참조 카운트 기법

: 잠재적으로 스레드로부터의 참조가 임의의 노드를 재할당하는 것을 차단(순환 참조를 생성하는 기능 때문에) 할 수 있다.

논문의 목적

: lock-free reference counting & 선형화 방식으로, 위험 포인터의 효율 참조 횟수의 강도를 결합

- 메모리 할당에 대한 임의의 체계와 호환

- 사용되는 원자 프리미티브를 사용하여 구현

- 지역뿐만 아니라 전역 참조의 안전을 보장

FAA (Fetch-and-Add) & CAS (Compare-And-Swap) 사용

: 공유 메모리 시스템은 동기화를 위해 단일 메모리 단어의 원자적인 읽기 및 쓰기 작업뿐만 아니라, 강한 원자적 프리미티브를 지원

노드 : 각각 임의의 데이터를 포함하는 일련의 메모리 세그먼트로 구성 (데이터구조의 일부)

- 임의 패턴으로 서로 참조하여 상호 연결

링크 : 각 노드는 다른 노드를 참조하는 포인터

역참조 : 링크를 통해 참조된 노드를 따르는 작업

시스템의 메모리 양이 제한되어 있으므로, 이러한 노드의 점유된 메모리는 시스템에서 동적으로 할당 및 재생

동시 환경에서는 재사용된 노드의 메모리에 대한 접근 가능성이 데이터구조 및/또는 전체 시스템의 정확성에 치명적

garbage collector : 재사용을 올바르게 결정하는 논리 단위

- 향후에 접근이 불가능한 garbage만 회수

- 관련 스레드와 상호 작용하고 노드에 대한 접근에 제한

- lock-free이고 선형화

- 시스템의 기본 메모리 할당자와 호환

- 노드에 대한 로컬 참조가 있는 스레드는 해당 노드에 포함된 모든 링크를 역참조 가능

위험 포인터 (HP) 체계를 사용할 수 있는 reference counting 방법

: 위험 포인터는 4개의 데이터 구조에서 내부 링크의 안전을 보장하기 위해 로컬 참조 및 reference count의 안전을 보장하는 데 사용

- 각 스레드는 삭제되었지만 아직 재사용되지 않은 노드 목록을 유지 관리

- 목록의 길이가 특정 임계 값에 도달하면 재사용이 가능한지 검사

- 링크에서 삭제된 노드에 대한 참조수를 최소로 유지

노드

- mm\_ref: [int] - 해당 노드를 참조 접근 가능한 전역 링크 수

- mm\_trace: [bool] -

- mm\_del: [bool] - 노드가 삭제되었나

- link[index]: [pointer] - 활성화된 포인터

정리 : 삭제된 노드의 링크를 활성 노드에 대한 참조로 바꾸는 절차

- 삭제된 노드에 대한 로컬 참조가 있는 스레드는 반드시 적절한 활성 노드를 찾고 그래서 링크를 이용

모든 스레드의 모든 삭제 목록을 읽을 수 있습니다

모든 노드가 여전히 재사용되지 않으면, 이는 다른 스레드의 삭제목록에 있는 노드로 인한 것이므로, 스레드는 다른 모든 스레드의 삭제 목록도 정리하려고 합니다

DeRefLink

: 주어진 링크를 안전하게 참조 해제하고, 참조 해제된 노드에 대한 위험 포인터를 설정하여 리턴 된 노드에 접근할 수 있도록 향후 안전을 보장

ReleaseRef

: 주어진 노드에 더 이상 현재 스레드가 접근하지 않을 때

: 노드를 가리키는 이 스레드의 위험 포인터를 제거

- 노드가 삭제되고, reference count가 0이고 다른 위험 포인터가 이를 가리 키지 않으면, Scan함수에 의해 노드를 회수

(노드에서 DeleteNode라고하는 스레드에 의해 호출됨)

CompareAndSwapRef

: 링크에 대한 동시 업데이트

: 링크에서 공통 CAS를 수행하고 그에 따라 각 노드의 참조 횟수를 적절히 업데이트

- return 업데이트 성공여부 결과

스레드에 링크가 포함된 노드에 대한 위험 포인터 참조가 있는 경우,

함수는 링크에서 DeRefLink를 호출하는 모든 스레드가 안전하게 그렇게 할 수 있는지 확인합니다.

요구 사항은 CompareAndSwapRef의 호출 스레드에 지정된 노드에 대한 위험 포인터가 있어야 한다는 것입니다.

StoreRef

: 동시 업데이트가 없는 링크를 업데이트

스레드에 링크가 포함된 노드에 대한 위험 포인터 참조가 있는 경우,

함수는 링크에서 DeRefLink를 호출하는 스레드가 안전하게 그렇게 할 수 있는지 확인합니다.

요구 사항은 StoreRef의 호출 스레드에 지정된 노드에 대한 위험 포인터가 있어야 하며

다른 스레드가 링크에 동시에 쓰지 않아야 한다는 것입니다

(그렇지 않으면 CompareAndSwapRef가 대신 호출되어야 함).

NewNode

: 새로운 노드를 할당하고, 미래의 접근 안전을 보장하기 위해 자유 위험 포인터를 설정해 리턴

DeleteNode

: 노드가 데이터 구조에서 제거되고 재사용 할 수 있는 메모리가 필요할 때 호출

- 데이터 구조의 활성 노드에서 삭제된 노드에 대한 모든 참조를 제거

- 안전하게 삭제할 때까지 삭제된 노드를 delete하지 않음

TerminateNode

: 지정된 노드의 링크 중 다른 노드에 대한 소유권이 없는지 확인

- 다른 노드 또는 스레드에서 해당 노드로 소유권이 없는 경우 삭제된 노드에서 호출

CleanUpNode

: 주어진 노드의 링크에서 요구된 모든 참조가 활성 노드만 가리키게 하여 임의의 수의 삭제된 노드를 통한 중복 통로를 제거

Scan

: 이 스레드에 의해 삭제된 아직 재사용되지 않은 모든 노드를 검색하고 일치하는 위험 포인터가 없고 노드 내부의 링크에서 계산된 참조가 없는 노드만 재사용

CleanUpLocal

: 이 스레드에 의해 삭제된 노드의 링크에서 중복된 소유권 참조를 제거하려고 시도

CleanUpAll

: 모든 스레드에 의해 삭제된 노드의 링크에서 중복된 소유권 참조를 제거하려고 시도