자 프리미티브만 사용한다.