배포. 계산. (2002) 15 : 255–271
DOI (Digital Object Identifier) 10.1007 / s00446-002-0079-z

잠금없는 기준 카운팅

David L. Detlefs, Paul A. Martin, Mark Moir, Guy L. Steele Jr.

Sun Microsystem Laboratories, 1 Network Drive, Burlington, MA 01803, USA (이메일 : Mark.Moir@sun.com)

접수 : 2002 년 1 월 / 접수 : 2002 년 3 월

추상. 가비지 콜렉션이 있다고 가정
동적 크기의 구현을보다 쉽게 디자인 할 수 있습니다
동시 데이터 구조. 그러나이 가정은 한계
그들의 적용 성. 우리는 의미있는 방법론을 제시합니다.
icant 급 데이터 구조로 설계자들은 먼저
가비지 수집에 따라 디자인하기 쉬운 문제
구현 한 다음 방법론을 적용하여 달성
가비지 수집과 무관 한 것. 우리의 방법론은
달성하기가 훨씬 어렵고 일반적으로
값 비싼 구현. 또한 잠금 해제 가능성이 높습니다.
잘 설계된 시스템에서 실제로는 충분합니다.
경합이 낮아야합니다. 이 논문에서 우리는
잠금없는 데이터 구조, 특히 *동적 설계*
크기가 큰 것, 즉 크기가 커질 수있는 데이터 구조
시간이지나면서 줄어 듭니다.
가비지 수집 (GC)이 단순화 될 수 있음은 잘 알려져 있습니다.

잘 알려진 참조 카운팅 기술을 기반으로
이중 비교 및 스왑 작업을 사용합니다.

키워드 : 잠금없는 동기화 – 참조 횟수
– 메모리 관리 – 동적 데이터 구조

1. 소개

우리는 단순화하는 데 사용할 수있는 방법론을 제시
동적 크기의 비 차단 공유 데이터 구조 설계
많은 연구 관심이 있었다
최근 몇 년 동안 비 차단 데이터의 구현에 지불
구조는 주로 잠금 및 대기없는 im-
주름. 그것이 보장된다면 구현은 *대기가 없다*
한정된 수의 단계를 수행 한 후에는
스레드의 타이밍 동작에 관계없이 완료
다른 작업을 동시에 실행). 구현
이다 *잠금이없는* 이 보장 경우 유한 한 단계 후
모든 작업 중 *일부* 작업이 완료됩니다. 두 속성
동기화를 위해 상호 배타적 인 잠금 장치 사용 금지
잠금을 잡고있는 동안 실이 늦어지면
잠금이 필요한 다른 작업은 완료 할 수 없습니다.
잠금 홀더가 다시 장착 될 때까지 몇 단계를 거쳐도
자물쇠를 임대합니다. 잠금없는 프로그래밍은 점점 더 중요 해지고 있습니다
이 문제와 관련된 다른 문제를 극복하기 위해
성능 병목 현상, 감수성을 포함한 잠금
지연 및 실패, 설계 복잡성 및 실시간
시스템, 우선 순위 반전.

잠금-자유는 보증인이기 때문에 대기-자유보다 약합니다.
대기 시간이없는 동안 시스템 전체에서만 진행됩니다.
dom은 작업 별 진행을 보장합니다. 연구 경험
그러나 더 강력한 대기 자유 속성은
동적 크기의 순차적 구현의 설계
데이터 구조. 또한 스토리지 제공 외에도
GC는 경영상의 이점을 통해
동시 데이터 설계시 심각한 *동기화* 문제
구조 [3,15,22]. 그러나 불행히도 단순히 디자인
가비지 수집 환경에 대한 구현은
전체 솔루션이 아닙니다. 첫째, 모든 프로그래밍 환경이 아닙니다
ments는 GC를 지원합니다. 둘째, 고용하는 거의 모든 사람들이
잠금 및 / 또는 중지와 같은 과도한 동기화
세계 메커니즘, 그들의 확장에 의문을 제기
ity. 마지막으로, 우리의 특별한 관심의 명백한 제한
그룹은 GC 종속 구현을 사용할 수 없다는 것입니다
가비지 수집기 자체 구현에서!

따라서이 작업의 목표는 프로그래머가
잠금없는 데이터를 설계 할 때 GC의 장점을 활용하십시오.
단점을 피하면서 구조 구현. 에
이를 위해 프로그래머가 사용할 수있는 방법론을 제공합니다
먼저 GC 종속 디자인의 쉬운 문제를 해결하기 위해
구현 한 다음 방법론을 순서대로 적용
GC 독립적 인 것을 달성하기 위해.

우리는 잠금을 유지하기 위해 방법론을 설계했습니다.
돔. 즉, GC 종속 구현이 잠금 인 경우
무료도 마찬가지입니다.
방법론. 여기에 두 가지 설명이 필요합니다. 먼저, 일부
독자들은 잠금이없는
대부분의 가비지 콜을 고려할 때 GC 종속 구현
장사는 스레드가 수행하는 동안 스레드 실행을 중지합니다.
작업. 그러나 이것이 *데이터 구조를* 의미하는 것은 아닙니다.
구현 자체는 잠금이 없습니다. lock-의 정의
시스템의 자유는 진보를 요구하지 않습니다 (예 : GC
또는 OS)는 스레드가 단계를 수행하지 못하게합니다.

둘째, 잠금이없고 GC 독립적 인
적용하여 동적 크기의 데이터 구조 구현
우리의 방법론, 그러나 우리는 객체가 어떻게 창조되는지를 명시하지 않았다.

먹고 파괴; 일반적으로 malloc과 free는 잠금이 없으며
따라서이를 기반으로 한 구현도 잠금이 없습니다. 방법-
대부분의 프로덕션 품질 메모리 할당자는
잠금에 대한 경합을 피하십시오 – 예를 들어
각 스레드에 대한 할당 버퍼를 할당하십시오.
잠금과 관련된 대부분의 문제. 실제로, 우리의 공동
리그 주사위와 Garthwaite는 최근 새로운 제안
"멀티 프로세서-
지역성을 개선하고 동기화를 줄이기 위해
오버 헤드. 결과적으로 할당자는 잠금에만 의존합니다.
매우 가끔 확장되므로 확장 성이 매우 뛰어납니다. 그들의 동안
할당자는 여전히 잠금을 사용하고 간단하고 유사합니다.
잘 알려진 비 차단 기술을 적용하여
완전히 비 차단입니다.

방법론의 중요한 특징 – 데이터 구조
이를 기반으로하는 구현은 – 그것이
성장과 축소를위한 시행의 ory 소비
시간이 지남에 따라
메모리 할당 메커니즘. 반대로 잠금이없는 이미지
동적 데이터 구조의 보완은 종종
저장 공간을 확보 할 수없는 특수"프리리스트"유지
일반적으로 다른 목적으로 재사용되거나 (예 : [19,25])
타입 안정 메모리와 같은 특별한 시스템 지원 [7].

우리의 방법론은 잘 알려진 쓰레기를 기반으로합니다.
*참조 카운팅*의 수집 기술 . 우리는 우리를 참조
LFRC (Lock-Free Reference Counting) 방법론. 에서
이 방법을 사용하면 각 개체의 개수에
그것을 가리키는 포인터이며,이 경우에만 해제 할 수 있습니다
카운트는 0입니다. 메모리에 대한 참조 카운팅 접근법
관리는 사이클을 감지하지 못한다는 한계가 있습니다.
쓰레기 (각각 도달 할 수없는 물건의 고리를 상상
다음에 대한 포인터가 있습니다. 모든 참조 카운트는 적어도 하나입니다.
따라서 어떤 객체도 쓰레기로 식별되지 않습니다). LFRC
이 제한을 상속합니다. 대부분의 경우 두 사이클 중 어느 것이나
가비지에 빠지거나 구현을 쉽게 수정할 수 있습니다.
이 가능성을 없애기 위해. 다른 경우에는 더 어렵다
또는 심지어는 불가능하다. 그런 경우에 우리의 방법론
모든 쓰레기가 재생 될 것이라고 보장 할 수는 없습니다.

정확한 참조 카운트를 유지하기 위해
원자 적으로 객체에 대한 포인터를 만들고
해당 객체의 참조 카운트를 증가시키고 원자 적으로
객체에 대한 포인터를 만들고 참조 횟수를 줄입니다.
이렇게하면 참조 할 때만 개체를 해제하여
따라서 카운트가 0이되면 객체가 아닌지 확인할 수 있습니다.
조기에 해방되었지만 결국에는 해방됩니다.
객체에 대한 포인터는 남아 있습니다.

위의 ap-에서 발생하는 주요 어려움
proach는 두 개의 개별 메모리를 원자 적으로 수정해야 할 필요성
위치 : 포인터와 대상의 참조 횟수
그것은 지적했다. 이것은 하드웨어를 사용하여 달성 할 수 있습니다
이 원 자성을 강화하기위한 동기화 프리미티브 또는
외관을 달성하기 위해 잠금없는 소프트웨어 메커니즘 사용
원자 성의. 후자의 접근법에 대한 연구는 다양한 결과를 낳았습니다.
다중 변수 syn-의 잠금 및 대기없는 구현
동기화 작업 (예 : [20,23]). 그러나이 모든 것은
결과가 복잡하고 상당한 오버 헤드가 발생하거나
정적으로 할당 된 메모리의 위치에서만 작동합니다.
동적 크기 지원 목표에 부적합
비 차단 데이터 구조. 따라서이 논문에서 우리는
전자의 접근을 간절히 바란다. 특히, 우리는
DCAS (double compare-and-swap) 명령어 기능
독립적으로 선택된 두 개의 메모리에 원자 적으로 액세스 할 수 있습니다.
양이온. (DCAS는 2 절에 정확하게 정의되어 있습니다.)
명령이 널리 사용 가능하지는 않습니다.
과거의 하드웨어 (예 : [21]). 또한 중요하다
하드웨어 동기화 메커니즘의 역할을 조사하기 위해
확장 가능한 비 블로킹의 가능성을 결정하는 데 필요한 nism
동기화. 따라서 무엇이 될 수 있는지 연구하는 것이 적절합니다
대체 하드웨어 메커니즘으로 달성됩니다.

DCAS조차도 참조 카운트를 유지할 수 없습니다
항상 정확합니다. 예를 들어 공유 포인터가
메모리는 객체 *v*를 가리키고 포인터를
다른 객체 *w*를 가리키면 원자 적으로 수정해야합니다.
포인터, *w* 의 참조 횟수 증가 및 *v* 의 감소
참조 횟수. 그러나 약한 요구 사항은
기준 카운트에 충분하면이 요구 사항이 충족됩니다.
DCAS를 사용하여 달성 *할 수* 있습니다. 이 약화는
참조 카운트가 항상 필요한 것은 아님
정확한. 중요한 요구 사항은
객체에 대한 포인터의 수가 0이 아니므로 참조도 마찬가지입니다.
포인터의 수가 0이면 참조-
따라서 카운트는 결국 0이됩니다. 이 두 가지 요구 사항
객체가 프리마
확실히, 그리고없는 개체의 참조 횟수
포인터는 결국 0이되어 해제 될 수 있습니다. [1]

이러한 관찰은 스레드가 안전하다는 것을 의미합니다.
새로운 객체 *를* 만들기 *전에* 객체의 참조 카운트 *를* 증가
스레드가 결국 생성하는 한 포인터
포인터, 또는 참조 카운트를 감소시켜 보상
이전 증분 이것은 우리가 사용하는 접근법입니다
방법론.

참고 용으로 더 약한 특성
counts는 우리가 single-
CAS (location compare-and-swap) 명령어
참조 카운트를 유지하기 위해 광범위하게 사용 가능합니다. 그러나 언제
공유 메모리 위치에서 포인터를로드해야합니다.
로드 된 객체의 참조 카운트를 증가시킵니다.
가치 포인트. 이 참조 횟수에만 액세스 할 수있는 경우
단일 위치 CAS의 경우 객체가 손상 될 위험이 있습니다.
참조 카운트를 증가시키기 전에 해제됩니다.
참조 카운트를 증가시키는 후속 시도는
사용 가능한 메모리가 손상되어 실제로는
다른 목적을위한 고양이. DCAS를 사용하면
원자 적으로 포인터를 확인하면서 참조 횟수
이 개체에 여전히 존재합니다. 이 시나리오는 왜
참조 횟수를 유지하기 위해 CAS를 사용하는 순진한 접근
작동하지 않습니다. 공식적으로 불가능한 것은 아닙니다.
용기 사실, 우리의 후속 작업은
논란의 여지가있는 CAS 기반의 효과적인 접근법
비 차단 동적 크기에서 메모리 관리 처리
공유 데이터 구조 [8-10]. 이 개발은 더 많은 것을 추가합니다
연구에 기반한 연구는
널리 이용 가능하지 않은 프리미티브 프리미티브가 적합하고
중대한.

LFRC 방법론의 유용성을 설명하기 위해
GC 종속 알고리즘을 변환하는 데 어떻게 사용할 수 있는지 보여줍니다.
최근에 [3]에서 GC 독립적 인 것으로 출판되었다. 알
[3]에 제시된 산술은

[1] 정확히 말하면 쓰레기가 가능하다는 점을 지적해야합니다
스레드가 영구적으로 실패하는 경우 존재 및 해제되지 않음
말도 안돼