(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2024년06월10일
(11) 등록번호 10-2672579
(24) 등록일자 2024년05월31일

(51) 국제특허분류(Int. Cl.)
*G06F 3/06* (2006.01) *G06F 12/1009* (2016.01)
*G06F 12/1027* (2016.01) *G06F 16/9035* (2019.01)
*G06F 16/907* (2019.01) *G06F 9/4401* (2018.01)
*G06F 9/455* (2018.01) *G06F 9/50* (2018.01)
*G06F 9/52* (2018.01) *G06F 9/54* (2018.01)
(52) CPC특허분류
*G06F 3/064* (2013.01)
*G06F 3/0604* (2013.01)
(21) 출원번호 10-2022-7006728
(22) 출원일자(국제) 2020년10월01일
심사청구일자 2022년02월25일
(85) 번역문제출일자 2022년02월25일
(65) 공개번호 10-2022-0036981
(43) 공개일자 2022년03월23일
(86) 국제출원번호 PCT/IB2020/059188
(87) 국제공개번호 WO 2021/070016
국제공개일자 2021년04월15일
(30) 우선권주장
16/599,141 2019년10월11일 미국(US)
(56) 선행기술조사문헌
KR1020060066497 A
KR1020190020843 A

(73) 특허권자
**인터내셔널 비지네스 머신즈 코포레이션**
미국 10504 뉴욕주 아몬크 뉴오차드 로드
(72) 발명자
**잘로마, 제이미**
미국 텍사스 78758-3493, 오스틴, 버넷 로드 11400, 아이비엠 코포레이션
**로저스, 마크**
미국 텍사스 78758-3493, 오스틴, 버넷 로드 11400, 아이비엠 코포레이션
(74) 대리인
**허정훈**

전체 청구항 수 : 총 25 항 심사관 : 신현상

(54) 발명의 명칭 **가상 메모리 메타데이터 관리**

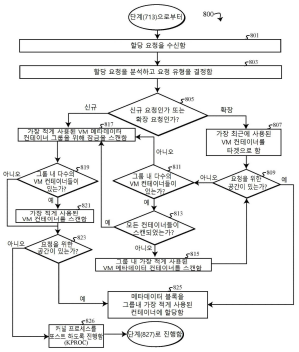
## (57) 요 약

가상 메모리 메타데이터 관리

확장 가능(scalable) 가상 메모리 메타데이터 관리는 컴퓨팅 시스템에 대해 이용 가능한 실제 물리적 메모리의 전체 양을 나타내는 복수의 사전-인스턴스화된(pre-instantiated) VM 메타데이터 컨테이너들을 포함하고, 이와

(뒷면에 계속)

*대 표 도* - 도8a



함께 추가 인스턴스화된 VM 메타데이터 컨테이너들은 필요에 따라 생성된다. VM 메타데이터 컨테이너들의 개별 및/또는 그룹들이 메타데이터 컨테이너 그룹들에 할당되며, 각 컨테이너 그룹은 상기 VM 메타데이터 컨테이너 그룹들에 할당된 획득된 잠금에 의해서 컨트롤된다. 가상 메모리 메타데이터는 "가장 적게 사용된" 기술을 사용하여 관리된다. 할당 요청들에 응답하여, 할당기(allocator)는 컨테이너 그룹들/VM 컨테이너들을 스캔하고 상기 가장 적게 사용된 컨테이너 그룹의 상기 가장 적게 사용된 VM 메타데이터 컨테이너에 대한 메모리 객체 메타데이터 할당을 충족시켜, 상기 개별 VM 메타데이터 컨테이너들 및/또는 상기 컨테이너 그룹들을 거의 동일 비율(rate)로 채운다.

(52) CPC특허분류
*G06F 3/061* (2013.01)
*G06F 3/0667* (2013.01)
*G06F 3/0671* (2013.01)
*G06F 3/0683* (2013.01)
*G06F 9/4401* (2013.01)
*G06F 9/45558* (2013.01)
*G06F 9/5016* (2013.01)
*G06F 9/5077* (2013.01)
*G06F 9/52* (2013.01)

# *명 세 서*

## *청구범위*

### 청구항 1

컴퓨터-구현(computer-implemented) 방법에 있어서, 상기 방법은:

복수의 가상 메모리(VM) 메타데이터 컨테이너들을, 프로세서에 의해서, 인스턴스화하는 단계(instantiating) - VM 메타데이터 컨테이너들의 총수는 컴퓨터 시스템에 제공된 총 실제 물리적 메모리를 가상 메모리로서 표현하기에 충분한 VM 메타데이터 블록들을 포함함 -;

상기 복수의 VM 메타데이터 컨테이너들을 둘 또는 그 이상의 컨테이너 그룹들로, 상기 프로세서에 의해서, 그룹화하는 단계(grouping) - 상기 둘 또는 그 이상의 컨테이너 그룹들 각각은 서로 다른 잠금(different lock)에 의해서 컨트롤됨 -;

메모리 객체 메타데이터를 VM 메타데이터 블록에 할당하는 요청에 응답하여, 가장 적게 사용된 컨테이너 그룹인 제 1의 컨테이너 그룹을 위해 상기 둘 또는 그 이상의 컨테이너 그룹들을, 상기 프로세서에 의해서, 스캔하는 단계(scanning);

상기 제 1의 컨테이너 그룹 내에서 가장 적게 사용된 VM 메타데이터 컨테이너인 제 1의 VM 메타데이터 컨테이너를 위해 상기 제 1의 컨테이너 그룹을, 상기 프로세서에 의해서, 쿼리하는 단계(querying);

상기 메모리 객체 메타데이터를 상기 제 1의 VM 메타데이터 컨테이너의 VM 메타데이터 블록에, 상기 프로세서에 의해서, 할당하는 단계(allocating) - 상기 할당하는 단계에 의하여 상기 메모리 객체 메타데이터를 할당하는 요청을 충족함-를 포함하는

컴퓨터-구현 방법.

### 청구항 2

제 1항에 있어서, 상기 복수의 VM 메타데이터 컨테이너들을 인스턴스화하는 단계는 상기 컴퓨터 시스템을 부팅(boot-up)하는 동안 수행되고 상기 컴퓨터 시스템의 운영 체제에 의해서 자동으로 관리되는

컴퓨터-구현 방법.

### 청구항 3

제 1항에 있어서, 상기 방법은:

메모리 객체 확장(grow) 요청을, 상기 프로세서에 의해서, 수신하는 단계 및 상기 메모리 객체와 연관된 메모리 객체 메타데이터에 할당된 VM 메타데이터 블록들의 수를 증가시키는 단계;

상기 메모리 객체 메타데이터를 저장하는 VM 메타데이터 블록들의 할당 확장 요청을 충족하기 위한(meet) VM 메타데이터 블록 요건들을, 상기 프로세서에 의해서, 계산하는 단계;

상기 제 1의 VM 메타데이터 컨테이너의 남은 공간의 양을 상기 VM 메타데이터 블록 요건들과, 상기 프로세서에 의해서, 비교하는 단계;

상기 제 1의 VM 메타데이터 컨테이너의 남은 공간의 양이 상기 VM 메타데이터 블록 요건들보다 크다고 상기 비교하는 단계가 결정하는 것에 응답하여, 상기 메모리 객체 메타데이터를 저장하기 위한 추가 VM 메타데이터 블록을 상기 제 1의 VM 메타데이터 컨테이너에, 상기 프로세서에 의해서, 할당하는 단계- 이에 의해서 상기 메모리 객체의 확장 요청을 충족함-를 더 포함하는

컴퓨터-구현 방법.

### 청구항 4

제 1항에 있어서, 상기 방법은:

상기 메모리 객체 메타데이터와 연관된 메모리 객체 확장 요청을, 상기 프로세서에 의해서, 수신하는 단계, 및 상기 메모리 객체에 할당된 VM 메타데이터 블록들의 수를 증가시키는 단계;

상기 메모리 객체 확장 요청을 충족하기 위한 VM 메타데이터 블록 요건들을, 상기 프로세서에 의해서, 계산하는 단계;

상기 제 1의 VM 메타데이터 컨테이너의 남은 공간의 양이 상기 메모리 객체 확장요청을 충족하기 위한 공간 요건들보다 큰지를 결정하기 위해 상기 제 1의 VM 메타데이터 컨테이너를 스캔하는 단계;

상기 제 1의 VM 메타데이터 컨테이너의 남은 공간의 양이 상기 메모리 객체 확장 요청을 충족하기 위한 공간 요건들보다 크지 않다고 결정하는 것에 응답하여, 상기 메모리 객체 확장 요청을 만족시키기에 충분한 남은 공간을 갖는 제 2의 VM 메타데이터 컨테이너를 위해 상기 제 1의 컨테이너 그룹 내에서, 상기 프로세서에 의해서, 스캔하는 단계를 더 포함하는

컴퓨터-구현 방법.

**청구항 5**

제 4항에 있어서, 상기 방법은:

상기 메모리 객체 확장 요청을 만족시키기에 충분한 공간을 갖는 상기 제 2의 VM 메타데이터 컨테이너가 상기 제 1의 컨테이너 그룹 내에서 발견될 수 없다고 결정하는 것에 응답하여, 제 2의 컨테이너 그룹 내에서 상기 메모리 객체 확장 요청을 만족시키기에 충분한 남은 공간을 갖는 상기 제 2의 VM 메타데이터 컨테이너를 위해, 상기 프로세서에 의해서, 스캔하는 단계; 및

상기 메모리 객체 메타데이터를 저장하기 위해 상기 제 2의 VM 메타데이터 컨테이너의 VM 메타데이터 블록을, 상기 프로세서에 의해서, 할당하는 단계를 더 포함하는

컴퓨터-구현 방법.

**청구항 6**

컴퓨터-구현 방법에 있어서, 상기 방법은:

하나 또는 그 이상의 VM 메타데이터 컨테이너들을 포함하는 복수의 컨테이너 그룹들로 구성된 메타데이터 블록 할당에 관해 메모리 관리 작업(operation) 수행요청을, 프로세서에 의해서, 수신하는 단계 - 상기 복수의 컨테이너 그룹들의 개별(individual) 컨테이너 그룹들에 대한 액세스는 서로 다른 잠금들(locks)에 의해 컨트롤됨-;

제 1의 컨테이너 그룹을 컨트롤하는 제 1의 잠금을, 상기 프로세서에 의해서, 획득하는 단계(acquiring);

상기 메타데이터 블록 할당의 일부인 상기 제 1의 컨테이너 그룹의 VM 메타데이터 블록들에 의해서 저장된 메모리 객체 메타데이터에 관해 모든 메모리 관리 작업들을, 상기 프로세서에 의해서, 수행하는 단계;

상기 제 1의 잠금을, 상기 프로세서에 의해서, 해제하는 단계(releasing); 및

제 2의 컨테이너 그룹을 컨트롤하는 제 2의 잠금을, 상기 프로세서에 의해서, 획득하는 단계를 포함하는

컴퓨터-구현 방법.

**청구항 7**

제 6항에 있어서, 상기 메타데이터 블록 할당의 일부인 상기 제 1의 컨테이너 그룹 내의 상기 메모리 객체 메타데이터에 관해 모든 메모리 관리 작업들을 수행하는 단계는:

상기 메타데이터 블록 할당의 일부인 상기 제 1의 컨테이너 그룹에 의해서 저장된 모든 VM 메타데이터 블록들을, 상기 프로세서에 의해서, 동시에 해방하는 단계(freeing)를 포함하는

컴퓨터 구현 방법.

**청구항 8**

컴퓨터 구현 방법에 있어서, 상기 방법은:

복수의 가상 메모리(VM) 메타데이터 컨테이너들을, 프로세서에 의해서, 인스턴스화하는 단계(instantiating) -VM 메타데이터 컨테이너들의 총수는 컴퓨터 시스템에 제공된 총 실제 물리적 메모리를 가상 메모리로서 표현하기에 충분한 VM 메타데이터 블록들을 포함함-;

상기 복수의 VM 메타데이터 컨테이너들을 둘 또는 그 이상의 컨테이너 그룹들로, 상기 프로세서에 의해서, 그룹화하는 단계(grouping)- 상기 둘 또는 그 이상의 컨테이너 그룹들의 각각은 서로 다른 잠금에 의해서 컨트롤됨-;

메모리 객체를 확장하는 요청(a request to grow)과 상기 메모리 객체의 메모리 객체 메타데이터를 저장하는 데 할당된 VM 메타데이터 블록들의 수를 증가하는 요청을, 상기 프로세서에 의해서, 수신하는 단계(receiving);

상기 메모리 객체의 메모리 객체 메타데이터의 스토리지를 위해 적어도 하나의 VM 메타데이터 블록을 이전에 할당하는 것을 담당한(responsible for) 제 1의 컨테이너 그룹의 제 1의 VM 메타데이터 컨테이너를, 상기 프로세서에 의해서, 결정하는 단계(determining);

상기 확장하는 요청을 충족하기 위해 메타데이터 블록 요건들을, 상기 프로세서에 의해서, 계산하는 단계(calculating);

상기 확장하는 요청을 충족하기 위해 상기 메타데이터 블록 요건들이 상기 제 1의 VM 메타데이터 컨테이너의 자유(free) VM 메타데이터 블록들의 크기보다 큰지를 결정하는 단계(determining)를 포함하는

컴퓨터 구현 방법.

**청구항 9**

제 8항에 있어서, 상기 복수의 VM 메타데이터 컨테이너를 인스턴스화하는 단계는 상기 컴퓨터 시스템을 부팅하는 동안 수행되고 상기 컴퓨터 시스템의 운영 체제에 의해서 자동으로 관리되는

컴퓨터 구현 방법.

**청구항 10**

제 8항에 있어서, 상기 방법은:

상기 확장하는 요청을 충족하기 위한 상기 메타데이터 블록 요건들이 상기 제 1의 VM 메타데이터 컨테이너의 자유 메타데이터 블록들의 크기보다 크지 않다고 결정하는 것에 응답하여, 상기 메모리 객체 메타데이터를 상기 제 1의 메타데이터 컨테이너의 자유 메타데이터 블록에, 상기 프로세서에 의해서, 할당하는 단계를 더 포함하는

컴퓨터 구현 방법.

**청구항 11**

제 8항에 있어서, 상기 방법은:

상기 확장하는 요청을 충족하기 위한 상기 메타데이터 블록 요건들이 상기 제 1의 VM 메타데이터 컨테이너의 자유 메타데이터 블록들의 크기보다 크다고 결정하는 것에 응답하여;

상기 제 1의 컨테이너 그룹 내의 가장 적게 사용된 VM 메타데이터 컨테이너를 위해 상기 제 1의 컨테이너 그룹을, 상기 프로세서에 의해서, 스캔하는 단계; 및

상기 메모리 객체 메타데이터를 상기 제 1의 컨테이너 그룹 내에서 가장 적게 사용된 VM 메타데이터 컨테이너의 자유 메타데이터 블록에, 상기 프로세서에 의해서, 할당하는 단계를 더 포함하는

컴퓨터 구현 방법.

**청구항 12**

제 8항에 있어서, 상기 방법은:

상기 확장하는 요청을 충족하기 위한 상기 메타데이터 블록 요건들이 상기 제 1의 메타데이터 컨테이너의 자유 메타데이터 블록들의 크기보다 크다고 결정하는 것에 응답하여;

제 2의 컨테이너 그룹 내의 가장 적게 사용된 VM 메타데이터 컨테이너를 위해 제 2의 컨테이너 그룹을, 상기 프

로세서에 의해서, 스캔하는 단계; 및

상기 메모리 객체 메타데이터를 상기 제 2의 컨테이너 그룹 내에서 가장 적게 사용된 VM 메타데이터 컨테이너의 자유 메타데이터 블록에, 상기 프로세서에 의해서, 할당하는 단계를 더 포함하는

컴퓨터 구현 방법.

**청구항 13**

제 8항에 있어서, 상기 방법은:

상기 확장하는 요청을 충족하기 위한 상기 메타데이터 블록 요건들이 상기 제 1의 메타데이터 컨테이너의 자유 메타데이터 블록들의 크기보다 크다고 결정하는 것에 응답하여;

가장 적게 사용된 컨테이너 그룹을 위해 상기 둘 또는 그 이상의 컨테이너 그룹들을, 상기 프로세서에 의해서, 스캔하는 단계;

가장 적게 사용된 VM 메타데이터 컨테이너를 위해 상기 가장 적게 사용된 컨테이너 그룹을, 상기 프로세서에 의해서, 추가로 스캔하는 단계;

상기 확장하는 요청을 충족하기 위한 상기 메타데이터 블록 요건들이 상기 가장 적게 사용된 VM 메타데이터 컨테이너의 자유 메타데이터 블록들의 크기보다 크다고, 상기 프로세서에 의해서, 결정하는 단계;

상기 확장하는 요청을 충족하기에 적어도 충분한 자유 VM 메타데이터 블록들을 포함하는 하나 또는 그 이상의 신규 VM 메타데이터 컨테이너들을 생성하기 위해 커널 프로세스(kernel process)를 포스트하는 단계(posting)를 더 포함하는

컴퓨터 구현 방법.

**청구항 14**

컴퓨터 시스템에 있어서, 상기 시스템은:

프로세서; 및

상기 프로세서에 결합된 컴퓨터-판독 가능(computer-readable) 스토리지 매체를 포함하고, 상기 컴퓨터-판독 가능 스토리지 매체는 컴퓨터-구현 방법을 실행하는 프로그램 명령들을 포함하며, 상기 방법은;

복수의 가상 메모리(VM) 메타데이터 컨테이너들을, 상기 프로세서에 의해서, 인스턴스화하는 단계(instantiating) - VM 메타데이터 컨테이너들의 총수는 컴퓨터 시스템에 제공된 총 실제 물리적 메모리를 가상 메모리로서 표현하기에 충분한 VM 메타데이터 블록들을 포함함 -;

상기 복수의 VM 메타데이터 컨테이너들을 둘 또는 그 이상의 컨테이너 그룹들로, 상기 프로세서에 의해서, 그룹화하는 단계(grouping) - 상기 둘 또는 그 이상의 컨테이너 그룹들 각각은 서로 다른 잠금(different lock)에 의해서 컨트롤됨 -;

메모리 객체 메타데이터를 VM 메타데이터 블록에 할당하는 요청에 응답하여, 가장 적게 사용된 컨테이너 그룹을 위해 상기 둘 또는 그 이상의 컨테이너 그룹들을, 상기 프로세서에 의해서, 스캔하는 단계(scanning); 및

상기 가장 적게 사용된 컨테이너 그룹 내에서 가장 적게 사용된 VM 메타데이터 컨테이너를 위해 상기 가장 적게 사용된 컨테이너 그룹을, 상기 프로세서에 의해서, 쿼리하는 단계(querying)를 포함하는

컴퓨터 시스템.

**청구항 15**

제 14항에 있어서, 상기 방법은:

상기 메모리 객체 메타데이터를 상기 가장 적게 사용된 VM 메타데이터 컨테이너의 VM 메타데이터 블록에, 상기 프로세서에 의해서, 할당하는 단계(allocating) - 상기 할당하는 단계에 의하여 상기 메모리 객체 메타데이터를 할당하는 요청을 충족함-를 더 포함하는

컴퓨터 시스템.

**청구항 16**

제 14항에 있어서, 상기 방법은:

상기 가장 적게 사용된 VM 메타데이터 컨테이너가 상기 메모리 객체 메타데이터를 저장하는 데 필요한 양의 VM 메타데이터 블록들을 갖고 있지 않다고(lack), 상기 프로세서에 의해서, 결정하는 단계를 더 포함하는

컴퓨터 시스템.

**청구항 17**

제 16항에 있어서, 상기 방법은:

상기 메모리 객체 메타데이터의 할당 요청을 충족하기에 충분한 이용 가능 VM 메타데이터 블록들을 포함하는 하나 또는 그 이상의 신규 VM 메타데이터 컨테이너들을 생성하기 위해 커널 프로세스를, 상기 프로세서에 의해서, 포스트하는 단계를 더 포함하는

컴퓨터 시스템.

**청구항 18**

제 17항에 있어서, 상기 방법은:

상기 컴퓨터 시스템의 상태를, 상기 프로세서에 의해서, 쿼리하는 단계;

상기 컴퓨터 시스템의 상태를, 상기 컴퓨터 시스템에 의한 사용을 위해 이용 가능한 컴퓨팅 자원들의 최소 양을 유지하기 위한 운영 체제의 시스템 관리 계획과, 상기 프로세서에 의해서, 비교하는 단계;

상기 컴퓨터 시스템의 상태를 상기 시스템 관리 계획과 비교하여, 이용 가능한 컴퓨팅 자원들의 최소양을 유지하는 것이 충족되지 않았다고 결정하는 것에 응답하여, 상기 하나 또는 그 이상의 신규 VM 메타데이터 컨테이너들을, 상기 프로세서에 의해서, 인스턴스화하는 단계를 더 포함하는

컴퓨터 시스템.

**청구항 19**

제 17항에 있어서, 상기 방법은:

상기 컴퓨터 시스템의 상태를, 상기 프로세서에 의해서, 쿼리하는 단계;

상기 컴퓨터 시스템에 의한 사용을 위해 이용 가능한 컴퓨팅 자원들의 최소 양을 유지하기 위해 상기 컴퓨터 시스템의 상태를 운영 체제의 시스템 관리 계획과, 상기 프로세서에 의해서, 비교하는 단계; 및

상기 컴퓨터 시스템의 상태를 상기 시스템 관리 계획과 비교하여, 상기 이용 가능한 컴퓨팅 자원들의 최소 양이 충족되었다고 결정하는 것에 응답하여, 상기 하나 또는 그 이상의 신규 VM 메타데이터 컨테이너들을 생성하기 위한 상기 커널 프로세스를, 상기 프로세서에 의해서, 취소하는 단계(cancelling)를 더 포함하는

컴퓨터 시스템.

**청구항 20**

제 15항에 있어서, 상기 방법은:

상기 메모리 객체 메타데이터와 연관된 메모리 객체의 확장 및 상기 메모리 객체에 할당된 VM 메타데이터 블록들의 수의 증가 요청을, 상기 프로세서에 의해서, 수신하는 단계;

상기 메모리 객체의 확장 요청을 충족하기 위해 VM 메타데이터 블록 요건들을, 상기 프로세서에 의해서, 계산하는 단계;

상기 가장 적게 사용된 VM 메타데이터 컨테이너의 남은 공간의 양이 상기 메모리 객체 확장 요청을 충족하기 위한 공간 요건들보다 큰지를 결정하기 위해 상기 가장 적게 사용된 VM 메타데이터 컨테이너를, 상기 프로세서에 의해서, 스캔하는 단계;

상기 가장 적게 사용된 VM 메타데이터 컨테이너의 남은 공간의 양이 상기 메모리 객체 확장 요청을 충족하기 위

한 상기 공간 요건들보다 크지 않다고 결정하는 것에 응답하여, 상기 메모리 객체 확장 요청을 만족시키기에 충분한 공간을 갖는 제 2의 VM 메타데이터 컨테이너를, 상기 프로세서에 의해서, 식별하는 단계 - 상기 제 2의 VM 메타데이터 컨테이너는 상기 가장 적게 사용된 컨테이너 그룹 내에서 발견될 수 없음-; 및

상기 메모리 객체 메타데이터를 저장하기 위해 제 2의 컨테이너 그룹 내에서 상주하는 제 2의 VM 메타데이터 컨테이너의 VM 메타데이터 블록들을, 상기 프로세서에 의해서, 할당하는 단계를 더 포함하는

컴퓨터 시스템.

**청구항 21**

컴퓨터-판독가능 프로그램 명령들을 갖는 하나 또는 그 이상의 컴퓨터-판독가능 스토리지 매체에 있어서, 상기 프로그램 명령들은 상기 하나 또는 그 이상의 컴퓨터-판독가능 스토리지 매체상에 저장되며, 컴퓨터-구현 방법을 실행하고, 상기 방법은:

복수의 가상 메모리(VM) 메타데이터 컨테이너들을 인스턴스화하는 단계(instantiating) - VM 메타데이터 컨테이너들의 총수는 컴퓨터 시스템에 제공된 총 실제 물리적 메모리를 가상 메모리로서 표현하기에 충분한 VM 메타데이터 블록들을 포함함 -;

상기 복수의 VM 메타데이터 컨테이너들을 둘 또는 그 이상의 컨테이너 그룹들로 그룹화하는 단계(grouping);

메모리 객체 메타데이터를 VM 메타데이터 블록에 할당하는 요청에 응답하여, 가장 적게 사용된 컨테이너 그룹을 위해 상기 둘 또는 그 이상의 컨테이너 그룹들을 스캔하는 단계(scanning); 및

상기 가장 적게 사용된 컨테이너 그룹 내에서 가장 적게 사용된 VM 메타데이터 컨테이너를 위해 상기 가장 적게 사용된 컨테이너 그룹을 쿼리하는 단계(querying)를 포함하는

컴퓨터-판독가능 스토리지 매체.

**청구항 22**

제 21항에 있어서, 상기 방법은:

상기 메모리 객체 메타데이터를 상기 가장 적게 사용된 VM 메타데이터 컨테이너의 VM 메타데이터 블록에 할당하는 단계(allocating) - 상기 할당하는 단계에 의하여 상기 메모리 객체 메타데이터를 할당하는 요청을 충족함-를 더 포함하는

컴퓨터-판독가능 스토리지 매체.

**청구항 23**

제 21항에 있어서, 상기 방법은:

상기 가장 적게 사용된 VM 메타데이터 컨테이너는 상기 메모리 객체 메타데이터를 저장하는 데 필요한 양의 VM 메타데이터 블록들을 갖고 있지 않다(lack)고 결정하는 단계를 더 포함하는

컴퓨터-판독가능 스토리지 매체.

**청구항 24**

제 23항에 있어서, 상기 방법은:

상기 메모리 객체 메타데이터 할당 요청을 충족하기에 충분한 이용 가능 VM 메타데이터 블록들을 포함하는 하나 또는 그 이상의 신규 VM 메타데이터 컨테이너들을 생성하기 위해 커널 프로세스를, 포스트하는 단계를 더 포함하는

컴퓨터-판독가능 스토리지 매체.

**청구항 25**

제 24항에 있어서, 상기 방법은:

상기 컴퓨터 시스템의 상태를 쿼리하는 단계;

상기 컴퓨터 시스템에 의한 사용을 위해 이용 가능한 컴퓨팅 자원들의 최소 양을 유지하기 위해 상기 컴퓨터 시스템의 상태를 운영 체제의 시스템 관리 계획과, 프로세서에 의해서, 비교하는 단계;

상기 컴퓨터 시스템의 상태를 운영 체제의 시스템 관리 계획과 비교하여, 이용 가능한 컴퓨팅 자원들의 최소 양을 유지하는 것이 충족되지 않았다고 결정하는 것에 응답하여, 상기 하나 또는 그 이상의 신규 VM 메타데이터 컨테이너들을 인스턴스화하는 단계를 더 포함하는

컴퓨터-판독가능 스토리지 매체.

## 발명의 설명

### 기 술 분 야

[0001] [0001] 본 발명은 일반적으로 가상 메모리의 분야에 관한 것이고, 더욱 구체적으로는 가상 메모리 메타데이터의 관리에 관한 것이다.

### 배 경 기 술

[0002] [0002] 가상 메모리(VM)는 운영 체제들에 의해서 제공되는 기능이다. 운영 체제는 가상 메모리 공간을 생성하고, 애플리케이션들과 프로그램들은 마치 그 공간이 인접한 물리적 메모리 디바이스의 하나의 조각인 것처럼 액세스할 수 있다. 가상 메모리 공간은 물리적 메모리와 함께 협력하여 작동하는 디스크-기반(disk-based) 스토리지 자원들의 조합일 수 있다. 가상 메모리는 주소들의 대체 세트(alternate set)로서 기능하고 이용 가능 메모리의 범위를 확장한다. 운영 체제들은 사용할 가상 메모리를 위해 디스크 공간의 양을 할당함에 의해서 가상 메모리를 구성할 수 있다. 애플리케이션들과 프로그램들은 가상 메모리 공간 내에서 이들 가상 주소들을 데이터와 명령들을 저장하기 위해 사용하는데, 더 빈번하게 주소 지정될 수 있는 다른 데이터 세트들 또는 명령들만큼 자주 주소 지정할 필요가 없는 데이터와 명령들을 저장하기 위해 사용한다. 가상 메모리 주소 위치들이 운영 체제에 의해서 호출될 때, 가상 메모리에 의해서 저장된 데이터와 명령들은 물리적 메모리의 물리적 주소로 복사된다. 가상 메모리를 물리적 메모리에 복사하기 위해서, 운영 체제들은 가상 메모리를 여러 페이지들로 나눈다. 각 페이지는 디스크-기반 스토리지에 의해서 저장되는 고정된 수의 주소들을 포함하는데, 운영 체제가 그 주소를 호출할 때까지 포함한다. 호출되었을 때, 운영 체제는 매핑이라고 알려진 프로세스에서 가상 주소들을 실제 주소들로 변환한다. 디스크-기반 스토리지에서 메인 메모리로 가상 페이지들을 복사하는 것은 페이징 또는 스와핑으로 알려져 있다.

## 발명의 내용

### 해결하려는 과제

[0003] [0003] 메모리 할당은 요청 시 메모리의 블록들을 배정하는 프로세스이다. 할당기(allocator)는 소수의 큰 블록들로 운영 체제로부터 메모리를 받고, 이들을 더 작은 블록들로 나누어서 요청들을 만족시킨다. 할당기는 미래 할당 요청을 만족시키기 위해서 모든 반환된 블록들을 운영 체제에 대해서 사용하기 위해 이용 가능하게 한다.

### 과제의 해결 수단

[0004] [0004] 본 발명의 실시 예들은 확장할 수 있는 가상 메모리(scalable virtual memory)의 관리를 위해서 컴퓨터-구현(computer-implemented) 방법, 연관 컴퓨터 시스템 및 컴퓨터 프로그램 제품과 관련이 있다. 컴퓨터-구현 방법은: 복수의 가상 메모리(VM) 메타데이터 컨테이너들을, 프로세서에 의해서, 인스턴스화하는 단계(instantiating) - VM 메타데이터 컨테이너들의 총수는 컴퓨터 시스템에 제공된 총 실제 물리적 메모리를 가상 메모리로서 표현하기에 충분한 VM 메타데이터 블록들을 포함함 -; 상기 복수의 VM 메타데이터 컨테이너들을 둘 또는 그 이상의 컨테이너 그룹들로, 상기 프로세서에 의해서, 그룹화하는 단계(grouping) - 상기 둘 또는 그 이상의 컨테이너 그룹들 각각은 서로 다른 잠금(different lock)에 의해서 컨트롤됨 -; 메모리 객체 메타데이터를 VM 메타데이터 블록에 할당하는 요청에 응답하여, 가장 적게 사용된 컨테이너 그룹인 제 1의 컨테이너 그룹을 위해 상기 둘 또는 그 이상의 컨테이너 그룹들을, 상기 프로세서에 의해서, 스캔하는 단계(scanning); 상기 제 1의 컨테이너 그룹 내에서 가장 적게 사용된 VM 메타데이터 컨테이너인 제 1의 VM 메타데이터 컨테이너를 위해 상기 제 1의 컨테이너 그룹을, 상기 프로세서에 의해서, 쿼리하는 단계(querying); 상기 메모리 객체 메타데이터를 상기 제 1의 VM 메타데이터 컨테이너의 VM 메타데이터 블록에, 상기 프로세서에 의해서, 할당하는 단계

(allocating) - 상기 할당하는 단계에 의하여 상기 메모리 객체 메타데이터를 할당하는 요청을 충족함-를 포함한다.

[0005] [0005] 본 발명의 실시 예들은 확장할 수 있는 가상 메모리의 관리를 위해서 컴퓨터-구현 방법, 관련 컴퓨터 시스템 및 컴퓨터 프로그램 제품과 관련이 있다, 상기 컴퓨터-구현 방법은: 하나 또는 그 이상의 VM 메타데이터 컨테이너들을 포함하는 복수의 컨테이너 그룹들로 구성된 메타데이터 블록 할당에 관해 메모리 관리 작업(operation) 수행요청을, 프로세서에 의해서, 수신하는 단계 - 상기 복수의 컨테이너 그룹들의 개별(individual) 컨테이너 그룹들에 대한 액세스는 서로 다른 잠금들(locks)에 의해 컨트롤됨-; 제 1의 컨테이너 그룹을 컨트롤하는 제 1의 잠금을, 상기 프로세서에 의해서, 획득하는 단계(acquiring); 상기 메타데이터 블록 할당의 일부인 상기 제 1의 컨테이너 그룹의 VM 메타데이터 블록들에 의해서 저장된 메모리 객체 메타데이터에 관해 모든 메모리 관리 작업들을, 상기 프로세서에 의해서, 수행하는 단계; 상기 제 1의 잠금을, 상기 프로세서에 의해서, 해제하는 단계(releasing); 및 제 2의 컨테이너 그룹을 컨트롤하는 제 2의 잠금을, 상기 프로세서에 의해서, 획득하는 단계를 포함한다.

[0006] [0006] 본 발명의 실시 예들은 확장할 수 있는 가상 메모리의 관리를 위해서 컴퓨터-구현 방법, 연관 컴퓨터 시스템 및 컴퓨터 프로그램 제품과 관련이 있다, 상기 컴퓨터-구현 방법은: 복수의 가상 메모리(VM) 메타데이터 컨테이너들을, 프로세서에 의해서, 인스턴스화하는 단계(instantiating) - VM 메타데이터 컨테이너들의 총수는 컴퓨터 시스템에 제공된 총 물리적 RAM을 가상 메모리로서 표현하기에 충분한 VM 메타데이터 블록들을 포함함-; 상기 복수의 VM 메타데이터 컨테이너들을 둘 또는 그 이상의 컨테이너 그룹들로, 상기 프로세서에 의해서, 그룹화하는 단계(grouping)- 상기 둘 또는 그 이상의 컨테이너 그룹들의 각각은 서로 다른 잠금에 의해서 컨트롤됨-; 메모리 객체의 확장(grow)과 상기 메모리 객체의 메모리 객체 메타데이터를 저장하는 데 할당된 VM 메타데이터 블록들의 수의 증가(increase) 요청을, 상기 프로세서에 의해서, 수신하는 단계(receiving); 상기 메모리 객체의 메모리 객체 메타데이터의 스토리지를 위해 적어도 하나의 VM 메타데이터 블록을 이전에 할당하는 것을 담당한(responsible for) 제 1의 컨테이너 그룹의 제 1의 VM 메타데이터 컨테이너를, 상기 프로세서에 의해서, 결정하는 단계(determining); 상기 확장 요청을 충족하기 위해 메타데이터 블록 요건들을, 상기 프로세서에 의해서, 계산하는 단계(calculating); 상기 확장 요청을 충족하기 위해 상기 메타데이터 블록 요건들이 상기 제 1의 VM 메타데이터 컨테이너의 자유(free) VM 메타데이터 블록들의 크기보다 큰지를 결정하는 단계(determining)를 포함한다.

[0007] [0007] 본 발명의 실시 예들은 확장할 수 있는 가상 메모리의 관리를 위해서 컴퓨터-구현 방법, 관련 컴퓨터 시스템 및 컴퓨터 프로그램 제품과 관련이 있다, 상기 컴퓨터 시스템은: 프로세서; 및 상기 적어도 하나의 프로세서에 결합된 컴퓨터-판독 가능(computer-readable) 스토리지 매체를 포함하고, 상기 컴퓨터-판독 가능 스토리지 매체는 컴퓨터-구현 방법을 실행하는 프로그램 명령들을 포함하며, 상기 방법은; 복수의 가상 메모리(VM) 메타데이터 컨테이너들을, 상기 프로세서에 의해서, 인스턴스화하는 단계(instantiating) - VM 메타데이터 컨테이너들의 총수는 컴퓨터 시스템에 제공된 모든 실제 물리적 메모리를 가상 메모리로서 표현하기에 충분한 VM 메타데이터 블록들을 포함함 -; 상기 복수의 VM 메타데이터 컨테이너들을 둘 또는 그 이상의 컨테이너 그룹들로, 상기 프로세서에 의해서, 그룹화하는 단계(grouping); 메모리 객체 메타데이터를 VM 메타데이터 블록에 할당하는 요청에 응답하여, 가장 적게 사용된 컨테이너 그룹을 위해 상기 둘 또는 그 이상의 컨테이너 그룹들을, 상기 프로세서에 의해서, 스캔하는 단계(scanning); 및 상기 가장 적게 사용된 컨테이너 그룹 내에서 가장 적게 사용된 VM 메타데이터 컨테이너를 위해 상기 가장 적게 사용된 컨테이너 그룹을, 상기 프로세서에 의해서, 쿼리하는 단계(querying)를 포함한다.

## *도면의 간단한 설명*

[0008] [0008] 도 1은 데이터 처리 시스템의 내부 및 외부 컴포넌트들의 블록도의 일 실시 예이며, 여기에 기술된 실시 예들은 본 발명에 따라 구현될 수 있다.

[0009] [0009] 도 2a는 본 발명에 따라 메모리 객체 메타데이터를 할당하기 위한 컴퓨팅 환경의 일 실시 예에 대한 블록도를 도시한다.

[0010] [0010] 도 2b는 본 발명에 따라 메모리 객체 데이터를 할당하기 위한 컴퓨팅 환경의 대체 실시 예에 대한 블록도를 도시한다.

[0011] [0011] 도 3a는 가상 메모리에 저장된 데이터 또는 명령들을 저장하고, 검색하기 위한 컴퓨팅 환경의 실시 예들에 대한 블록도를 도시한다.

[0012] 도 3b는 논리적 캐쉬를 포함하는 컴퓨팅 환경인, 가상 메모리에 저장된 데이터 또는 명령들을 저장 및 검색하기 위한 컴퓨팅 환경의 대체 실시 예에 대한 블록도를 도시한다.

[0013] 도 3c는 물리적 캐쉬를 포함하는 컴퓨팅 환경인, 가상 메모리에 저장된 데이터 또는 명령들을 저장 및 검색하기 위한 컴퓨팅 환경의 또 다른 대체 실시 예에 대한 블록도를 도시한다.

[0014] 도 4a는 본 발명에 따라, 서로(one another) 병렬로 운영하는 사전-인스턴스화된(pre-instantiated) VM 메타데이터 컨테이너들을 구현하는 것의 일 실시 예를 도시하는 블록도를 예시한다.

[0015] 도 4b는 본 발명에 따라, 서로 병렬로 운영하는 VM 메타데이터 컨테이너들의 VM 메타데이터 블록들에 대한 메모리 객체 메타데이터의 할당들의 분산을 도시하는 블록도를 도시한다.

[0016] 도 5는 서로 다른 잠금들에 의해서 개별적으로 컨트롤되는 복수의 컨테이너 그룹들 사이에 확산한(spread across) 메모리 객체 메타데이터를 포함하는 메타데이터 블록 할당의 일 실시 예를 도시하고 컨테이너 그룹에 의해서 가상 메모리 객체 메타데이터의 관리를 우선 순위에 따라 처리하는 블록도를 도시한다.

[0017] 도 6은 본 발명에 따라 메모리 객체 메타데이터를 할당하기 위한 컴퓨팅 환경의 또 다른 대체 실시 예에 대한 블록도를 도시한다.

[0018] 도 7은 본 발명에 따라 VM 메타데이터 컨테이너들을 인스턴스화하는 할당기를 위한 알고리즘의 일 실시 예에 대해 설명한다.

[0019] 도 8a는 메모리 객체 메타데이터의 할당을 VM 메타데이터 컨테이너의 VM 메타데이터 블록들에 구현하는 알고리즘의 일 실시 예를 도시한다.

[0020] 도 8b는 메모리 객체 메타데이터의 할당을 위해서 추가적인 VM 메타데이터 컨테이너들을 생성하기 위해 포스트 커널 프로세스를 구현하는 알고리즘의 일 실시 예를 도시한다.

## *발명을 실시하기 위한 구체적인 내용*

[0009] [0021] 여기서 사용된 용어는 오직 특정 실시 예들만을 기술하는 목적을 위한 것이며, 본 발명을 제한하려고 의도된 것은 아니다. 여기서 사용된 바와 같이, 단수형들은 문맥이 달리 명확하게 나타내지 않는 한, 복수형들도 포함하도록 의도되었다. 본 명세서(specification)에서 사용할 때, "포함하다" 및/또는 "포함하는"이라는 용어들은, 정해진 특징들, 단계들, 작업들, 엘리먼트들, 및/또는 컴포넌트들의 존재를 명시하지만, 하나 또는 그 이상의 다른 특징들, 단계들, 작업들, 엘리먼트들, 컴포넌트들 및/또는 관련 그룹들의 존재 또는 추가를 배제하지 않는다는 점을 추가로 이해해야 한다.

[0010] [0022] 아래 청구항들에서 대응구조들, 재료들, 동작들, 및 모든 수단들 또는 단계 플러스 기능 엘리먼트들의 등가물들은 구체적으로 청구된 바와 같이 다른 청구된 엘리먼트들과 조합하여 기능을 수행하기 위한 모든 구조, 재료, 또는 동작을 포함하도록 의도되었다. 본 명세서의 설명은 예시와 설명의 목적을 위해서 제공된 것이며 개시된 형태의 설명이 전부라거나 이에 제한하려고 의도한 것은 아니다. 많은 수정들과 변형들이 본 발명의 범위로부터 벗어나지 않고 당업자들에게 명백할 것이다. 실시 예는 본 발명의 원리들 및 실제 적용을 가장 잘 설명하기 위해서 그리고 당업자들이 다양한 수정들을 갖는 다양한 실시 예들에 대한 설명을 고려된 특정용도에 적합하도록 이해할 수 있게 하기 위해 선택되었고 설명되었다.

[0011] 개요

[0012] [0023] 본 발명의 실시 예들은 가상 메모리 메타데이터 작업들(operations) 및 보다 구체적으로는, 확장할 수 있는 가상 메모리 관리를 위해서 효율적인 방법들, 시스템들 및 컴퓨터-프로그램(computer-program) 제품들이 필요하다는 것을 인식한다. 시스템 메모리 및 프로세서들의 수 및/또는 처리 능력들이 증가함에 따라, 서로 다른 프로세스들로부터 시스템에 대한 수요는 늘어난다(scales up). 시스템 메모리 규모(scaling)가 증가함에 따라, 기존 메모리 할당 기술들은 가상 메모리 관리 작업들의 성능 저하로 어려움을 겪는다. 기존 가상 메모리 관리 기술들은 필요에 따라 VM 메타데이터 컨테이너들의 생성을 종종 구현한다. "VM 메타데이터 컨테이너"란 애플리케이션, 프로그램 또는 프로세스들에 의해서 사용되는 메모리 객체를 설명하는 메타데이터를 저장하는 것을 담당하는 VM 메타데이터 블록들을 보유하는 컨테이너들이라 할 수 있다. 많은 양들의 메모리를 포함하는 컴퓨팅 시스템들은 모든 기존의 물리적 메모리를 가상 메모리로 표현할 수 있는 충분한 VM 메타데이터 블록들을 확보하기 위해 다수의 VM 메타데이터 컨테이너들을 활용할 수 있다.

[0013] [0024] 현재 이용 가능한 VM 관리 기술들은 단일 잠금(single lock)을 사용하여 VM 메타데이터 블록들의 할당들을 컨트롤하며, 상기 단일 잠금은 컴퓨팅 시스템의 모든 VM 메타데이터 컨테이너들에 걸쳐 액세스를 컨트롤한다. VM 메타데이터 컨테이너에 대한 메모리 객체 메타데이터의 할당은 "최초적합(first fit)" 알고리즘을 따를 수 있고, 이에 의해서 메모리 객체 메타데이터를 포함할 수 있을 만큼 충분히 큰 이용 가능한 제 1의 VM 메타데이터 컨테이너는 채워질(filled) 것이며, 만일 기존 VM 메타데이터 컨테이너가 메모리 객체 메타데이터의 공간 요건들(the space requirements)을 수용할 수 없다면 후속 VM 메타데이터 컨테이너가 추가될 수 있다. VM 메타데이터 컨테이너는 가상 메모리 작업들을 관리하는 시스템에 의해서 필요에 따라 생성되기 때문에, 오직 하나의 잠금만이 생성되고 사용되며, 그 결과 비록 다수의 메타데이터 컨테이너들이 이용 가능 할 수 있더라도, 한 번에 오직 하나의 VM 메타데이터 컨테이너만을 사용할 수 있다. 이 것은 차례로 가상 메타데이터 컨테이너들로 하여금 메모리 객체 메타데이터로 프런트-로드(front-loaded)되게 하는데, 이는 가상 메모리 할당을 관리하는 시스템들이 충분한 공간이 있는 제 1의 메타데이터 컨테이너를 선호하여서, 메타데이터 컨테이너가 가득차서 다음 이용 가능한 메타데이터 컨테이너로 이동하기 전까지, 발견된 제 1의 메타데이터 컨테이너를 사용하기 때문이다. 그 결과, 남은 메타데이터 컨테이너는 시스템이 빈 메타데이터 컨테이너로 이동하여 빈 VM 메타데이터 컨테이너를 메모리 객체 메타데이터로 채우는 것을 시작할 때까지 빈 채로 또는 대부분 빈 채로 남겨질 수 있다.

[0014] [0025] 가상 메모리 관리에 의해서 채용되는 최초적합 알고리즘(the first fit algorithm)의 해로운 영향들은 메모리의 양들이 적은 시스템들에서는 눈에 잘 띄지 않을 수 있는데, 그 이유는 적은 양의 메모리를 갖는 시스템들은 많은 수의 VM 메타데이터 컨테이너들을 사용하지 않기 때문이다. 그러나, 본 발명의 실시 예들은 많은 양의 메모리를 이용하는 시스템들은 많은 수의 VM 메타데이터 컨테이너들을 생성하지만 최초적합 기술이 오직 단일 잠금만 획득하는 구현을 사용하기 때문에 서로에 대해서 병렬로 이들 VM 메타데이터 컨테이너들을 액세스할 수 없다는 것을 인식한다. 본 출원의 실시 예들은 많은 양의 메모리를 이용하는 시스템들을 위해 가상 메모리 관리 작업들의 스케일링(scaling)을 향상시키는데, 그 결과 성능 개선을 가져오고, 최초적합 알고리즘을 이용하는 시스템들과 비교할 때 가상 메모리 관리 작업들의 질적 저하(degradation)를 감소시킨다. 본 발명의 실시 예들은 또한 다수의 잠금 메커니즘들(multiple locking mechanisms)을 사용하여 가상 메모리 잠금 메커니즘들의 세분화(granularity)를 증가시키는데, 그렇게 하여 시스템들이 복수의 VM 메타데이터 컨테이너들에 병렬로 액세스할 수 있게 해 주는데(한 번에 하나의 VM 메타데이터 컨테이너를 이용하기 보다는), 그 결과 메모리 할당 성능을 향상시킨다.

[0015] [0026] 본 발명의 실시 예들은, 예를 들어, 시스템을 부팅할 때 복수의 VM 메타데이터 컨테이너들을 사전-인스턴스화(pre-instantiate)할 수 있다. 실시 예들은 VM 메타데이터 컨테이너들에 제공된 가상 메모리의 총량이 시스템에 이용 가능한 실제 물리적 메모리(RAM과 같은)의 양과 동일한 방식으로 VM 메타데이터 컨테이너를 인스턴스화할 수 있다. 예를 들어, 만일 시스템에 대해 이용 가능한 RAM의 총량이 32TB이라면, 결합된 모든 VM 메타데이터 컨테이너들로부터 이용 가능한 결합된 가상 메모리의 총량은 32TB이다. 필요에 따라 추가 메타데이터 컨테이너들이 생성될 수 있다. 여기에 개시된 바와 같은 사전-인스턴스화된 VM 메타데이터 컨테이너들을 사용하면, 시스템이 추가 VM 메타데이터 컨테이너를 생성할 필요가 있기 전에 시스템은 모든 물리적 메모리를 가상 메모리로 할당할 수 있다. 본 발명의 실시 예들은 인스턴스화할 VM 메타데이터 컨테이너의 수를 결정하도록 시스템의 운영 체제를 지시하는 하나 또는 그 이상의 알고리즘들을 따를 수 있다. 결정들은 이용 가능한 프로세서 및 메모리 자원들의 양에 기초할 수 있고, 하나 또는 그 이상의 VM 메타데이터 컨테이너들은 컨테이너 그룹들에 배치될 수 있으며, VM 메타데이터 컨테이너의 그룹들 각각은 시스템에 의해 획득된 단일 잠금에 할당될 수 있다.

[0016] [0027] 본 발명의 실시 예들은 "가장 적게 사용된(least used)" 할당 알고리즘을 이용할 수 있다("최초 적합(first fit)" 대신에). 여기서 설명된 "가장 적게 사용된"기술들은 복수의 컨테이너 그룹들 중 하나에 할당될 수 있는 VM 메타데이터 컨테이너들의 사전-인스턴스화된 그룹들(the pre-instantiated groups)의 이점을 취한다. 신규 메타데이터 블록 할당들이 요청되었을 때, VM 메타데이터 컨테이너들은 메모리 할당기에 의해서 쿼리될 수 있다(queried by a memory allocator). 할당기는 메타데이터 컨테이너 그룹들의 사용 레벨들을 샘플(sample) 할 수 있고 그 다음 할당기는 할당 요청을 만족시키기 위해 컨테이너 그룹 내에서 가장 적게 사용되거나 더 적게 사용된(a least used or lesser used) 메타데이터 컨테이너를 선택할 수 있다. 샘플링 동안, 할당기는, 잠금 경합(lock contention)을 최소화하기 위해, 메타데이터 컨테이너들이 잠금 해제되어 있는 동안 메타데이터 컨테이너들 또는 메타데이터 컨테이너들의 그룹들을 스캔할 수 있다. 그 결과, 잠금들에 의해서 정리된(organized) 메타데이터 컨테이너들의 그룹들은 동일 비율들 또는 거의 동일 비율들의 메모리 객체 메타데이터로 채워질 수 있다(filled). 또한, 단일 잠금 내에서 다수의 메타데이터 컨테이너들을 포함하는 컨테이너 그룹

들(groupings)을 유지함으로써, 할당기의 실시 예들은 메모리 객체 메타데이터로 채우기 위해 가장 적게 사용된 메타데이터 컨테이너들의 그룹 내에서 가장 적게 사용된 메타데이터 컨테이너를 선택할 수 있다.

[0017] [0028] 본 발명의 실시 예들은, 일부 상황들에서, 메모리 객체들이 확장 요청(request to grow)을 할 수 있으며, 이는 VM 메타데이터 블록(들)의 할당을 이미 받은 메모리 객체에 의해서 소모되는 가상 메모리의 양을 증가시킨다는 것을, 추가로 인식한다. 확장 요청을 수신하는 할당기의 실시 예들은 메모리 객체 메타데이터를 수신하기 위해 이전에 할당된 오리지널 메타데이터 컨테이너를 타겟으로 함으로써 확장 요청들을 충족할 수 있다. 타겟이 된 메타데이터 컨테이너가 확장 요청을 충족하기에 충분한 자유 공간(free space)을 포함하지 않는 상황들에서도, 할당기는 그러한 요청을 만족시킬 수 있는데, 할당기에 의해서 초기에 타겟된 타겟 메타데이터 컨테이너와 동일한 컨테이너 그룹 내의 또 다른 VM 메타데이터 컨테이너에 메모리 객체 메타데이터를 할당함으로써 그러한 요청을 만족시킬 수 있고, 그 결과 동일 컨테이너 그룹 내에 VM 메타데이터 블록들의 메모리 객체의 할당들을 유지할 수 있고 따라서 액세스는 단일 잠금에 의해서 컨트롤된다. 단일 잠금 내에 메모리 객체 메타데이터를 유지함으로써 메모리 객체 할당들 다수의 잠금들 사이에(across multiple locks) 걸쳐 있음(spanning)으로부터 발생할 수 있는 잠금 스래싱(lock thrashing)에 대해서 보호할 수 있다. 단일 잠금 내에서 메모리 객체 할당을 유지함으로써 메모리 할당 관리를 최적화할 수 있는데, 작업들(operations)을 수행하거나 메모리 객체 메타데이터에 액세스하기 위해 하나의 잠금 이상을 획득할 필요를 피할 수 있기 때문이다.

[0018] [0029] 본 발명의 실시 예들은 확장 요청들이나 신규 메모리 할당들이 수신되고, 단일 잠금 내의 VM 메타데이터 컨테이너들이 거의 가득 차 있는(nearly full) 일부 상황들에서, 메모리 객체 할당들을 단일 잠금에 의해서 관리되는 단일 VM 메타데이터 컨테이너 또는 VM 메타데이터 컨테이너들의 그룹으로 국한시키는 것은 그러한 요청 당시에는 실현 가능하지 않을 수 있다는 것을 추가로 인식한다. 메모리 객체 메타데이터는 서로 다른 컨테이너 그룹들 내의 다수의 VM 메타데이터 컨테이너들 사이에 걸쳐 있을 수 있다. 본 발명의 실시 예들은 동일 잠금을 여러 번(multiple times) 획득해야 하는 것을 피하도록 메모리 관리 작업들을 수행하기 전에 컨테이너 그룹에 기초하여 작업들의 성능을 소트(sort)할 수 있다. 메타데이터 블록 할당을 위해 메모리 작업들을 메타데이터 블록들이 원래 할당된 순서대로 수행하는 대신, VM 작업들은 획득된 단일 잠금 아래에 상주하는 메모리 객체 메타데이터를 한번에 관리할 수 있고, 그 다음 상기 잠금은 상기 작업이 끝날 때 해제될 수 있으며 다시는 재-획득될(re-acquired) 필요가 없다. VM 메타데이터 블록들은 VM 메타데이터가 할당된 컨테이너 그룹들에 기초하여 그룹화될 수 있다. 예를 들어, 처음 3개의 블록들은 "잠금 a"에 할당되고, 제 4의 블록은 "잠금 b"에 할당되며, 제 5의 블록은 "잠금 c"에 할당되고, 제 6의 블록은 "잠금 b"에 할당되는 잠금들 a, b, 및 c 아래에 상주하는 복수의 VM 메타데이터 컨테이너들에 대해 VM 메타데이터 블록들의 할당이 할당될 수 있다. 처음 3개의 블록들에 대해 "잠금 a"를 먼저 획득하고, 그 다음, 제 4의 블록에 대해 잠금 b를 획득하며, 제 5의 블록에 대해 잠금 c를 획득하고, 그 다음에 제 6의 블록에 대해 다시 잠금 b를 재-획득함으로써 VM 관리 작업들을 수행하는 대신, 본 발명의 실시 예들은 처음 3개의 블록들에 대해서는 "잠금 a"를 획득하고, 처음 3개의 메타데이터 블록들은 해방하며(free), "잠금 a"는 해제하고(release), 그 다음 "잠금 b"를 획득하며, 제 4의 블록과 제 6의 블록을 해방하고, "잠금 b"를 해제 함으로써 메모리 관리 작업들을 수행할 수 있다. 상기메모리 관리 작업들은, "잠금 c"에 위치한 제 5의 블록을 해제한 후 "잠금 b"를 재-획득하는 대신에, 제 4의 블록과 제 6의 블록에 대한 할당들이 해방되었기 때문에, "잠금 b"를 완전히 해제할 수 있다.

[0019] 데이터 처리 시스템

[0020] [0030] 본 발명은 시스템, 방법, 및/또는 통합의 모든 가능한 기술적 세부 레벨에서 컴퓨터 프로그램 제품이 될 수 있다. 컴퓨터 프로그램 제품은 컴퓨터 판독 가능 스토리지 매체(또는 미디어)를 포함할 수 있으며, 이 매체상에 프로세서가 본 발명의 실시 예들을 수행하도록 하는 컴퓨터 판독 가능 프로그램 명령들을 갖는다.

[0021] [0031] 상기 컴퓨터 판독 가능 스토리지 매체는 명령 실행 장치에 의해 사용될 명령들을 유지 및 저장할 수 있는 유형의(tangible) 디바이스일 수 있다. 상기 컴퓨터 판독 가능 스토리지 매체는, 예를 들면, 전자 스토리지 디바이스, 자기 스토리지 디바이스, 광 스토리지 디바이스, 전자기 스토리지 디바이스, 반도체 스토리지 디바이스, 또는 전술한 것들의 모든 적절한 조합일 수 있으며, 그러나 이에 한정되지는 않는다. 컴퓨터 판독 가능 스토리지 매체의 더 구체적인 예들의 비포괄적인 목록에는 다음이 포함될 수 있다: 휴대용 컴퓨터 디스켓, 하드 디스크, 랜덤 액세스 메모리(RAM), 판독-전용 메모리(ROM), 소거 및 프로그램가능 판독-전용 메모리(EPROM 또는 플래시 메모리), 정적 랜덤 액세스 메모리(SRAM), 휴대용 컴팩트 디스크 판독-전용 메모리(CD-ROM), 디지털 다용도 디스크(DVD), 메모리 스틱, 플로피 디스크, 천공-카드들 또는 명령들이 기록된 홈에 있는 융기된 구조들 같이 기계적으로 인코드 된 장치, 및 전술한 것들의 모든 적절한 조합. 본 명세서에서 사용될 때, 컴퓨터 판독 가능 스토리지 매체는 무선 전파들이나 다른 자유롭게 전파되는 전자기파들, 도파관이나 기타 전송 매체(예를

들어, 광섬유 케이블을 통해 전달되는 광 펄스들)를 통해 전파되는 전자기파들, 또는 선(wire)을 통해 전송되는 전기 신호들 같이 그 자체로 일시적인(transitory) 신호들로 해석되지는 않는다.

[0022] [0032] 여기서 기술되는 컴퓨터 판독 가능 명령들은, 예를 들어, 인터넷, 근거리 네트워크, 광역 네트워크 및/또는 무선 네트워크 등의 네트워크를 통해 컴퓨터 판독 가능 스토리지 매체로부터 각각 컴퓨팅/처리 디바이스들로 또는 외부 스토리지 디바이스로부터 외부 컴퓨터로 다운로드 될 수 있다. 상기 네트워크는 구리 전송 케이블들, 광 전송 섬유들, 무선 전송, 라우터들, 방화벽들, 스위치들, 게이트웨이 컴퓨터들 및/또는 엣지 서버들을 포함할 수 있다. 각 컴퓨팅/처리 장치 내 네트워크 어댑터 카드 또는 네트워크 인터페이스는 상기 네트워크로부터 컴퓨터 판독 가능 프로그램 명령들을 수신하고 그 컴퓨터 판독 가능 프로그램 명령들을 각각의 컴퓨팅/처리 디바이스 내의 컴퓨터 판독 가능 스토리지 매체에 저장하기 위해 전송한다.

[0023] [0033] 본 발명의 연산들을 실행하기 위한 컴퓨터 판독 가능 프로그램 명령들은 Smalltalk, C++ 또는 그와 유사 언어 등의 객체 지향 프로그래밍 언어와 "C" 프로그래밍 언어 또는 그와 유사한 프로그래밍 언어 등의 종래의 절차적 프로그래밍 언어들을 포함하여, 하나 또는 그 이상의 프로그래밍 언어들을 조합하여 작성된(written) 어셈블러 명령들, 명령-세트-아키텍처(ISA) 명령들, 머신 명령들, 머신 종속 명령들, 마이크로코드, 펌웨어 명령들, 상태-셋팅 데이터, 집적회로를 위한 구성 데이터, 또는 소스 코드나 목적 코드일 수 있다. 상기 컴퓨터 판독 가능 프로그램 명령들은 전적으로 사용자의 컴퓨터상에서, 부분적으로 사용자의 컴퓨터상에서, 독립형(stand-alone) 소프트웨어 패키지로서, 부분적으로 사용자의 컴퓨터상에서 그리고 부분적으로 원격 컴퓨터상에서 또는 전적으로 원격 컴퓨터나 서버상에서 실행될 수 있다. 위에서 마지막의 경우에, 원격 컴퓨터는 근거리 네트워크(LAN) 또는 광역 네트워크(WAN)을 포함한 모든 종류의 네트워크를 통해서 사용자의 컴퓨터에 접속될 수 있고, 또는 이 접속은 (예를 들어, 인터넷 서비스 제공자를 이용한 인터넷을 통해서) 외부 컴퓨터에 이루어질 수도 있다. 일부 실시 예들에서, 예를 들어 프로그램 가능 로직 회로, 필드-프로그램 가능 게이트 어레이들(FPGA), 또는 프로그램 가능 로직 어레이들(PLA)을 포함한 전자 회로는 본 발명의 실시 예들을 수행하기 위해 전자 회로를 맞춤화하도록 상기 컴퓨터 판독 가능 프로그램 명령들의 상태 정보를 활용하여 상기 컴퓨터 판독 가능 프로그램 명령들을 실행할 수 있다.

[0024] [0034] 본 발명의 특징들이 본 발명의 실시 예들에 따른 방법들, 장치들(시스템들), 및 컴퓨터 프로그램 제품들의 플로 차트 예시도들 및/또는 블록도들을 참조하여 기술된다. 플로 차트 예시도들 및/또는 블록도들의 각 블록과 플로 차트 예시도들 및/또는 블록도들 내 블록들의 조합들은 컴퓨터 판독 가능 프로그램 명령들에 의해 구현될 수 있다는 것을 이해할 수 있을 것이다.

[0025] [0035] 이들 컴퓨터 판독 가능 프로그램 명령들은 범용 컴퓨터, 특수목적용 컴퓨터, 또는 기타 프로그램가능 데이터 처리 장치의 프로세서에 제공되어 머신(machine)을 생성하고, 그렇게 하여 그 명령들이 상기 컴퓨터 또는 기타 프로그램가능 데이터 처리 장치의 프로세서를 통해서 실행되어, 상기 플로 차트 및/또는 블록도의 블록 또는 블록들에 명시된 기능들/동작들을 구현하기 위한 수단을 생성할 수 있다. 이들 컴퓨터 판독 가능 프로그램 명령들은 또한 컴퓨터 판독 가능 스토리지 매체에 저장될 수 있으며, 컴퓨터, 프로그램가능 데이터 처리 장치 및/또는 기타 디바이스들에 지시하여 명령들이 저장된 상기 컴퓨터 판독 가능 스토리지 매체가 상기 플로 차트 및/또는 블록도의 블록 또는 블록들에 명시된 기능/동작의 특징들을 구현하는 명령들을 포함하는 제조품(an article of manufacture)을 포함하도록 특정한 방식으로 기능하게 할 수 있다.

[0026] [0036] 상기 컴퓨터 판독 가능 프로그램 명령들은 또한 컴퓨터, 기타 프로그램가능 데이터 처리 장치, 또는 다른 디바이스에 로드 되어, 상기 컴퓨터, 기타 프로그램가능 장치 또는 다른 디바이스에서 일련의 동작 단계들이 수행되게 하여 컴퓨터 구현 프로세스를 생성하며, 그렇게 하여 상기 컴퓨터, 기타 프로그램가능 장치, 또는 다른 디바이스 상에서 실행되는 명령들이 플로 차트 및/또는 블록도의 블록 또는 블록들에 명시된 기능들/동작들을 구현할 수 있다.

[0027] [0037] 도면들 내 플로 차트 및 블록도들은 본 발명의 여러 실시 예들에 따른 시스템들, 방법들 및 컴퓨터 프로그램 제품들의 가능한 구현들의 아키텍처, 기능(functionality), 및 작업(operation)을 예시한다. 이와 관련하여, 상기 플로 차트 또는 블록도들 내 각 블록은 상기 명시된 논리적 기능(들)을 구현하기 위한 하나 또는 그 이상의 실행 가능한 명령들을 포함한 모듈, 세그먼트 또는 명령들의 일부분을 나타낼 수 있다. 일부 다른 실시 예들에서, 상기 블록에 언급되는 기능들은 도면들에 언급된 순서와 다르게 일어날 수도 있다. 예를 들면, 연속으로 도시된 두 개의 블록들은 실제로는 사실상 동시에 실행될 수도 있고, 또는 이 두 블록들은 때때로 관련된 기능에 따라서는 역순으로 실행될 수도 있다. 블록도들 및/또는 플로 차트 예시도의 각 블록, 및 블록도들 및/또는 플로 차트 예시도 내 블록들의 조합들은 특수목적용 하드웨어 및 컴퓨터 명령들의 명시된 기능들 또는 동

작들, 또는 이들의 조합들을 수행하는 특수목적용 하드웨어-기반 시스템들에 의해 구현될 수 있다는 것에 또한 주목해야 한다.

[0028] [0038] 본 발명의 다양한 실시 예들에 대한 설명은 예시의 목적으로 제공되는 것이며, 개시된 실시 예들이 전부라거나 이들에 한정하려는 의도가 있는 것은 아니다. 많은 수정들 및 변형들이 기술된 실시 예들의 범위와 정신을 벗어남이 없이 이 기술 분야에서 통상의 지식을 가진 자들에게 명백할 것이다. 여기서 사용된 용어들은 본 발명의 실시 예들의 원리들, 시장에서 발견된 기술들에 대한 실제 응용 또는 기술적 개선을 잘 설명하기 위해서, 또는 이 기술 분야에서 통상의 지식을 가진 자들이 여기서 기술된 실시 예들을 이해할 수 있도록 하기 위해, 선택되고 기술되었다.

[0029] [0039] 도 1은 데이터 처리 시스템(100)의 블록도를 예시하며, 이는 여기에 기술된 컴퓨팅 작업들을 수행할 수 있는 컴퓨팅 시스템의 간략하게 한 예일 수 있다. 데이터 처리 시스템(100)은 도 2a-6에 도시된, 그리고 본 발명의 실시 예들에 따른, 컴퓨팅 환경(200), (290), (600)에 도시된 하나 또는 그 이상의 컴퓨팅 시스템들 또는 디바이스들을 표현한 것일 수 있다. 도 1은 데이터 처리 시스템(100)의 일 실시 예에 대한 예시만 제공하고, 다른 실시 예들이 구현될 수 있는 환경들에 관하여서는 어떠한 제한들도 의미하지 않는다는 것을 이해해야 한다. 일반적으로, 도 1에 예시된 컴포넌트들은 머신-판독가능(machine-readable) 프로그램 명령들을 실행할 수 있는 모든 전자 디바이스를 표현할 수 있다.

[0030] [0040] 도 1은 데이터 처리 시스템(100)의 일 예를 도시하지만, 데이터 처리 시스템(100)은 베어 메탈 컴퓨터 시스템들과 가상화 컴퓨터 시스템들을 포함하여, 다른 많은 형태들을 취할 수 있다. 예를 들어, 데이터 처리 시스템(100)은 개인용 데스크톱 컴퓨터 시스템들, 랩탑들, 노트북들, 태블릿들, 서버들, 클라이언트 시스템들, 네트워크 디바이스들, 네트워크 터미널들, 씬 클라이언트들, 씩 클라이언트들, 키오스크들, 전화 정보통신 디바이스들(예: 스마트폰들), 멀티프로세서 시스템들, 마이크로프로세서-기반 시스템들, 미니컴퓨터 시스템들, 메인프레임 컴퓨터 시스템들, 스마트 디바이스들, 또는 사물 인터넷(IoT) 디바이스들의 형태를 취할 수 있다. 데이터 처리 시스템(100)은 네트워크 컴퓨팅 환경 및/또는 분산 클라우드 컴퓨팅 환경에서 작동할 수 있으며, 이는 여기에 기술된 모든 시스템들 또는 디바이스들 및/또는 당업자들이 알고 있거나 사용하는 추가적 컴퓨팅 디바이스들 또는 시스템들도 포함할 수 있다.

[0031] [0041] 데이터 처리 시스템(100)은 하나 또는 그 이상의 프로세서(들)(103), 메모리(105), 영구 스토리지(106), 캐시(107),통신 유닛(111), 및 하나 또는 그 이상의 입/출력(I/O) 인터페이스(들)(115) 사이의 전자 통신을 공급할 수 있는 정보통신 패브릭(112)를 포함할 수 있다.정보통신 패브릭(112)은 프로세서(들)(103)(마이크로프로세서들, 통신들, 네트워크 프로세서들, 등과 같은), 메모리(105), 외부 디바이스들(117), 및 데이터 처리 시스템(100) 내의 기타 모든 하드웨어 컴포넌트들 사이에 데이터를 전달하도록 및/또는 정보를 컨트롤하도록 디자인된 모든 아키텍처를 통해 구현될 수 있다. 예를 들어, 정보통신 패브릭(112)는 주소 버스(305) 또는 데이터 버스(307)과 같은, 하나 또는 그 이상의 버스들로 구현될 수 있다.

[0032] [0042] 메모리(105) 및 영구 스토리지(106)는 컴퓨터-판독가능 저장 매체일 수 있다. 메모리(105)의 실시 예들은 랜덤 액세스 메모리(RAM)와 캐시(107) 메모리를 포함할 수 있다. 일반적으로, 메모리(105)는 적절한 모든 휘발성 또는 비-휘발성(non-volatile) 컴퓨터-판독가능 저장 매체를 포함할 수 있고 메모리(105)는 프로그래밍 된 펌웨어 또는 기타 소프트웨어를 포함할 수 있다. 여기에 기술된, 프로그램(들)(114), 소프트웨어 애플리케이션들(223), 사용자 프로세스들(221) 과 서비스들은, 컴퓨터 시스템(100)의 각각의 프로세서(들)(103) 중 하나 또는 그 이상에 의한 실행 및/또는 액세스를 위해서 메모리(105) 및/또는 영구 스토리지(106)에 저장될 수 있다.

[0033] [0043] 영구 스토리지(106)은 복수의 자기 하드 디스크 드라이브들을 포함할 수 있다. 대안으로, 또는 자기 하드 디스크 드라이브들에 대해 추가적으로, 스토리지(106)은 하나 또는 그 이상의 솔리드-스테이트 하드 드라이브들, 반도체 스토리지 디바이스들, 읽기-전용 메모리들(ROM), 소거 및 프로그램 가능 읽기-전용 메모리들(EPROM), 플래시 메모리들, 또는 프로그램 명령들 또는 디지털 정보를 저장할 수 있는 기타 모든 컴퓨터-판독가능 저장 매체를 포함할 수 있다. 영구 스토리지(106)에서 사용하는 매체의 실시 예들은 또한 이동식(removeable)일 수 있다. 예를 들어, 이동식 하드 드라이브는 영구 스토리지(106)에 사용할 수 있다. 다른 예들은 또한 영구 스토리지(106)의 일부인 다른 컴퓨터-판독가능 저장 매체로 전송하기 위해서 드라이브에 삽입되는 광학 및 자기 디스크들, 썸 드라이브들, 스마트 카드들을 포함한다.

[0034] [0044]통신 유닛(111)은 데이터 처리 시스템들(100) 사이에, 예를 들어, 정보통신 네트워크(250)를 통해 하나 또는 그 이상의 컴퓨터 시스템들 또는 디바이스들 사이에 전자 통신들을 용이하게 한다. 예시적인 실시 예에서, 통신 유닛(111)은 TCP/IP 어댑터 카드들, 무선 Wi-Fi 인터페이스 카드들, 3G, 4G, 또는 5G 무선 인터페이스 카

드들 또는 기타 유무선 정보통신 링크들과 같은 네트워크 어댑터들 또는 인터페이스들을 포함할 수 있다. 통신 네트워크들은, 예를 들어, 구리선들, 광섬유들, 무선 전송, 라우터들, 방화벽들, 스위치들, 게이트웨이 컴퓨터들, 엣지 서버들 및/또는, 디바이스들, 호스트 시스템(들)(201), 터미널들 또는 기타 네트워크 컴퓨터 시스템들을 포함하는 통신 네트워크들의 노드들의 일부이거나, 또는 노드들에 대해 접속되는, 기타 네트워크 하드웨어를 포함할 수 있다. 본 발명의 실행 실시 예들을 실시하기 위해 사용되는 소프트웨어와 데이터는 통신 유닛(111)을 통해(예를 들어, 인터넷, 근거리 통신망 또는 기타 광역 통신망들을 통해) 네트워크 환경에서 운영하는 컴퓨터 시스템에 다운로드 될 수 있다. 통신 유닛(111)로부터, 프로그램(들)(114)의 소프트웨어와 데이터는 영구 스토리지(106)에 로드 될 수 있다.

[0035] [0045] 하나 또는 그 이상의 I/O 인터페이스들(115)은 데이터 처리 시스템(100)에 접속될 수 있는 다른 디바이스들에 데이터의 입력과 출력을 허용할 수 있다. 예를 들어, I/O 인터페이스(115)는 하나 또는 그 이상의 외부 디바이스들(117)에 대한 접속을 제공할 수 있고, 하나 또는 그 이상의 외부 디바이스들(117)은 하나 또는 그 이상의 스마트 디바이스들, IoT 디바이스들, 오디오 시스템, 카메라 시스템들, 하나 또는 그 이상의 센서 디바이스(들)과 같은 레코딩 디바이스들, 키보드, 컴퓨터 마우스, 터치 스크린, 가상 키보드, 터치패드, 포인팅 디바이스, 또는 기타 휴먼 인터페이스 디바이스들과 같은 입력 디바이스들을 포함할 수 있다. 외부 디바이스들(117)은, 예를 들어, 썸 드라이브들, 휴대용 광학 또는 마그네틱 디스크들, 메모리 카드들과 같은, 휴대용 컴퓨터-판독가능 저장 매체를 또한 포함할 수 있다. I/O 인터페이스(115)는 휴먼-판독가능(human-readable) 디스플레이(118)에 접속할 수 있다. 휴먼-판독가능 디스플레이(118)은 사용자에게 데이터를 디스플레이 하는 메커니즘을 제공하고, 예를 들어, 컴퓨터 모니터나 화면, 일 수 있다. 휴먼-판독가능 디스플레이(118)는 또한 통합 디스플레이일 수 있고 태블릿 컴퓨터의 내장(built-in) 디스플레이와 같은, 터치 스크린으로서 기능을 할 수 있다.

[0036] 가상 메모리 메타데이터의 할당들 관리 시스템

[0037] [0046] 도면들을 참조하면, 도 2a-6은 하나 또는 그 이상의 VM 메타데이터 컨테이너들(402a)-(402f)(일반적으로 "VM 메타데이터 컨테이너들(402)"라 함)을 사용하여 메모리 객체들에 대한 가상 메모리의 할당을 관리하기 위한 시스템들, 방법들 및 컴퓨터 프로그램 제품들을 구현하기 위해 컴퓨팅 환경(200, 290, 600) 내에서 운영하는 하나 또는 그 이상의 데이터 처리 시스템들(100)을 사용하여 실행될 수 있는 하나의 접근방식을 도시한다. 컴퓨팅 환경들의 실시 예들(200, 290, 600)은 디바이스 네트워크(250)을 통해 상호 접속된 하나 또는 그 이상의 데이터 처리 시스템들(100)을 포함할 수 있다. 디바이스 네트워크(250)에 접속된 데이터 처리 시스템들(100)은 전문화된 시스템 또는 디바이스들일 수 있고, 상기 시스템 또는 디바이스들은 하나 또는 그 이상의 호스트 시스템(201) 및/또는 하나 또는 그 이상의 클라이언트 시스템들(227a, 227b … 227n)(일반적으로 클라이언트 시스템(227)로 참조됨)의 상호 접속을 포함할 수 있지만, 이에 국한되지는 않으며, 상기 하나 또는 그 이상의 클라이언트 시스템들은 사용자 프로세스(221), 애플리케이션(223), 프로그램(114) 또는 서비스의 하나 또는 그 이상의 메모리 객체들에 대한 데이터와 명령을 저장하기 위해 가상 메모리의 할당(들)을 요구할 수 있는 데이터 또는 입력 명령들을 처리하기 위해 호스트 시스템(201)의 자원들을 요청하고 이용할 수 있다.

[0038] [0047] 도 2a-6의 실시 예들에 예시된 데이터 처리 시스템들(100)은, 도 2a-6의 도면들에 도시된 시스템들 및 디바이스들의 엘리먼트들을 포함할 수 있을 뿐만 아니라, 도시된 전문적인 데이터 처리 시스템들은 도 1에 도시된, 그리고 위에서 기술된 데이터 처리 시스템(100)의 하나 또는 그 이상의 컴포넌트들을 추가로 포함할 수 있다. 비록 도면들에 완전히 도시되지는 않을지라도, 데이터 처리 시스템(100)의 하나 또는 그 이상의 컴포넌트들은 하나 또는 그 이상의 프로세서(들)(103), 프로그램(들)(114), 메모리(105), 영구 스토리지(106), 캐시(107), 통신 유닛(111), 입/출력(I/O) 인터페이스(들)(115), 외부 디바이스(들)(117)과 휴먼-판독가능 디스플레이(118)의 통합을 포함하는(그러나 이에 국한적되지 않는), 시스템 하드웨어(203)로서 호스트 시스템(201) 및/또는 클라이언트 시스템(227)의 실시 예들에 통합될 수 있다.

[0039] [0048] 컴퓨팅 환경(200)의 실시 예는 호스트 시스템(201)의 실시 예의 컴포넌트들을 도시하는 블록도를 예시하고, 반면에, 컴퓨팅 환경(290)은 네트워크(250)을 통해 하나 또는 그 이상의 클라이언트 시스템들과 통신하는 도 2a로부터의 호스트 시스템(201)의 블록도를 도시한다. "호스트 시스템"(201)이란 용어는 컴퓨터 네트워크에 접속된 컴퓨터 또는 기타 디바이스와 같은, 데이터 처리 시스템(100)을 가리키고, 호스트 시스템(201)에 접속한 클라이언트 시스템들(227)의 사용자들에게 컴퓨팅 자원들, 서비스들, 사용자 프로세스들(221), 애플리케이션들(223), 프로그램들(114), 등에 대한 액세스를 제공할 수 있다. 예를 들어, 호스트 시스템(201)은 웹 호스트, 클라우드 호스트, 가상 호스트, 원격 호스트 및/또는 메인프레임 컴퓨터로 운영하는 서버(실제 또는 가상)일 수 있다. 호스트 시스템(201)에 접속한 클라이언트 시스템들(227)은 컴퓨터 시스템, 컴퓨팅 디바이스 또는 다른 유형의 데이터 처리 시스템(100)일 수 있으며, 이는 네트워크(250)을 통해 호스트 시스템(201)에 접속할 수 있고

호스트 시스템(201)에 의해서 제공된 이용 가능한 자원들에 액세스할 수 있다. 예를 들어, 하나 또는 그 이상의 프로그램들(114), 애플리케이션들(223), 사용자 프로세스들(221), 서비스들, 등에 액세스할 수 있다.

[0040] [0049] 네트워크(250)의 실시 예들은 호스트 시스템(201)을 클라이언트 시스템들(227)에 링크할 수 있고 유선, 무선 또는 광섬유 접속들을 사용하여 구축될 수 있다. 호스트 시스템(201), 클라이언트 시스템(227) 및 기타 데이터 처리 시스템(100)의 실시 예들은, 네트워크 인터페이스 컨트롤러, 네트워크 인터페이스 카드 또는 네트워크(250)과의 접속을 용이하게 할 수 있는 기타 네트워크 통신 디바이스와 같은, 통신 유닛(111)을 통해 네트워크(250)에 대해서 접속할 수 있고 통신할 수 있다. 컴퓨팅 환경들(200, 290, 600)의 일부 실시 예들에서, 하나 또는 그 이상의 호스트 시스템(201) 및 클라이언트 시스템(227)은 네트워크(250)을 통해 액세스될 때 클러스터된 컴퓨터들과 경계 없는 자원들의 단일 풀로서 작동하는 컴포넌트들을 이용하여 데이터 처리 시스템들(100)을 표현할 수 있다. 예를 들어, 그러한 실시 예들은 데이터 센터, 클라우드 컴퓨팅, 스토리지 지역 네트워크(SAN), 및 네트워크-부착된 스토리지(NAS) 애플리케이션들에서 사용될 수 있다.

[0041] [0050] 통신 디바이스(111)의 실시 예들은 특정 물리적 계층과 데이터 링크 계층 표준을 사용하여 통신할 수 있도록, 전문화된 전자 회로를 구현할 수 있다. 예를 들어, 이더넷, 파이버 채널, Wi-Fi 또는 토큰 링은 호스트 시스템(201), 클라이언트 시스템(227)과 네트워크(250)에 접속된 기타 데이터 처리 시스템들(100) 사이에 데이터를 전송한다. 통신 유닛(111)은 또한 전체 네트워크 프로토콜 스택(a full network protocol stack)을 허용할 수 있고, 이는 네트워크(250)을 통해서 호스트 시스템(201), 클라이언트 시스템(227), 및 네트워크(250)의 통신 채널들을 통해 함께 링크된 다른 데이터 처리 시스템들(100)의 그룹과의 통신을 가능하게 해 준다. 네트워크(250)은 호스트 시스템(201), 클라이언트 시스템(227), 및 네트워크(250)에 접속된 다른 데이터 처리 시스템들(100)(예를 들어, 네트워크-접근가능 스토리지 매체) 사이에서 통신과 자원 공유를 용이하게 할 수 있다. 네트워크(250)의 예들은 근거리 통신망(LAN), 홈 영역 통신망(HAN), 광역 통신망(WAN), 백본 통신망들(BBN), 피어-투-피어 통신망들(P2P), 캠퍼스 통신망들, 기업 통신망들, 인터넷, 클라우드 컴퓨팅 네트워크들과 당업자들에게 알려진 기타 모든 네트워크를 포함할 수 있다.

[0042] [0051] 도면들을 다시 참조하면, 도 2a는 가상 메모리, 메모리 객체들과 VM 메타데이터 블록 할당 요청들을 관리할 수 있는 호스트 시스템(201)의 실시 예를 기술하는 블록도를 도시한다. 도면들에 의해서 도시된 바와 같이, 가상 메모리와 VM 메타데이터의 할당을 관리하는 책임이 있는 호스트 시스템(201)(또는 기타 유형들의 데이터 처리 시스템들(100))은, 도시된 바와 같이, 하나 또는 그 이상의 추상적 계층들을 포함할 수 있다. 예를 들어, 호스트 시스템(201)의 실시 예들은 시스템 하드웨어(203) 계층, 운영 체제(205) 계층(이는 커널(206)을 포함할 수 있음), 및 사용자 공간(207) 계층을 포함할 수 있다. 시스템 하드웨어(203)의 실시 예들은 호스트 시스템(201)의 하나 또는 그 이상의 물리적 컴포넌트들을 포함할 수 있으며, 상기 호스트 시스템(201)의 하나 또는 그 이상의 물리적 컴포넌트들은 호스트 시스템(201)에 의해서 이용될 수 있고 하나 또는 그 이상의 클라이언트 시스템들(227)에 제공될 수 있는 컴퓨터 자원들을 제공할 수 있으며, 상기 하나 또는 그 이상의 클라이언트 시스템들(227)은 상기 자원들에 대한 액세스를 요청하거나 또는 호스트 시스템(201)에 의해서 제공된 하나 또는 그 이상의 애플리케이션들(223), 프로그램들(114), 및/또는 서비스들을 액세스한다. 예를 들어, 클라이언트 시스템들(227)에 할당된 하나 또는 그 이상의 가상화된 컴퓨팅 시스템들을 생성함에 의해서 그렇게 한다. 시스템 하드웨어(203) 계층에서 볼 수 있는 시스템 하드웨어(203)의 예들은, 도시된 바와 같이, 하나 또는 그 이상의 프로세서들(103), 메모리(105), 영구 스토리지(106) 및/또는 I/O 인터페이스들(115)을 포함할 수 있다(그러나 이에 국한되지 않는다).

[0043] [0052] 사용자 공간(207) 계층의 실시 예들은 실행 중인 애플리케이션들(223), 사용자 프로세스들(221), 프로그램들(114), 서비스 및 기타 소프트웨어에 할당된 호스트 시스템의(201) 메모리(105)의 일부일 수 있다. 일부 실시 예들에서, 사용자 공간(207)은 운영 체제(205)의 커널(206)의 바깥에서 실행하는 모든 코드를 참조할 수 있다. 사용자 공간(207)은 다양한 프로그램들(114), 서비스들, 애플리케이션들(223)을 포함할 수 있을 뿐만 아니라, 하나 또는 그 이상의 라이브러리들(224)를 또한 포함할 수 있으며, 상기 라이브러리들(224)는 운영 체제(205)가 사용자 공간의(207) 소프트웨어 코드와 커널(206) 사이의 인터페이스를 위해서 액세스할 수 있다. 애플리케이션들(223) 및 기타 소프트웨어는 호스트 시스템(201)에 의해서 열리기 때문에, 운영 체제(205)("OS(205)")의 실시 예들은 프로그램(114)와 요구되는 모든 자원들을, 모든 플러그인들, 및 라이브러리들(224)과 함께, 사용자 공간(207)에 로드 할 수 있다. 사용자 공간(207)의 실시 예들은 메모리(105)의 양이 증가함에 따라 증가할 수 있다. 예를 들어, 호스트 시스템(201)의 RAM 및/또는 이용 가능한 기타 유형들의 물리적 메모리(301)를 늘리면, 더 많은 사용자 공간(207)이 복수의 애플리케이션들(223)을 동시에 실행하는 데 이용 가능하게 될 수 있다.

[0044] [0053] 운영 체제(205) 계층의 실시 예들은, 운영 체제(205)의 커널(206)을 포함하는, 운영 체제(205)를 포함할 수 있다. OS(205)의 실시 예들은 시스템 하드웨어(203)의 자원들, 시스템 하드웨어(203) 자원들에 액세스하기 위한 디바이스-독립 인터페이스들을 관리할 수 있고 하나 또는 그 이상의 클라이언트 시스템들(227)과의 이들 자원들의 공유를 용이하게 할 수 있다. OS(205)는 시스템 프로그램들(114)과 사용자들에 의해서 액세스되는 애플리케이션들(223) 사이에서 시스템 하드웨어(203)의 사용을 조정할 수 있다. 일부 실시 예들에서, OS(205)는 커널(206)을 포함할 수 있다. 커널(206)은 운영 체제(205)의 중앙 모듈일 수 있다. 커널(206)은 사용자 공간(207) 내에서 실행될 수 있는 프로그램들(114), 애플리케이션들(223), 사용자 프로세스들(221), 서비스들, 등과는 독립적으로 실행될 수 있다. 커널(206)은 호스트 시스템(201)의 메모리(105)로 로드 될 수 있고 호스트 시스템(201)이나 기타 데이터 처리 시스템(100)을 부팅할 때 종종 로드 될 OS(205)의 제 1의 부분이 될 수 있다. 커널(206)의 실시 예들은, 일단 메모리(105)에 로드 되면, 필수 서비스들, 애플리케이션들 및 운영 체제(205)의 일부들을 실행하기 위해서 메모리(105)에 남아있을 수 있다. 커널(206)은 사용자 공간(207) 내에서 수행되거나 실행되는 소프트웨어에 시스템 하드웨어(203)를 접속하는 것을 책임 질 수 있다. 커널(206)에 의해서 제공되는 프로그램들 및 서비스들의 예들은 다음을 포함할 수 있는데(이에 국한되지 않음), 애플리케이션들(223)의 실행을 위한 프로세스/업무 관리(209), 가상 메모리 및 메모리 객체 메타데이터의 할당을 컨트롤하기 위한 가상 메모리 관리자(213), 디바이스 드라이버들의 사용을 통한 디바이스 관리(217), 그리고 하드 디스크들 및 솔리드 스테이트 드라이브들과 같은 영구 스토리지(106) 공간의 할당을 관리하기 위한 파일 시스템 관리자(219)를 포함할 수 있다.

[0045] [0054] 가상 메모리 관리자(213)의 실시 예들은 호스트 시스템(201)의 메모리 할당 프로세스들을 담당할 수 있다. 메모리 할당은 애플리케이션들(223), 프로그램들(114), 서비스들 및 기타 형태들의 소프트웨어가 가상 메모리 공간 및/또는 물리적 메모리 공간을 할당 받는 프로세스이다. 일부 실시 예들에서, 메모리는 할당은 주로 시스템 하드웨어(203) 작업(operation)일 수 있지만 OS(205) 및/또는 소프트웨어 애플리케이션들을 통해서도 관리될 수 있다. 프로그램들(114), 애플리케이션들(223) 및 서비스들에는 실행 중인 프로그램(114), 애플리케이션(223) 또는 서비스의 요건들에 따라 메모리(105) 내에서 특정 주소가 할당될 수 있고 일단 실행된 소프트웨어가 작업들을 완료하였거나, 실행된 소프트웨어가 쉬고 있다면(idle), 메모리(105)는 해제되고 및/또는 소프트웨어의 또다른 부분에 할당될 수 있거나 또는 호스트 시스템(201)의 기본 메모리(105)에 병합될 수 있다.

[0046] [0055] 호스트 시스템(201)과 같은, 데이터 처리 시스템들(100)은, 시스템에 설치된 총 물리적 메모리(301)보다 더 많은 메모리를 주소 지정할 수 있다. 여분의 메모리를 가상 메모리라 할 수 있다. 가상 메모리의 실시 예들은 영구 스토리지(106) 디바이스의 일부분 또는 섹션(section)으로 배치될 수 있다. 예를 들어, 하드 디스크는 호스트 시스템의(201) 메모리(105)를 에뮬레이트하는 전용 가상 메모리 섹션으로 설정되어 가상 메모리 공간을 제공할 수 있다. 예를 들어, 영구 스토리지(106) 디바이스에서 RAM을 에뮬레이트 함으로써 제공할 수 있다. 가상 메모리 방식(scheme)을 사용하면, 프로그램들(114), 애플리케이션들(223) 및 사용자 공간에서 실행하는 기타 소프트웨어는 물리적 메모리(301)보다 클 수 있고, 호스트 시스템(201)이 영구 스토리지(106)을 사용하여 물리적 메모리(301)의 사용을 확장할 수 있게 할 수 있으며, 가상 메모리의 각 가상 주소가 물리적 주소로 변환될 수 있기 때문에 메모리(105) 보호를 추가로 제공할 수 있다. 가상 메모리 관리자(213)의 실시 예들은 영구 스토리지(106)에 페이지 파일(들)(또는 "스왑 파일들")이라고 불리는 메모리(105) 이미지들을 생성할 수 있고 OS(205)는 마치 가상 메모리가 물리적 메모리(301)에 저장되는 것처럼 영구 스토리지(106)의 가상 메모리 위치들을 주소 지정할 수 있다. OS(205)가 물리적 메모리(301)에 의해서 저장되지 않은 메모리의 블록을 요청할 때마다, 가상 메모리 관리자(213)은 최근에 사용되지 않은 물리적 메모리(301)로부터 메모리 블록을 가져와서 그것을 페이징 파일에 쓸 수 있다. 가상 메모리 관리자(213)는 OS(205)가 요청한 메모리의 블록을, 페이징 파일로부터 추가로 읽을 수 있다. 그런 다음가상 메모리 관리자(213)은 페이징 파일로부터 그 메모리의 블록을 가져와서 최근에 사용되지 않았던 오래된 블록의 장소에 위치한 물리적 메모리(301)로 이동할 수 있다. 이러한 프로세스를 스왑핑(swapping)이라고 하는데, 물리적 메모리(301)로부터의 페이지 파일들을 가상 메모리로써 영구 스토리지(106)에 의해서 저장된 페이지 파일들과 교환하기 때문이다.

[0047] [0056] 도 3a를 참조하면, 가상 메모리 기술들이 스왑핑을 사용하여 영구 스토리지(106) 내의 페이지 파일들을 물리적 메모리(301)에 저장된 페이지 파일들로 교환하는 방법을 보여주는 일 예가 도시된다. 도면에 도시된 바와 같이, 프로세서(103)은, 예를 들어, 주소 버스(305)를 통해, 물리적 메모리(301)의 주소로부터 메모리의 블록(즉, 애플리케이션(223)에 의해서 사용되는 메모리 객체)을 요청한다. 만일 요청된 페이지 파일이 물리적 메모리(301)이 아니라 영구 스토리지(106)에 위치되어 있다면, 가상 메모리의 페이지 파일은 물리적 메모리(301)의 주소로 스왑될 수 있고 데이터 버스(307)를 통해 반환된 데이터는, 프로세서(103)으로 돌아온다. 일부 실시

예들에서, 프로세서(103)는 시스템 하드웨어(203)에 내장될(built into) 수 있는 메모리 관리 유닛(303)(MMU)을 사용할 수 있다. MMU(303)는 영구 스토리지(106)의 가상 메모리에 저장된 페이지 파일의 위치를 기술하는 논리 주소들을 물리적 메모리(301)의 물리적 주소로 변환하는 기능을 수행할 수 있다. MMU(303)의 실시 예들은 요청된 메모리 블록들을 물리적 메모리(301)로부터 검색하기 위해 도면들 3a - 3c에 도시된 것처럼 가상 주소들을 물리적 주소들로 변환하는 기능 또는 업무 및/또는 페이지 파일을 영구 스토리지(106)에서 물리적 메모리(301)로부터 스왑핑하는 기능 또는 업무를 수행할 수 있다. 일부 실시 예들에서, 캐시(107)는 가상 메모리 관리 중에 자주 또는 최근에 MMU(303)에 의해서 사용된 논리적 또는 물리적 주소들의 검색을 개선하기 위해 호스트 시스템(201)에 포함될 수 있다. 예를 들어, 논리적 캐시(도 3b에 도시된 것 바와 같이)는 물리적 주소들로 변환되어 있는 최근 또는 자주 사용된 논리적 주소를 검색하기 위해 MMU(303)에 의해서 액세스될 수 있다. 마찬가지로, 물리적 캐시(도 3c에 도시된 바와 같이)가 변환된 논리 주소들에 대응될 수 있는 자주 사용된 물리적 주소들의 빠른 색인(lookup)을 제공하기 위해 포함될 수 있다.

[0048] [0057] 일부 실시 예들에서, 호스트 시스템(201)의 가상 메모리 관리자(213)는 할당기(215)를 포함할 수 있다. 할당기(215)의 실시 예들은 사용자 공간(207) 내에 배치된 애플리케이션들(223) 또는 소프트웨어에 대한 할당을 위해 메모리 객체들(애플리케이션들(223)에 의해서 사용되는 메모리의 블록들)과 호스트 시스템(201)에 이용 가능한 메모리의 자유 블록들을 관리하는 것과 연관된 업무들이나 기능들을 수행하는 것을 담당할 수 있다. 할당기(215)에 의해서 관리되는 메모리 객체들과 메모리의 자유 블록들을 보다 효율적으로 관리하기 위해서, 할당기(215)의 실시 예들은 메타데이터)(여기서는"메모리 객체 메타데이터"라 함)를 생성할 수 있고 저장할 수 있는데, 이 메타데이터는 할당기(215)에 액세스할 수 있는 가상 메모리의 덩어리들(chunks)을 기술한다. 예를 들어, 메타데이터는 물리적 또는 가상 메모리에 의해서 저장된 메모리 객체들의 크기, 상태들 및 포인터 주소들을 기술할 수 있다. 할당기(215)의 실시 예들은 관리되는 가상 메모리의 블록들을 기술하는 메타데이터에 전용인 하나 또는 그 이상의 VM 메타데이터 블록들에 메모리 객체 메타데이터를 조직하고 저장할 수 있다. 일부 실시 예들에서, VM 메타데이터 블록들은 메모리 객체에 대한 할당을 위해 이용 가능한 한 덩어리의 가상 메모리의 헤더일 수 있다.

[0049] [0058] VM 메타데이터 블록들의 실시 예들은, 도 4a-4b의 예시적 예에 도시된 바와 같이, 할당기(215)에 의해서 하나 또는 그 이상의 VM 메타데이터 컨테이너들(402a-402f)(일반적으로 VM 메타데이터 컨테이너들(402)라 함)로 조직화될 수 있다. VM 메타데이터 컨테이너(402)의 실시 예들은 복수의 VM 메타데이터 블록들을 포함할 수 있다. 컨테이너당 VM 메타데이터 블록들의 수는 호스트 시스템(201)에 이용 가능한 물리적 메모리(301)의 총량, VM 메타데이터 컨테이너들(402)의 크기 및 호스트 시스템(201)에 의해서 생성된 VM 메타데이터 컨테이너들의 수에 따라서 달라진다. 일부 실시 예들에서, VM 메타데이터 컨테이너들(402)의 크기와 수는 운영 체제(205)에 의해서 자동으로 결정될 수 있다. 대체 실시 예들에서, 운영 체제(205)에 의해 생성된 VM 메타데이터 컨테이너의 스토리지 크기 및/또는 수는 사용자-지정(user-specified)일 수 있다. 그러므로, VM 메타데이터 컨테이너(402)당, 할당기(215)에 이용 가능한 VM 메타데이터 블록들의 수는, 호스트 시스템(201), 컴퓨팅 환경 및 VM 메타데이터 컨테이너들(402)를 인스턴스화하는(instantiating) 방법에 따라 달라질 수 있다.

[0050] [0059] 도 4a에 도시된 예시적인 실시 예에서, 운영 체제(205)에 의해서 인스턴스화된 VM 메타데이터 컨테이너(402)의 크기와 수는 가상 메모리로서 호스트 시스템(201)에 이용 가능한 물리적 메모리(301)의 총량을 표현하기에 충분한 VM 메타데이터 컨테이너(402)를 포함할 수 있다. 한번에 물리적 메모리(301)의 총량을 표현하기에 충분한 VM 메타데이터 컨테이너(402)를 인스턴스화 함에 의해서, 할당기(215)는 신규 컨테이너(403)의 생성을 요청하기 전에 모든 물리적 메모리(301)을 가상 메모리로서 할당할 수 있다. 할당기(215)의 실시 예들은 호스트 시스템(201)이 부팅(boot up)될 때 총 물리적 메모리(301)를 가상 메모리로 표현하기 위해 VM 메타데이터 컨테이너들의 총수를 사전-인스턴스화(pre-instantiate) 할 수 있다. VM 메타데이터 컨테이너들(402)의 인스턴스화가 운영 체제(205)에 의해서 결정되는 실시 예들에서, VM 메타데이터 컨테이너(403)의 인스턴스화는, 호스트 시스템(201)에 이용 가능한 프로세서(103) 및 메모리(105) 자원들의 양에 따라, VM 메타데이터 컨테이너들(402)의 최적의 수 및/또는 그에 관한 크기를 결정하는 알고리즘에 기초하여 구현될 수 있다. 예를 들어, 만일 OS(205)가 단일 VM 메타데이터 컨테이너가 128GB의 가상 메모리를 포함하도록 선택한다면, 물리적 메모리(301)의 총량은 인스턴스화 할 VM 메타데이터 컨테이너의 총수를 결정하기 위해 단일 VM 메타데이터 컨테이너(402)의 선택된 크기에 의해 나누어 질 수 있다. 예를 들어, 만일 호스트 시스템(201)이 1024GB의 물리적 메모리(301)를 포함한다면, 운영 체제는, 개별적으로 128GB 조각을 표현하는, 8개의 VM 메타데이터 컨테이너들을 인스턴스화 할 수 있다.

[0051] [0060] 당업자는 단일 메타데이터 컨테이너(402)의 크기를 조정하면, 대략 개별 VM 메타데이터 컨테이너들(40

2)이 인스턴스화 예시될 수 있다는 점을 인지할 것이다. 예를 들어, 개별 VM 메타데이터 컨테이너들(402)에 128GB를 할당하는 대신에, 단일 VM 메타데이터 컨테이너는 64GB에 할당될 수 있다. 따라서, 1024GB의 물리적 메모리(301)를 포함하는 호스트 시스템은 물리적 메모리(301)의 전체 1024GB를 완전히 표현하기 위해서, 운영 체제(205)에 의해 16개의 VM 메타데이터 컨테이너들(402)로 인스턴스화 될 수 있다. 마찬가지로, 일부 실시 예들에서, 사용자는 단일 메타데이터 컨테이너(402)의 원하는 크기를 입력할 수 있고, 호스트 시스템(201)의 총 물리적 메모리(301)에 따라, 운영 체제(205)는 호스트 시스템의 물리적 메모리(301)의 총량을 반영하는 사용자-규정(user-prescribed) 크기에 개별적으로 대응하는 복수의 VM 메타데이터 컨테이너들(402)를 생성할 수 있다.

[0052] [0061] 도 4a에 도시된 바와 같이, 호스트 시스템(201)에 의해서 인스턴스화 된 복수의 VM 메타데이터 컨테이너들(402)은 복수의 컨테이너 그룹들(407a)-(407d)(도면들에서 "그룹(407a)" ~ "그룹(407d)"로 도시되었고 여기서는 일반적으로 컨테이너 그룹(407)이라 함)로 더 조직될 수 있다. 각 컨테이너 그룹(407)은 컨테이너 그룹에 대한 액세스를 컨트롤하는 하나의 잠금(a lock)에 할당될 수 있다. 도 4a-4b에 도시된 예시적인 실시 예에서, VM 메타데이터 컨테이너(402)는 둘 또는 그 이상의 컨테이너 그룹들(407)로 조직되어 있다. VM 메타데이터 컨테이너(402)를 둘 또는 그 이상의 컨테이너 그룹들(407)로 나눔에 의해서, 모든 VM 메타데이터 컨테이너들(402)에 대한 액세스를 컨트롤하기 위해 단일 잠금을 사용하는 대신에, 둘 또는 그 이상의 잠금들이 할당된 컨테이너 그룹에 대한 액세스를 독립적으로 컨트롤하여 복수의 병렬 메모리 할당 작업들(operations)이 수행될 수 있게 한다. 단일 잠금 시스템 아래에서는, 후속 메모리 할당 작업으로 이동하기 전에 수행될 각 메모리 할당 작업이 진행될 때까지 기다려야 하는 것에 때문에 호스트 시스템(201)이 제한될 수 있다(hampered). 도 4a의 실시 예에 의해서 도시된 바와 같이, 운영 체제(205)에 의해서 사전-인스턴스화된 VM 메타데이터 컨테이너들(402)는 3개의 분리된 컨테이너 그룹들(407a, 407b, 407c)로 나뉘어지고, 개별 컨테이너 그룹들(407)은 컨테이너 그룹(407)당 두 개의 VM 메타데이터 컨테이너들(402)을 더 포함한다. 비록 컨테이너 그룹당 오직 두 개의 VM 메타데이터 컨테이너들(402)만 있는 오직 세 개의 컨테이너 그룹들(407)만이 도 4a에서 예시되는 이러한 예에서 도시되었지만, 어떠한 수의 VM 메타데이터 컨테이너들(402)도 컨테이너 그룹(407)당 인스턴스화 될 수 있다. 예를 들어, 3, 4, 5, 10, 15, 20, 등. 또는 더 많은 수의 VM 메타데이터 컨테이너들(402)이 개별 컨테이너 그룹들(407)의 각각에 할당될 수 있다. 게다가, 어떠한 수의 컨테이너 그룹들(407)도 VM 메타데이터 컨테이너들(402)을 조직하기 위해 획득될 수 있다. 예를 들어, 운영 체제(205)는 복수의 VM 메타데이터 컨테이너(402)를 2, 4, 6, 12, 25, 50, 100, 등 또는 더 많은 수의, 컨테이너 그룹들(407)로 조직될 수 있다. 더욱이, 신규 VM 메타데이터 컨테이너(403)가 운영 체제(205)에 의해서 생성됨에 따라, 신규 VM 메타데이터 컨테이너(403)는 할당된 잠금들(407a, 407b, 407c)에 의해서 컨트롤되는 기존 컨테이너 그룹들에 추가될 수 있거나 또는 새롭게 획득된 잠금(407d)에 의해서 컨트롤되는 신규 컨테이너 그룹(407d)로 조직될 수 있다.

[0053] [0062] 일부 실시 예들에서, 획득된 컨테이너 그룹들(407) 각각은, 도 4a에서 도시된 바와 같이, 동일 수의 VM 메타데이터 컨테이너들(402)을 포함할 수 있다. 일부 예들에서, 컨테이너 그룹(407)당 VM 메타데이터 컨테이너들의 수는 비대칭적으로 조직될 수 있고, 일부 컨테이너 그룹들(407)에서 다른 컨테이너 그룹들(407)보다 많거나 적게 VM 메타데이터 컨테이너들(402)을 포함할 수 있다. VM 메타데이터 컨테이너들(402)의 비대칭 수의 이유들은 시스템마다 다를 수 있다. 예를 들어, 가상 메모리로써 호스트 시스템의 총 물리적 메모리(301)를 표현하기 위해 인스턴스화된 VM 메타데이터 컨테이너들의 수는 홀수(odd number)일 수 있다. 다른 예들에서, 도 4a에서 도시된 바와 같이, 신규 메타데이터 컨테이너(403)의 생성이 컨테이너 그룹들(407) 사이에서 VM 메타데이터 컨테이너들(402)의 이전 대칭을 방해할 수 있다. 예를 들어, 도 4a에서, 컨테이너 그룹들(407a, 407b 및 407c)은 두 개의 메타데이터 컨테이너들(402a)-(402f)를 포함하지만, 그러나, 컨테이너 그룹(407d)에 배치된 신규 VM 메타데이터 컨테이너(403)의 생성으로, 컨테이너 그룹(407d)는 오직 하나의 신규 VM 메타데이터 컨테이너(403)만을 포함한다(적어도 다음 VM 메타데이터 컨테이너(402)가 생성될 때까지). 일부 예들에서, 복수의 신규 VM 메타데이터 컨테이너들(403)이 한 번에 생성될 수 있다. 예를 들어, 도 4a에 도시된 것과 같은 하나 대신에 컨테이너 그룹(407d)에 할당된 두 개의 신규 VM 메타데이터 컨테이너들(403)을 생성할 수 있다.

[0054] [0063] 할당기(215)의 실시 예들은 할당 알고리즘에 따라 VM 메타데이터 블록들을 VM 메타데이터 컨테이너들(402)에 할당할 수 있다. 할당기(215)에 의해서 사용되는 할당 알고리즘의 예시적인 실시 예는, 도 4a에 도시한 바와 같이 가장 적게 사용된 VM 메타데이터 컨테이너(402) 내의 VM 메타데이터 블록으로 메모리 객체를 기술하는 메타데이터를 할당하는 알고리즘을 사용하여 메모리 객체 메타데이터를 VM 메타데이터 블록들에 할당하는 알고리즘을 따름으로써, 하나의 할당된 잠금(an assigned lock)에 의해서 컨트롤되는 다수의 컨테이너 그룹(407) 시스템의 이점을 취할 수 있다. VM 메타데이터 컨테이너(402)를 순서대로 채우는 결과를 가져오는 위에서 기술한 메타데이터를 할당하는 제 1의 이용 가능한 알고리즘과는 달리, 메타데이터를 가장 적게 사용된 메타데이터 컨테이너(402)에 할당하는 것은, 도 4b에서 도시된 바와 같이 VM 메타데이터 컨테이너들(402)이 컨테이너 그룹

들(407)에 걸쳐 VM 메타데이터 블록들을 보다 고르게 할당하는 결과를 가져올 수 있다. 도 4b의 예시적인 실시예에서, 할당된 VM 메타데이터 블록들(408a-408f)은 다수의 컨테이너 그룹들(407a)-(407c) 사이를 걸치는 VM 메타데이터 컨테이너들(402a)-(402f) 사이에서 고르게(또는 거의 고르게) 확산된다(spread). 신규 메타데이터 블록 할당이 요청되면, 할당기(215)는 스캔을 수행함에 의해서 컨테이너 그룹들(407) 및/또는 컨테이너 그룹들(407) 내의 VM 메타데이터 컨테이너들(402)의 사용 레벨들을 샘플하는 업무(the task of sampling)를 수행할 수 있다. 상기 샘플링 스캔의 결과에 기초하여, 할당기(215)의 실시 예들은 복수의 컨테이너 그룹들(407) 사이에서 가장 적게 사용된 컨테이너 그룹(407)인 컨테이너 그룹(407)을 선택할 수 있다. 일부 실시 예들에서, 할당기(215)는 잠금 경합(lock contention)을 최소화하기 위해 사용 레벨들의 샘플링 스캔들 동안 잠금을 획득함이 없이 컨테이너 그룹들(407)을 스캔할 수 있다. 상기 샘플링 스캔들은 간단하거나 또는 비-포괄적(brief or non-exhaustive)이어서 컨테이너 그룹들(407) 내의 VM 메타데이터 컨테이너들(402)의 대략적인 사용 레벨들을 빠르게 제공할 수 있다. 샘플링 스캔들의 실시 예들은 가장 적게 사용된 컨테이너 그룹(407)의 절대 측정 값을 반드시 제공하지는 않으며, 그보다는 추정된 사용 레벨만 제공할 수 있을 뿐이다. 가장 적게 사용된 컨테이너 그룹(407)을 식별함으로써, 할당기(215)는 상기 요청을 만족시키기에 충분한 VM 메타데이터 블록들을 포함할 수 있는 컨테이너 그룹(402) 내의 하나 또는 그 이상의 VM 메타데이터 컨테이너들(402)에 대한 할당 요청의 충족을 타겟으로 할 수 있다. 할당기(215)는 메모리 객체 메타데이터를 가장 적게 사용된 컨테이너 그룹(407) 내에서 가장 적게 사용된 VM 메타데이터 컨테이너(402)의 VM 메타데이터 블록에 저장할 수 있다.

[0055] [0064] 할당기(215)의 실시 예들은 또한 기존 메모리 객체 메타데이터가 VM 메타데이터 블록에 이미 할당한 메모리 객체의 확장 요청들(requests to grow)을 내보내는 가상 메모리 관리 기능들과 연관 업무들도 수행할 수 있다. 할당기(215)는 메모리 객체의 메타데이터를 VM 메타데이터 블록에 이미 저장하고 있을 수 있는 하나 또는 그 이상의 VM 메타데이터 컨테이너들(402)를 먼저 타겟으로 함으로써 기존 메모리 객체들의 확장 요청들을 충족할 수 있다. 할당기(215)는 메모리 객체와 연관된 기존 메모리 객체 메타데이터를 포함하는 VM 메타데이터 컨테이너(402)를 스캔하고 메모리 객체의 메타데이터를 이전에 저장한 동일 VM 메타데이터 컨테이너(402)를 사용하여 할당기(215)가 확장 요청을 만족시킬 수 있는지를 결정할 수 있다. 만일 충분한 자유 공간이 이전에 사용된 VM 메타데이터 컨테이너(402)에 제공되지 않았다면, 할당기(215)는 이전에 사용된 VM 메타데이터 컨테이너(402)와 동일한 컨테이너 그룹(407) 내의 다른 VM 메타데이터 컨테이너(402)를 사용하여 상기 요청을 만족시키려고 시도할 수 있다. 예를 들어, 만일 확장 요청(a growth request)이 할당기(215)가 VM 메타데이터 컨테이너(402a) 내의 VM 메타데이터 블록에 메타데이터를 이전에 할당하였던 메모리 객체를 확장하는 것을 추구하고 있다면, 할당기(215)는 상기 확장 요청을 충족하기에 충분한 VM 메타데이터 블록들이 있는지를 결정하기 위해 VM 메타데이터 컨테이너(402a)를 스캔할 것이다. 만일 충분한 자유 공간이 이용 가능하다면, 할당기는 상기 메모리 객체를 위한 메타데이터를 VM 메타데이터 컨테이너(402a) 내의 VM 메타데이터 블록에 쓸 수 있다. 만일 그렇지 않다면, 할당기는, 컨테이너 그룹(407a) 내에 있는 메모리 객체의 모든 메타데이터를 유지하기 위해서, 상기 메모리 객체에 VM 메타데이터 블록을 할당하기에 충분한 자유 공간이 이용 가능한지를 결정하기 위해 VM 메타데이터 컨테이너(402b)를 스캔할 것이다.

[0056] [0065] 일부 예들에서, 할당기(215)가 동일한 컨테이너 그룹(407) 내의 메모리 객체를 위한 메타데이터를 유지하는 것은 가능하지 않을 수 있으며, 그 결과 서로 다른 잠금들에 의해서 컨트롤되는 복수의 컨테이너 그룹들(407) 사이에 걸쳐 있는 두 개의 분리된 VM 메타데이터 컨테이너들(402) 내의 VM 메타데이터 블록들에 메모리 객체 메타데이터를 할당할 수 있다. 예를 들어, 컨테이너 그룹(407a) 내에 상주하는 VM 메타데이터 컨테이너(402b)와 컨테이너 그룹(407b) 내에 상주하는 VM 메타데이터 컨테이너(402d)에 메모리 객체를 위해 VM 메타데이터 블록들을 할당할 수 있다. 다수의 잠금들(407) 사이에 걸치는 그러한 상황은 단일 컨테이너 그룹(407) 내의 VM 메타데이터 컨테이너(402)가 가득 차거나(full) 및/또는 기존 메모리 객체의 확장 요청들을 수용하기에 충분한 VM 메타데이터 블록들이결여(lack)되었을 때 일어날 수 있다.

[0057] [0066] 일부 실시 예들에서, 할당기(215)는, 메모리 객체 메타데이터(503)이 복수의 컨테이너 그룹들(407)에 걸쳐 확산되어 액세스가 복수의 잠금들(507a-507e)에 의해서 컨트롤 될 수 있을 때 작업(an operation)을 수행하기 위해, 호스트 시스템(201)에 의해서 액세스되고 사용되는 메모리 객체 메타데이터(503)에 관한 메모리 관리 작업들을 간소화(streamline)할 수 있다. 예를 들어, 메모리 객체 메타데이터(503)을 저장하기 위해 더 이상 필요하지 않는 VM 메타데이터 블록들을 해방하는(freeing) 작업들을 수행할 수 있다. 메타데이터가 할당된 순서로 메모리 객체와 연관된 메타데이터 블록 할당(501)에 관한 작업들을 수행하는 대신에, 할당기(215)의 실시 예들은 컨테이너 그룹들(407)에 기초하여 메모리 관리 작업들을 수행할 수 있으며 따라서 잠금(507)을 한 번 획득하고 그 다음 잠금(507)을 해제함으로써 모든 작업들을 수행할 수 있다. 전체 컨테이너 그룹(407)을 위해 작업들을 한 번에 수행함으로써, 할당기(215)는 잠금(507a-507e)을 여러 번 획득하고 재획득할 필요가 없다. 오히려,

VM 관리 작업은 전체 컨테이너 그룹(407)을 위해수행 될 수 있고, 그 다음 잠금(507)은 해제될 수 있는데, 미래의 작업을 수행하기 위해 잠금을 다시 재-획득할 필요가 없이, 해제될 수 있다.

[0058] [0067] 예를 들어, 도 5는 복수의 컨테이너 그룹들(407)과 잠금들(507) 사이에 걸쳐 있는 메모리 객체 메타데이터(503)을 포함하는 메모리 객체를 위한 메타데이터 블록 할당(501)에 관한 메모리 작업들을 할당기(215)가 수행하는 예를 도시한다. 메모리 객체 메타데이터(503a1에서 503a8)는 잠금(507a)에 의해서 컨트롤되는 컨테이너 그룹(407a)에 할당된다. 메모리 객체 메타데이터(503b1-503b6)는 잠금(507b)에 의해서 컨트롤되는 컨테이너 그룹(407b)에 할당되고, 한편 메모리 객체 메타데이터(503c, 503d, 및 503e)는 잠금들(507c, 507d, 507e)에 의해서 각각 컨트롤되는, 컨테이너 그룹(407c, 407d, 및 407e)에 각각 할당된다. 할당기(215)가 메모리 관리 작업들을 수행할 때(메모리 객체를 기술하는 메모리 객체 메타데이터(503)을 저장하는 VM 메타데이터 블록을 해방하는 것(freeing)과 같은), 할당기(215)는, 메타데이터 블록 할당(501)에 의해 도시된 순서로 하는 대신에, 컨테이너 그룹들(407)에 기초하여 메모리 작업들을 수행함으로써 작업들을 간소화할 수 있다. 예를 들어, 할당기(215)는 먼저, 필요에 따라 여러 번 개별 잠금(507)의 획득을 초래하는, 메모리 객체 메타데이터(503a1)를 컨트롤하는 잠금(507a)을 획득하는 것(acquiring), 메모리 객체 메타데이터(503a1)를 해방 하는 것(freeing), 도 5에 도시된 바와 같이 메타데이터 블록 할당(501a)를 따라 이동하는 것, 그리고 메모리 객체 메타데이터(503b1)을 해방하기 위해 다음 잠금(507b)를 획득하는 것, 그 다음에 메모리 객체 메타데이터 (503c1)을 해방 하기 위해 잠금(507c)를 획득하는 것(등등)을 피할 수 있다. 예를 들어, 메타데이터 블록 할당(501)에 도시된 메모리 객체 메타데이터(503)의 순서로 메모리 작업들을 수행한다면, 잠금(507a)은 8번, 잠금(507b)은 6번, 및 잠금들(507c, 507d, 및 507e) 각각은 4번 획득되는 결과를 초래할 것이다. 대신, 할당기(215)는 잠금들(507a-507e)을 각각 한 번씩 획득하여, 획득된 잠금(507)에 의해서 컨트롤되는 모든 메모리 객체 메타데이터(503)를 위한 메모리 관리 작업들을 수행하고, 잠금(507)은 해제하며, 다음 잠금(507)으로 이동할 수 있다. 예를 들어, 할당기(215)는 잠금(507a)을 획득하고, 잠금(507a)이 획득된 동안 메모리 객체 메타데이터 (503a1-503a8)에 관한 메모리 작업들을 수행하며(그것과 관련된 VM 메타데이터 블록들을 해방하는 것과 같은), 잠금(507a)을 해제하고 그리고 잠금(507b)의 획득(acquiring), 메모리 객체 메타데이터(503b1-503b6)을 저장하는 메타데이터 블록들의 해방(freeing), 및 잠금(507b)의 해제(releasing)를 수행 하러 이동할 수 있다. 따라서, 할당기(215)는 잠금들(507c, 507d, 507e)과 연관된 나머지 컨테이너 그룹들(407c, 407d, 407e)을 위해 상기 프로세스를 반복할 수 있다.

[0059] [0068] 할당기(215)의 실시 예들은 병렬 및/또는 동시에 발생하는 다수의 할당 및 할당-해제(de-allocation) 작업들을 지원할 수 있다. VM 메타데이터 컨테이너들(402) 사용의 실시 예들은 유동적인 절차(a fluid procedure)일 수 있고 호스트 시스템(201)의 OS(205)는 호스트 시스템(201)의 변화하는 상태에 따라 필요한 것보다 더 많은 VM 메타데이터 컨테이너들(402)을 실수로 생성할 수 있다. 애플리케이션(223)이 닫히거나 쉬고(idle) 있으면, 애플리케이션들(223)과 연관된 메모리 객체들은 메모리(105)로부터 해제될 수 있고, 이는 할당기(215)로 하여금 해제된 메모리 객체들의 메모리 객체 메타데이터를 제거할 수 있게 하고, 자유 VM 메타데이터 블록들을 VM 메타데이터 컨테이너들(402)로 반환하게 한다. 이에 따라, 메모리 객체들을 해제하고 자유 VM 메타데이터 블록들을 생성함으로써, VM 메타데이터 컨테이너들(402)의 스토리지 공간은 신규 VM 메타데이터가 새롭게 해방된 VM 메타데이터 블록들에 기록될 수 있도록 또는 메모리 객체들의 기존 메타데이터가 확장할 수 있는 공간을 제공할 수 있도록 이용 가능하게 된다.

[0060] [0069] 할당기(215)의 실시 예들은 최소한의 사용가능한 자원들의 양을 유지하고 호스트 시스템(201)에 의한 사용을 위해 손쉽게(readily) 이용 가능하게 하기 위해 운영 체제들(205)의 시스템-전체 계획(system-wide scheme)을 따를 수 있다(""최소 자유" 계획(a"min free" scheme)"이라 함). 할당기(215)의 실시 예들은 신규 VM 메타데이터 컨테이너들이 커널(206)으로부터 요청되어야 하는지를 결정하기 위해 최소 자유 계획에 반하여 신규 VM 메타데이터 컨테이너들(403)의 생성을 요청하는 시스템 호출들을 상쇄할 수 있다(balance). 위에서 언급한 바와 같이, VM 메타데이터 블록의 유용성(availability)은 지속적으로 변화될 수 있는데, 호스트 시스템(201)의 현재 상태가 신규 VM 메타데이터 컨테이너(403)의 생성 요청이 필요하다고 할당기(215)에 표시하면서, 하나 또는 그 이상의 VM 메타데이터 블록들은 신규 VM 메타데이터 컨테이너(403)의 생성 요청 및/또는 인스턴스화의 실행 전에 해방될 수 있다(free up). 예를 들어, VM 메타데이터 블록 할당 요청은 할당기(215)에 의해서 수신되면 상기 할당기는 VM 메타데이터 컨테이너들(402)에 쿼리하여(query) VM 메타데이터 컨테이너들 중 어느 것도 상기 요청을 충족하기에 충분한 자유 공간을 포함하지 않음을 찾아낸다. 할당기(215)는 신규 VM 메타데이터 컨테이너(403)을 생성하기 위해 커널 프로세스를 포스트 한다(post). 그러나, 한편, 애플리케이션(223)의 다른 프로세스는 모든 VM 메타데이터 블록들을 해방해 버리는데(has freed), 그 결과 상기 요청을 충족하기에 충분한 자유 블록들이 발생하게 된다. VM 메타데이터 컨테이너(403)을 인스턴스화하기 전에(Prior to

instantiating), 상기 커널 프로세스는 호스트 시스템(201)의 상태를 쿼리하여 모든 VM 메타데이터 컨테이너들(402)에 관하여 자유롭게 이용 가능한 자원들의 양들을 지정한 최소 자유 계획들의 요건들을 충족하기에 충분한 자유 공간이 존재하는지를 결정할 수 있다. 만일 쿼리가 자유롭게 이용 가능한 자원들의 최소 레벨들이 충족된다는 표시를 반환한다면(포스트 된 커널 프로세스 요청과 쿼리의 시간 사이에 자원들이 해방되었다고 표시한다면), 커널 프로세스는 신규 VM 메타데이터 컨테이너(403)의 생성 요청을 충족하지 않고, 가상 메모리 관리의 유동성과, 불필요한, VM 메타데이터 컨테이너들(402)의 의도하지 않은 생성을 방지한다.

[0061] [0070] 호스트 시스템(201)의 예시 실시 예들에서, 신규 VM 메타데이터 컨테이너들(403)을 생성하기 위해 할당기(215)에 의해서 호출된 커널 프로세스는 신규 VM 메타데이터 컨테이너들(403)의 생성을 담당하는 단일 실체(the single entity)일 수 있다. 따라서, 그러한 실시 예 아래에서, 신규 VM 메타데이터 컨테이너들(403)의 생성은, 커널 프로세스 요청들이 포스트 된 순서로, 순차로 진행되며(serialized), 한 번에 하나씩 신규 VM 메타데이터 컨테이너들(403)의 생성이 이루어진다. 요청들 및 생성 프로세스의 순차화(serialization)로 인해, 커널 프로세스의 예시적인 실시 예는 신규 VM 메타데이터 컨테이너들(403)의 생성들 사이에 호스트 시스템(201)의 상태를 쿼리하여, 최소 자유 계획이 준수되고 있는지와 자유롭게 이용 가능한 자원들의 최조량이 존재하는지를 결정할 수 있다. 신규 VM 메타데이터 컨테이너들(403)의 생성 사이에 쿼리들을 수행함으로써, 커널 프로세스는, 자유롭게 이용 가능한 자원들의 최소량이 감지되면(즉, VM 메타데이터 블록들이 하나 또는 그 이상의 애플리케이션들(223) 또는 프로세스들에 의해서 해방되었다면) 즉시, 신규 VM 메타데이터 컨테이너들(403)의 계속적인 생성을 방지할 수 있고, 시스템 전체 최소 자유 레벨(the system wide min free level)을 충족하기에 충분한 VM 메타데이터 블록들이 이용 가능하게 되었을 때, 신규 VM 메타데이터 컨테이너들(403)의 의도하지 않은 초과-생성(inadvertent over-creation)를 방지 할 수 있다.

[0062] [0071] 호스트 시스템(201)의 일부 실시 예들에서, 기존 VM 메타데이터 컨테이너들(402)이 모두 가득 찼거나(full) 거의 가득 찼을(nearly full) 때, 호스트 시스템(201)의 가상 메모리 관리가 단일 잠금(407) 계획으로 이양되는 것(devolving)을 방지하기 위한 보호 장치들(safeguards)이 구현될 수 있다. 호스트 시스템(201)이 VM 메타데이터 블록들이 해방되기를 기다릴 때 신규 할당 요청들에 VM 메타데이터 블록들을 할당하기 위한 유일한 옵션으로 단일의 신규 메타데이터 컨테이너(403)만이 있는 상황을 피하기 위해, 커널 프로세스의 실시 예들은, 할당기(215)에 의해서 그렇게 하도록 포스트 될 때, 한번에 다수의 신규 VM 메타데이터 컨테이너들(403)을 생성할 수 있다. 위에서 언급한 바와 같이, 커널 프로세스는 신규 VM 메타데이터 컨테이너들(403)의 생성 사이에서 자유롭게 이용 가능한 자원들의 최소량을 계속적으로 체크할 수 있고, 일단 시스템 전체 최소 자유 레벨 계획을 충족하기에 충분한 이용 가능한 VM 메타데이터 블록들이 기존 VM 메타데이터 컨테이너들(402) 사이에서 자유롭게 이용 가능하다면, 신규 VM 메타데이터 컨테이너들(403)의 생성을 중단할 수 있다.

[0063] [0072] 도면들을 참조하면, 도 6은, 호스트 시스템(201)이 클라이언트 시스템들(227)에 의해서 액세스되는 하나 또는 그 이상의 애플리케이션들(223)을 컨테이너 클러스터(602)(컨테이너 603a - 603n으로써 도시됨)의 다수의 분리된 컨테이너화 된(containerized) 환경들로 컨테이너화 할 수 있는, 컴퓨팅 환경(600)을 포함하는, 대체 실시 예를 도시한다. 호스트 시스템(201)의 실시 예들은 호스트 시스템(201)에 의해서 제공되고 호스트 되는 컨테이너화 된 애플리케이션들을 위해서 운영 체제(205)를 통해 가상 메모리 작업들을 본 발명과 일치하는 방식으로 관리할 수 있다. 컨테이너들(603)의 실시 예들은, 컨테이너들(603)의 운영 환경 내에서, 애플리케이션 이미지(607a - 607n)와, 소프트웨어 종속물들(dependencies) (605a - 605)을 포함한다. 호스트 시스템(201)은 다수-사용자(multi-user) 운영 체제(예: 호스트 운영 체제(205))를 실행할 수 있고 애플리케이션(223)의 기능들의 실행 및 수행을 위해서 컨테이너화 된 컴퓨터 환경을 포함하는 하나 또는 그 이상의 컨테이너들(603a-603n)(일반적으로 컨테이너들(603)이라 함)에 호스트 시스템 하드웨어(203)를 통해 컴퓨팅 자원들을 제공할 수 있다. 컨테이너 클러스터(602)의 컨테이너들(603)에 의해서 공유되는 운영 체제(205)의 실시 예들은 애플리케이션 이미지들(607)을 실행하기 위해서 컨테이너들(603)에 할당된 가상 메모리를 관리할 수 있다.

[0064] [0073] 컴퓨팅 환경(600)의 실시 예들은 다수의 네트워크들, 도메인들, 및/또는 지리적 위치들 사이에 걸쳐 있을 수 있는 복수의 데이터 센터들로 조직될 수 있다. 일부 실시 예들에서, 상기 데이터 센터들은 물리적 위치들에 상주할 수 있고, 한편, 다른 실시 예들에서, 상기 데이터 센터들은 클라우드 네트워크에 걸쳐 분산된 복수의 호스트 시스템들(201) 및/또는 물리적으로 지역화 된(localized) 그리고 분산된 호스트 시스템들(201)의 조합을 포함할 수 있다. 데이터 센터들은, 도 6에서 도시된 바와 같은, 컨테이너들(603)의 환경 내에서 캡슐화 된(encapsulated) 컨테이너화 된 애플리케이션 이미지(607)를 실행하고 런(run) 하기 위해, 하나 또는 그 이상의 호스트 시스템들(201)을 포함할 수 있고, 하나 또는 그 이상의 호스트 시스템들(201)은 호스트 시스템 하드웨어(203), 호스트 운영 체제(205) 및/또는 컨테이너화 소프트웨어(containerization software)(601)을 제공하며,

컨테이너화 소프트웨어(601)는 오픈-소스 독커(the open-source Docker) 및/또는 오픈시프트 소프트웨어(OpenShift software)를 포함하나, 이에 국한하지 않는다. 비록 도 6에 도시된 예시적인 실시 예는 세 개의 컨테이너들(603)을 포함하지만, 도 6에서 도시되지만, 도 6의 실시 예는 복수의 컨테이너들(603)이 호스트 시스템(201)에 의해서 호스트 되고 및/또는 관리될 수 있다는 개념을 단지 예시할 뿐이다. 도 6의 실시 예는 호스트 시스템들(201)이 단지 3개의 컨테이너들(603)만 호스트하는 것으로 제한된다는 것을 암시하는 것으로 간주되어서는 안 된다. 호스트 시스템(201)에 의해서 호스트 되는 컨테이너들(603)의 수 및 관리되는 가상 메모리 작업들의 수는, 호스트 시스템 하드웨어(267)에 기초한, 이용 가능 컴퓨팅 자원들의 양과 컨테이너화 소프트웨어(601)에 의해서 컨테이너들(603) 내에 실행되는 애플리케이션 이미지들(607)에 의해서 요구된 컴퓨팅 자원들의 양에 따라서 달라질 수 있다.

[0065] [0074] 컨테이너화 소프트웨어(601)의 실시 예들은 컨테이너화 된 프로그램들(114) 및 애플리케이션들(223)의 개발(developing), 배송(delivering), 및 실행(running)을 위한 소프트웨어 플랫폼으로 작동할 수 있을 뿐만 아니라, 컨테이너들(603)의 컴퓨팅 환경 내에서 코드를 신속하게 배치(deployment)할 수 있도록 한다. 컨테이너들(603)의 실시 예들은 호스트 시스템들(201) 사이뿐만 아니라, 서로 다른 지리적 위치들에서 운영하고 있을 수 있는, 서로 다른 데이터 센터들 사이에서 이전될 수 있으며(transferred), 이 때문에 컨테이너들(603)은 컨테이너화 소프트웨어(601)을 실행하는 모든 호스트 시스템(201) 상에서 실행될 수 있다. 컨테이너화 소프트웨어(601)는 호스트 시스템(201)이, 호스트 시스템 하드웨어(203)과 호스트 시스템(201)의 기타 하부구조(infrastructure)로부터, 컨테이너화 된 애플리케이션들(223)과 프로그램들(221)을 분리할 수 있게 하고 실행하고 있는 컨테이너로 수송된 애플리케이션들(223)과 호스트 시스템의 운영 체제(205)를 통해 수행되고 실행되는 컨테이너화 된 애플리케이션들(223)을 위해서 가상 메모리 작업들을 관리할 수 있게 한다.

[0066] [0075] 컨테이너화 소프트웨어(601)은 컨테이너 603의 격리된 환경 내에서 애플리케이션 이미지들(607)을 패키지화하고 실행할 수 있는 능력을 호스트 시스템(201)에 제공한다. 개별 컨테이너들(603)에 의해서 제공된 격리 및 보안은 호스트 시스템(201)이, 단일 호스트 시스템(201) 상의 모든 애플리케이션 이미지들(607)을 위해 가상 메모리 작업들을 동시에 관리하면서, 애플리케이션 이미지들(607)의 다수의 인스턴스들(multiple instances)을 실행할 수 있게 한다. 컨테이너(603)은, 가상 머신들에 의해서 통상적으로 사용되는, 하이퍼바이저에 대한 어떤 필요성도 제거하기 때문에, 가벼울 수 있다. 오히려, 컨테이너들(603)은 호스트 운영 체제(205)의 커널(206) 내에서 직접 실행될 수 있다. 그러나, 애플리케이션 이미지들(607)의 실시 예들은 컨테이너화로 가상 머신들의 가상화를 결합함으로써 이익을 얻을 수 있다. 예를 들어, 호스트 시스템(201)은 컨테이너화 소프트웨어(601)을 실행하는 가상 머신일 수 있다.

[0067] [0076] 컨테이너화 소프트웨어(601)의 실시 예들은 컨테이너화 엔진(a containerization engine)을 포함할 수 있다(도시되지 않음). 컨테이너화 엔진은, 데몬 프로세스(daemon process)를 실행하는 서버 프로그램, REST API를 포함할 수 있는 클라이언트-서버 애플리케이션일 수 있으며, REST API는, 애플리케이션들(223) 및/또는 기타 프로그램들이 데몬 프로세스와 말하고(talk) 애플리케이션 이미지(607), 및 명령들을 입력하기 위한 커맨드-라인(command-line) 인터페이스(CLI) 클라이언트에 명령들을 제공하는데 사용할 수 있는, 하나 또는 그 이상의 인터페이스들을 명시한다. 하나의 실시 예에서, 클라이언트 시스템(227)은 호스트 시스템(201)의 컨테이너화 소프트웨어(601)와 통신하기 위해 CLI를 사용하여 명령들을 입력할 수 있다. 도 6에 도시된 예시적인 실시 예에서, 호스트 시스템(201)에 클라이언트 시스템(227)에 의해서 제공된 커맨드들은 애플리케이션 인터페이스(229)를 통해 메모리(105)에 또는 호스트 시스템(201)과 인터페이스 하는 클라이언트 시스템(227)의 영구 스토리지(106)에 로드 된 입력일 수 있다.

[0068] [0077] CLI의 실시 예들은 컨테이너화 엔진의 REST API를 사용하여 자동화된 스크립팅을 통해서 또는 직접 CLI 커맨드들을 통해서 데몬을 컨트롤하거나 데몬과 상호 작용할 수 있다. CLI로부터 수신된 명령들에 응답하여, REST API를 통해, 데몬은 컨테이너화 소프트웨어(601)의 메모리 객체들을 생성하고 관리할 수 있으며, 상기 컨테이너화 소프트웨어(601)는 컨테이너들(601) 내에 상주하는 하나 또는 그 이상의 소프트웨어 이미지들, 컨테이너들(601) 그 자체, 네트워크들, 데이터 볼륨들, 플러그인들, 등을 포함한다. 이미지는 컨테이너(603)를 생성하는 명령들을 갖는 읽기-전용(read-only) 템플릿일 수 있고 맞춤화 가능(customizable)할 수 있다. 컨테이너들(603)은 소프트웨어 이미지의 실행 가능 인스턴스일 수 있다. 컨테이너들(603)은 컨테이너화 소프트웨어(601) API를 사용하거나 CLI를 통해 생성, 시작, 중지, 이동 또는 삭제될 수 있다. 컨테이너들(603)은 하나 또는 그 이상의 네트워크들(250)에 접속될 수 있고, 스토리지 디바이스에 부착될 수 있으며 및/또는 컨테이너(603)의 현재 상태에 기초하여 신규 이미지를 생성할 수 있다.

[0069] 가상 메모리 메타데이터의 할당 관리 방법

[0070] [0078] 도 7-8b의 도면들은 메모리 객체 메타데이터의 할당을 관리하기 위해서 컴퓨터-구현 방법을 수행하는 알고리즘들(700, 800)의 실시 예들을 나타낸다. 예시되고 도시된, 알고리즘들(700, 800)은, 여기서 설명된 바와 같이, 도 1의 데이터 처리 시스템(100)에 의해서 일반적으로 정의된, 더 구체적으로는 도 2a-6에 도시된 전문적인 데이터 처리 시스템들의 실시 예들에 의해서 정의된, 하나 또는 그 이상의 컴퓨터 시스템들을 사용할 수 있다. 당업자는 도 7 및 도 8a-8b에 의해서 기술된 알고리즘들(700, 800)의 단계들이 제공된 순서와 다른 순서로 수행될 수 있다는 것을 인식해야 한다. 알고리즘들(700, 800)은 여기에 기술된 모든 단계들을 반드시 수행할 것을 요구하지 않을 수도 있다. 오히려, 알고리즘들(700, 800)의 일부 실시 예들은 단계들 중 하나 또는 그 이상을 사용하여 단계들의 서브세트를 수행함으로써 방법들을 바꿀 수 있으며, 이에 관하여는 후술한다.

[0071] [0079] 알고리즘(700)의 실시 예들은 호스트 시스템(201)의 부팅 프로세스 동안 복수의 VM 메타데이터 컨테이너들(402)를 사전-인스턴스화하는 프로세스를 기술할 수 있다. 알고리즘(700)은 단계(701)에서 시작할 수 있다. 단계(701)에서, 호스트 시스템(100) 또는 다른 유형의 컴퓨팅 시스템과 같은, 데이터 처리 시스템(100)은 호스트 시스템(201)을 켜고 운영 체제(205)를 로드하기 위해 부팅 프로세스를 시작한다. 운영 체제(205)의 커널(206)은 메모리(105)에 로드될 수 있다. 단계(703)에서, 운영 체제(205) 및/또는 커널(206)은 VM 메타데이터 컨테이너들(402)을 생성하기 위해서 인스턴스화 모드(the instantiation mode)를 결정할 수 있다. 예를 들어, VM 메타데이터 컨테이너들(402)의 인스턴스화가 운영 체제(205)의 파라미터들의 지시로 또는 사용자-지시(user-directed) 파라미터들의 지시로 생성되는지를 결정할 수 있다. 단계(705)에서 결정이 내려질 수 있으며, 만일 VM 메타데이터 컨테이너들을 생성하기 위한 인스턴스화 모드가 OS(205) 지시된 것으로 검출된다면, 알고리즘(700)은 단계(707)로 진행될 수 있다. 반대로, 만일 단계(705)에서 사용자-지시 인스턴스화 모드가 검출된다면, 알고리즘(700)은 단계(709)로 진행될 수 있고 사용자-지시 파라미터들에 기초하여 규정된 수(the prescribed number)의 VM 메타데이터 컨테이너들(402)을 생성할 수 있다.

[0072] [0080] 단계(707)에서, OS-지시(OS-directed) 인스턴스화 모드들을 위해서, 운영 체제(205)는 호스트 시스템(201)의 이용 가능 자원들에 기초하여 생성할 VM 메타데이터 컨테이너들의 수를 계산할 수 있다. 특히, 상기 운영 체제는, 가상 메모리로서 총 실제 물리적 메모리를 VM 메타데이터 컨테이너들(402)로 정확하게 표현하기 위해서, 가상 메모리를 관리하는 임무가 부여된 호스트 시스템(201) 또는 다른 데이터 처리 시스템(100)에 전속된 물리적 메모리(301)의 총량을 계산할 수 있고, 그리고 OS(205)에 의해서 결정된 VM 메타데이터 컨테이너들(402)의 사전-선택된(pre-selected), 최적화된 또는 원했던 크기에 의해서 물리적 메모리(301)의 총량을 나눌 수도 있다. 단계(709)에서, 운영 체제(205)의 커널 프로세스는, 인스턴스화 모드가 OS-지시인지 또는 사용자-지시인지에 기초하여, OS 지시 파라미터들 및/또는 사용자-지시 파라미터들 중 하나에 따라 선택된 크기의 복수의 VM 메타데이터 컨테이너들(402)를 생성할 수 있다. 알고리즘(700)의 단계(711)에서, 운영 체제(205)는, 단계(709)에서 생성된 VM 메타데이터 컨테이너들(402)을, 개별 컨테이너 그룹들(407)에 할당된 잠금(507)에 의해서 컨트롤되는 컨테이너 그룹들(407)로 그룹화할 수 있다. 예시적인 실시 예에서, 운영 체제(205)는 VM 메타데이터 컨테이너(402)를 적어도 두 개의 컨테이너 그룹들(407)로 조직할 수 있다. 그러나, 운영 체제(205)는 최적으로 결정된, 또는 사용자-정의(user-defined) 파라미터들에 의해서 설정된 임의의 수의 컨테이너 그룹들(407)을 생성할 수 있다. 따라서, 단계(713)에서, 상기 운영 체제는 호스트 시스템(713)을 운영 체제(205)로 부팅하는 것을 완료하고 부팅 프로세스 중에 인스턴스화 된 VM 메타데이터 컨테이너들(402)를 사용하여, 사용자-주도 작업들(user-driven operations) 및 가상 메모리 메타데이터의 관리를 포함하는, 추가 작업들을 수행할 준비를 한다.

[0073] [0081] 알고리즘(700)의 실시 예들은 위에서 기술한 대로 VM 메타데이터 컨테이너들(402)의 인스턴스화에 이어 알고리즘(800)으로 나아갈 수 있다. 알고리즘(800)은 단계(801)에서 시작을 진행할 수 있다. 단계(801)에서, 호스트 시스템(201)은 애플리케이션(223), 프로그램(114), 서비스 또는 기타 실행된 소프트웨어의 부분에서 사용되는 메모리의 블록에 메모리 객체를 할당하라는 메모리 할당 요청을 수신할 수 있다. 가상 메모리 관리자(213)의 할당기(215)에는 상기 할당 요청을 관리하고 메모리 객체를 기술하는 메모리 객체 메타데이터를 VM 메타데이터 컨테이너들(402)의 하나 또는 그 이상의 VM 메타데이터 블록들에 쓰는 업무가 부여될 수 있다(tasked). 할당기(215)는, 단계(803)에서, 할당 요청을 분석할 수 있고 (801)단계에서 수신된 할당 요청의 유형을 결정할 수 있다. 예를 들어, 상기 할당 요청이 신규 메모리 객체를 기술하는 메타데이터 관리를 위한 것 인지 하나 또는 그 이상의 VM 메타데이터 컨테이너들(402)에 저장된 메타데이터를 이미 포함하는 기존 메모리 객체 확장 요청인지를 결정할 수 있다.

[0074] [0082] 단계(805)에서, 단계(803)의 분석에 기초하여, 할당 요청이 신규 메모리 객체를 기술하는 신규 메타데이터를 저장하기 위한 것인지 또는 기존 VM 메타데이터 컨테이너(402)의 하나 또는 그 이상의 VM 메타데이터 블록

들에 이전에 저장된 메타데이터를 갖는 기존 메모리 객체를 확장하기 위한 것인지에 관한 결정이 내려질 수 있다. 만일, 상기 요청이 신규 메모리 객체 메타데이터를 관리하기 위한 것이라는 결정이 내려진다면, 알고리즘(800)은 단계(817)로 진행할 수 있다. 반대로, 만일 상기 요청이 기존 메모리 객체 메타데이터를 확장하기 위한 것이라면, 알고리즘(800)은 단계(807)로 진행할 수 있다. 단계(807)에서, 할당기(215)는 확장하도록 요청된 메모리 객체를 기술하는 메모리 객체 메타데이터를 이전에 저장했던 이전에 사용된 VM 메타데이터 컨테이너(402)를 초기에 타겟으로 할 수 있다. 할당기(215)는, 타겟이 된 VM 메타데이터 컨테이너(402)를 스캔할 수 있는데, 이는 타겟 된 VM 메타데이터 컨테이너(402) 내에서 이용 가능한 공간의 양이 확장 요청을 충족하는 데 필요한 공간의 양과 동일하거나 이 보다 더 큰지를 결정하기 위해서이다.

[0075] [0083] 단계(809)에서, 타겟이 된 VM 메타데이터 컨테이너(402)에 관하여 단계(807)에서 수행된 스캔에 기초하여, 타겟이 된 VM 메타데이터 컨테이너가 적어도 하나의 이용 가능한 VM 메타데이터 블록에 추가의 메모리 객체 메타데이터를 쓰기에 충분한 자유 공간을 포함하는지에 관한 결정이 내려진다. 만일 VM 메타데이터 컨테이너(402)에 공간이 충분하면, 알고리즘(800)은 단계(825)로 진행할 수 있고 할당기(215)는 타겟이 된 VM 메타데이터 컨테이너(402)의 VM 메타데이터 블록을 확장 요청의 주체인 메모리 객체를 위해 메모리 객체 메타데이터를 이전에 저장한 VM 메타데이터 컨테이너(402)에 할당할 수 있다. 반대로, 단계(809)에서, 만일 타겟이 된 VM 메타데이터 컨테이너(402) 내에 상기 확장 요청을 충족시키기에 공간이 충분하지 않다(insufficient)는 결정이 할당기(215)에 의해서 내려지면, 알고리즘(800)은 단계(811)로 진행할 수 있다. 단계(811)에서, 할당기(215)는 단계(807 및 809)에서 충분한 공간이 있는 지를 결정하기 위해 이전에 타겟이 되고 스캔 된 VM 메타데이터 컨테이너(402)와 동일한 컨테이너 그룹(407) 내에 다수의 VM 메타데이터 컨테이너들이 있는지를 추가로 결정할 수 있다. 만일 동일한 컨테이너 그룹(407) 내에 추가의 VM 메타데이터 컨테이너들(402)이 없다는 결정이 단계(811)에서 내려지면, 상기 알고리즘은 단계(817)로 진행할 수 있다. 그러나, 만일 동일한 컨테이너 그룹(407) 내에 추가적인 VM 메타데이터 컨테이너들(402)이 있다는 결정이 할당기(215)에 의해서 단계(811)에서 내려지면, 상기 알고리즘은 단계(813)로 진행할 수 있고, 여기서 모든 VM 메타데이터 컨테이너들(402)이 충분한 양의 공간을 확인하기 위해 스캔 되고 평가되었는지 아닌지를 추가로 결정할 수 있다. 만일 단계(813)에서, 모든 VM 메타데이터 컨테이너들(402)이 동일한 컨테이너 그룹(407) 내에서 스캔 되었고 상기 VM 메타데이터 컨테이너들(402) 중 어느 것도 충분한 공간을 포함하지 않는다고 결정되었다면, 알고리즘(800)은 단계(817)로 진행할 수 있다.

[0076] [0084] 반대로, 단계(813)에서, 만일 동일한 컨테이너 그룹(407) 내의 모든 VM 메타데이터 컨테이너들(402)이 스캔 되지 않았다는 결정이 할당기(215)에 의해서 내려지면, 알고리즘(800)은 단계(815)로 진행할 수 있다. 단계(815)에서, 할당기(215)는 단계(807)에서 스캔 된 VM 메타데이터 컨테이너와 동일한 컨테이너 그룹(407) 내의 나머지 VM 메타데이터 컨테이너들(402)을 스캔할 수 있다. 일부 실시 예들에서, 할당기(215)는, 상기 스캔에 기초하여, 컨테이너 그룹(407) 내에서 가장 적게 사용된 메타데이터 컨테이너(402)가 확장 요청을 만족시키기에 충분한 공간을 포함하는지 아닌지를 결정하기 위해 가장 적게 사용된 VM 메타데이터 컨테이너(402)를 먼저 타겟으로 할 수 있다. 알고리즘(800)은 단계(809)로 돌아가서 진행할 수 있고 신규로 타겟이 되어 (815)단계에서 스캔 된 VM 메타데이터 컨테이너(402)가 충분한 자유 공간을 포함하는지에 관한 또 다른 결정을 내릴 수 있으며 그리고 위에서 이전에 기술된 바와 같이 알고리즘(800)에 따라 진행할 수 있다.

[0077] [0085] 단계(817)에서, 할당기(215)는 VM 메타데이터 컨테이너들(402)을 포함하는 가장 적게 사용된 컨테이너 그룹을 위해서 컨테이너 그룹들(407)을 스캔할 수 있다. 단계(819)에서, 가장 적게 사용된 컨테이너 그룹(407)이 다수의 VM 메타데이터 컨테이너들(402)을 포함하는지에 관한 결정이, 컨테이너 그룹들(407)의 스캔에 기초하여, 내려질 수 있다. 만일 상기 스캔이 가장 적게 사용된 컨테이너 그룹(407) 내에 다수의 VM 메타데이터 컨테이너들(402)이 없다고 결정하면, 상기 알고리즘은 단계(823)로 진행할 수 있고, 여기서 가장 적게 사용된 컨테이너 그룹(407) 내의 유일한 VM 메타데이터 컨테이너(402)가 확장 요청이나 메모리 객체 메타데이터의 신규 할당을 위한 요청 중 하나를 만족시키기에 충분한 공간을 포함하는지를 결정할 수 있다. 반대로, 단계(819)에서, 만일 다수의 VM 메타데이터 컨테이너들(402)이 가장 적게 사용된 컨테이너 그룹(407) 내에서 존재한다는 결정이 할당기(215)에 의해서 내려지면, 할당기(215)는 계류중인 할당 요청(신규 요청 또는 확장 요청 중 하나)을 충족하기에 적절한 충분한 공간이 있는지를 결정하기 위해 가장 적게 사용된 컨테이너 그룹(407) 내에서 가장 적게 사용된 VM 메타데이터 컨테이너(402)를 타겟으로 하여 스캔 할 수 있다.

[0078] [0086] 단계(823)에서, 가장 적게 사용된 컨테이너 그룹(407) 내의 가장 적게 사용된 VM 메타데이터 컨테이너(402) 내에서 충분한 공간이 이용 가능한지에 관한 결정이 내려진다. 만일 충분한 공간이 가장 적게 사용된 컨테이너 그룹(407)의 VM 메타데이터 컨테이너(402) 중 하나에서 발견되면, 상기 알고리즘은 단계(825)로 진행할 수 있고, 여기서 선택된 VM 메타데이터 컨테이너(402) 내의 VM 메타데이터 블록에 메모리 객체 메타데이터를 할

당할 수 있다. 그러나, 만일 단계(823)에서 가장 적게 사용된 컨테이너 그룹 내의 가장 적게 사용된 VM 메타데이터 컨테이너가 상기 할당 요청을 충족하기에 충분한 공간이 없다는 결정이 내려지면, 알고리즘(800)은 단계(826)로 진행할 수 있다.

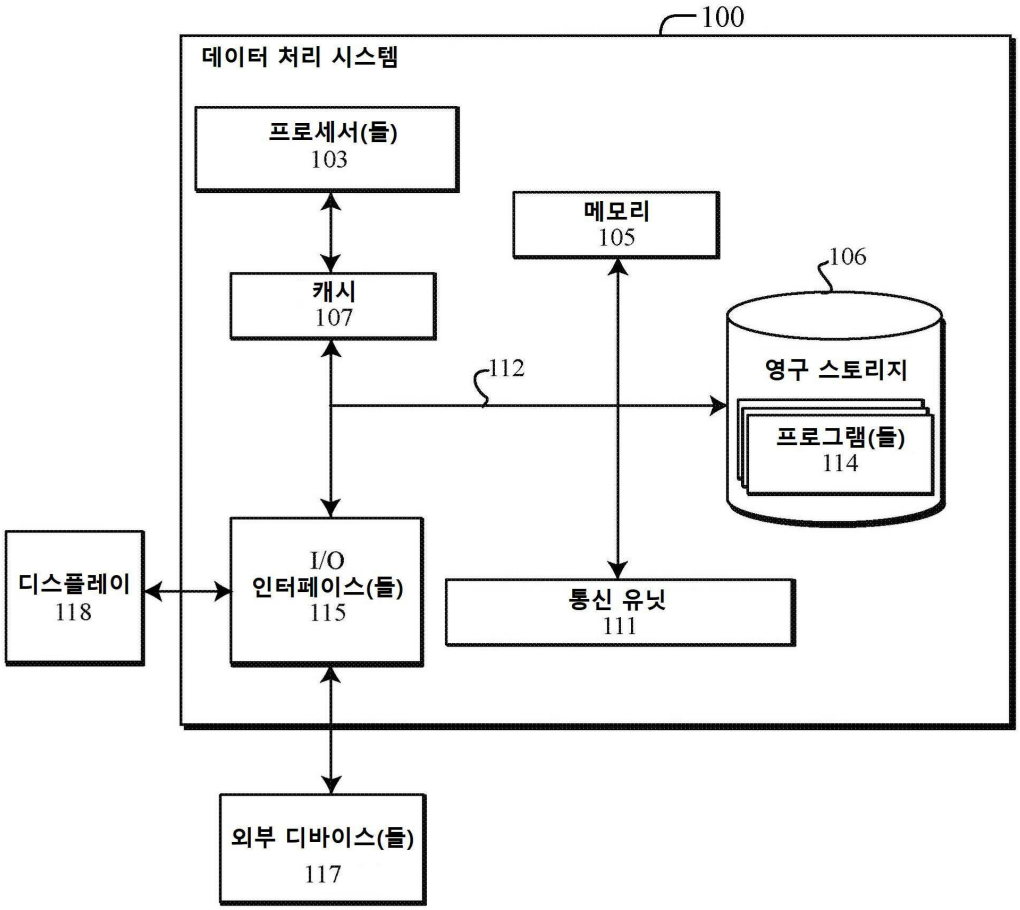
[0079] [0087] 단계(826)에서, 할당기(215)는 상기 할당 요청을 충족하기에 충분한 공간이 있는 하나 또는 그 이상의 신규 VM 메타데이터 컨테이너들(403)을 생성하는 요청을 커널 프로세스(kproc)에 포스트할 수 있다. 단계(827)에서, 상기 알고리즘은 상기 kproc 생성 요청이 커널(206)에 포스트 되었는지를 결정한다. 만일 신규 VM 메타데이터 컨테이너(403)를 생성하는 kproc 요청이 포스트되지 않으면, 신규 VM 메타데이터 컨테이너(403)의 생성은 충족되지 않고 kproc 슬립 모드로 돌아간다. 반면에, 만일 커널 프로세스에 대한 생성 요청이 포스트되었다는 결정이 내려지면, 알고리즘(800)은 단계(829)로 진행할 수 있고, 여기서 호스트 시스템(201)에서 이용 가능한 VM 메타데이터 블록들의 양을 결정하기 위해 상기 시스템 상의 모든 기존 VM 메타데이터 컨테이너들(402)을 스캔할 수 있다. 단계(829)에서의 스캔에 기초하여, 단계(831)에서 커널(206)은 시스템 전체 최소 량의 자유 자원들이 상기 할당 요청을 충족하기 위해 이용 가능한지, 이 경우에는 상기 요청이 이루어진 시간과 상기 생성요청이 커널(206)에 포스트 된 시간 사이에 상기 할당 요청을 충족하기 위해 자유 자원들이 이용 가능하게 되었는지, 아닌 지에 관한 결정을 내린다. 만일 자유롭게 이용 가능한 시스템 전체 자원들의 양이 OS(205)에 의해서 설정된 최소 자유 계획에 의해서 세트 된 임계값보다 더 크다면, 상기 알고리즘은 단계(837)로 진행하고, 여기서 kproc는 슬립 모드로 돌아가며 자원들의 할당은 상기 할당 요청을 완료하기에 충분한 해방된(freed up) 공간을 갖는 기존 VM 메타데이터 컨테이너(402)를 사용하여 충족될 수 있다.

[0080] [0088] 만일, 단계(831)에서, 커널(206)이 자유 자원들에 관한 시스템 전체 유용성(the system wide availability)이 OS(205)의 최소 자유 계획에 의해서 설정된 양 아래에(below) 있다고 결정하면, 커널(206)은 컨테이너 생성 모드에 들어갈 수 있다. 단계(833)에서, 상기 커널(206)이 단일 컨테이너 생성 모드에 들어갔는지 또는 다수(multiple) 컨테이너 생성 모드에 들어갔는지에 관한 결정이 추가로 내려진다. 만일 커널(206)이 단일 컨테이너 생성 모드에 들어갔다면, 알고리즘(800)은 단계(834)로 진행하며 커널(206)은 상기 할당 요청을 충족하기에 충분한 공간이 있는 신규 VM 메타데이터 컨테이너(403)을 생성하고 kproc는 슬립 모드로 돌아간다. 만일, 단계(833)에서, 커널(206)이 다수 컨테이너 생성 모드에 들어갔다면, 상기 알고리즘은 단계(835)로 진행하며 할당 요청을 충족하기에 충분한 공간을 포함하는 신규 VM 메타데이터 컨테이너(403)를 생성할 수 있다. 단계(836)에서, 알고리즘(800)은 다수의 신규 VM 메타데이터 컨테이너들(403)에 대한 생성 요청이 만족되었는지를 추가로 결정한다. 만일 다수의 VM 메타데이터 컨테이너들(403)에 대한 상기 생성 요청이 만족되지 않았다면, 알고리즘(800)은 단계(835)로 돌아갈 수 있고, 여기서 추가의 신규 VM 메타데이터 컨테이너들(403)을 생성할 수 있다. 일부 실시 예들에서, 상기 멀티-컨테이너(multi-container) 요청의 신규 VM 메타데이터 컨테이너들(403)을 생성하는 것 사이에, 상기 알고리즘은 단계(831)로 돌아갈 수 있고, 여기서 운영 체제(205)의 최소 자유 계획이 신규 생성된 VM 메타데이터 컨테이너(403)으로 상기 계획을 만족시키기에 적절한 레벨들에 도달했는지 또는 아닌지에 관한 결정이 내려진다. 만일 단계(835)에서 생성된 신규 VM 메타데이터 컨테이너(403)가 자유롭게 이용 가능한 자원의 양을 상기 최소 자유 계획의 임계값까지 올리지 않으면, 알고리즘(800)은 단계(835)로 돌아갈 수 있고, 여기서 상기 멀티-컨테이너 요청의 다음 신규 VM 메타데이터 컨테이너(403)을 생성할 수 있다. 만일, 단계(836)에서, 상기 멀티-컨테이너 요청이 완료되었다고 결정되면, 알고리즘(800)은 단계(837)로 진행하고 kproc는 슬립 모드로 돌아간다.

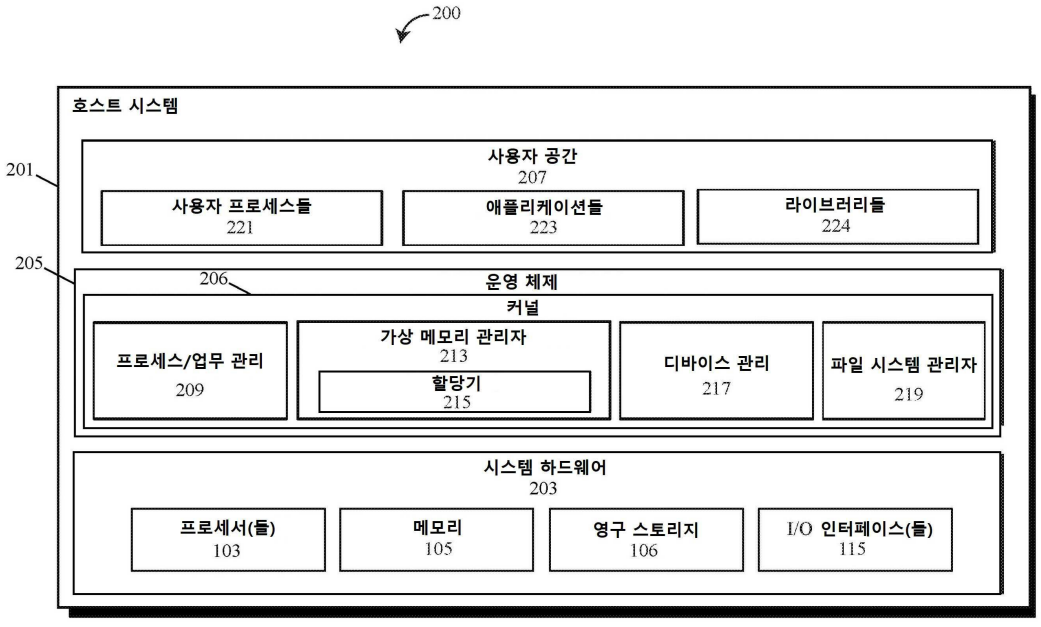
[0081] [0089] 본 발명의 다양한 실시 예들에 대한 설명은 예시의 목적으로 제공되는 것이며, 개시된 실시 예들이 전부라거나 이들에 한정하려는 의도가 있는 것은 아니다. 많은 수정들 및 변형들이 기술된 실시 예들의 범위와 정신을 벗어남이 없이 이 기술 분야에서 통상의 지식을 가진 자들에게 명백할 것이다. 여기서 사용된 용어들은 본 발명의 실시 예들의 원리들, 시장에서 발견된 기술들에 대한 실제 응용 또는 기술적 개선을 잘 설명하기 위해서, 또는 이 기술 분야에서 통상의 지식을 가진 자들이 여기서 기술된 실시 예들을 이해할 수 있도록 하기 위해, 선택되고 기술되었다.

도면

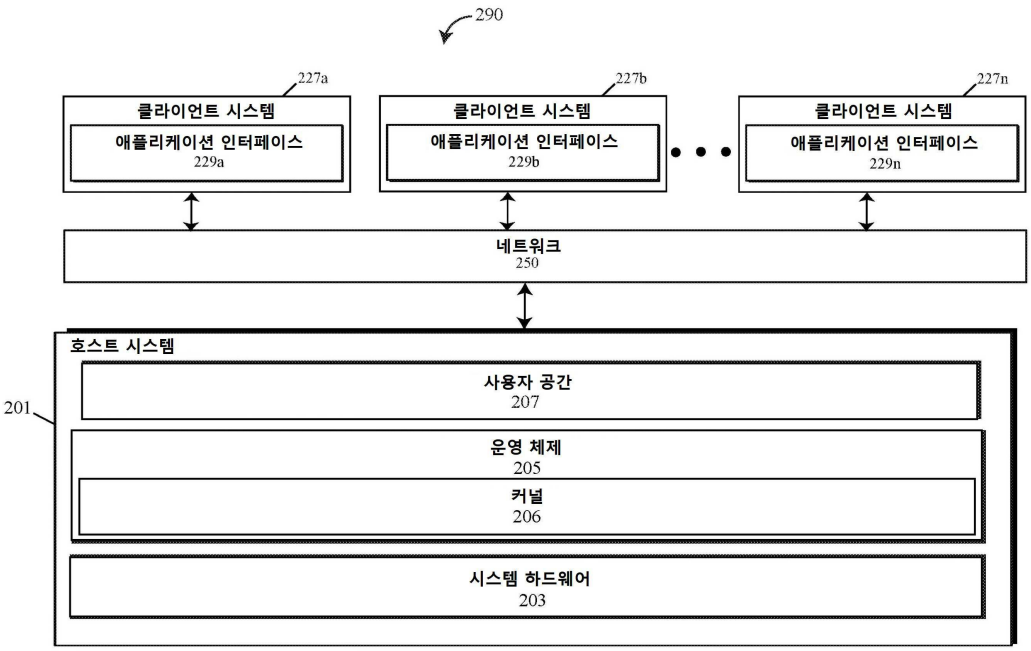
도면1


100
데이터 처리 시스템
프로세서(들)
103
메모리
105
106
캐시
107
112
영구 스토리지
프로그램(들)
114
디스플레이
118
I/O
인터페이스(들)
115
통신 유닛
111
외부 디바이스(들)
117

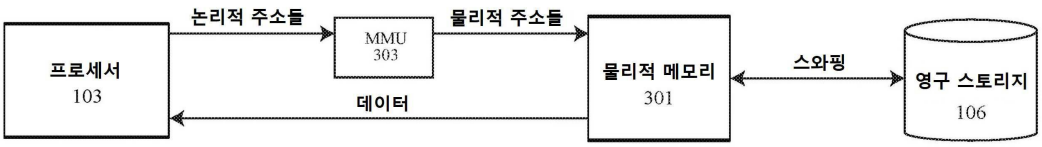
도면2a


200
호스트 시스템
201
사용자 공간
207
사용자 프로세스들
221
애플리케이션들
223
라이브러리들
224
205
206
운영 체제
커널
프로세스/업무 관리
209
가상 메모리 관리자
213
할당기
215
디바이스 관리
217
파일 시스템 관리자
219
시스템 하드웨어
203
프로세서(들)
103
메모리
105
영구 스토리지
106
I/O 인터페이스(들)
115

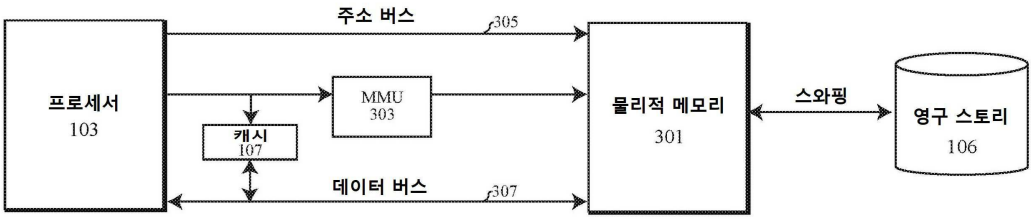
도면2b



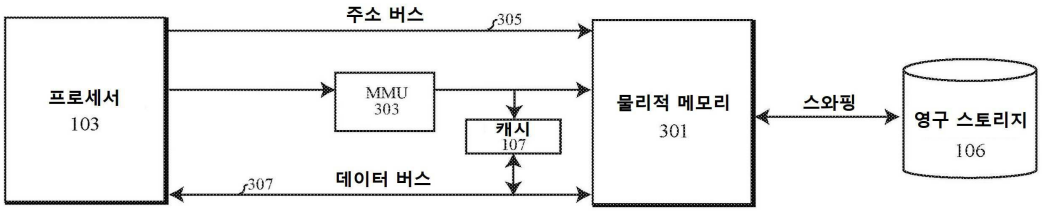

도면3a



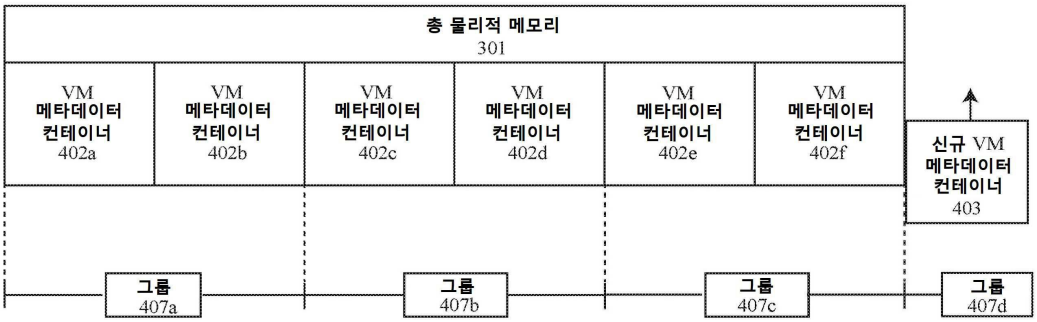

도면3b



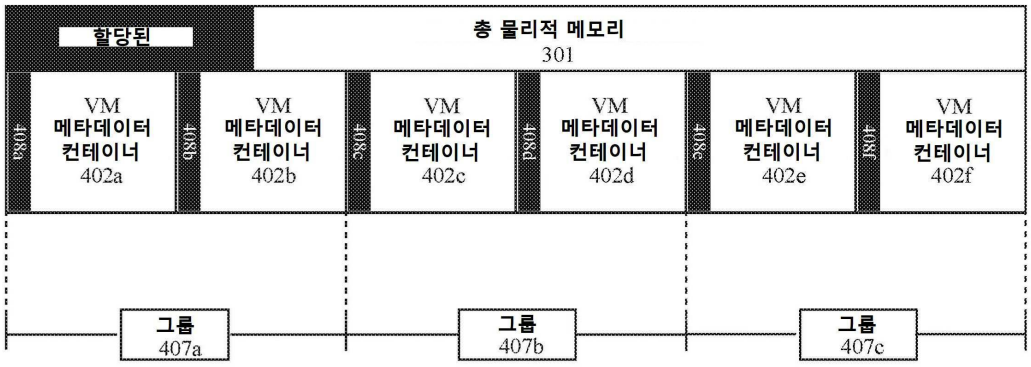

도면3c



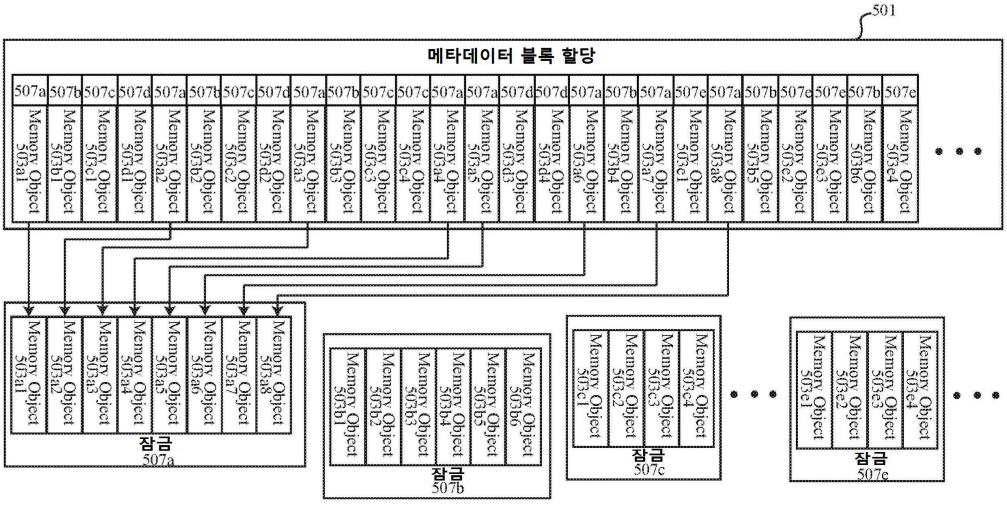
도면4a



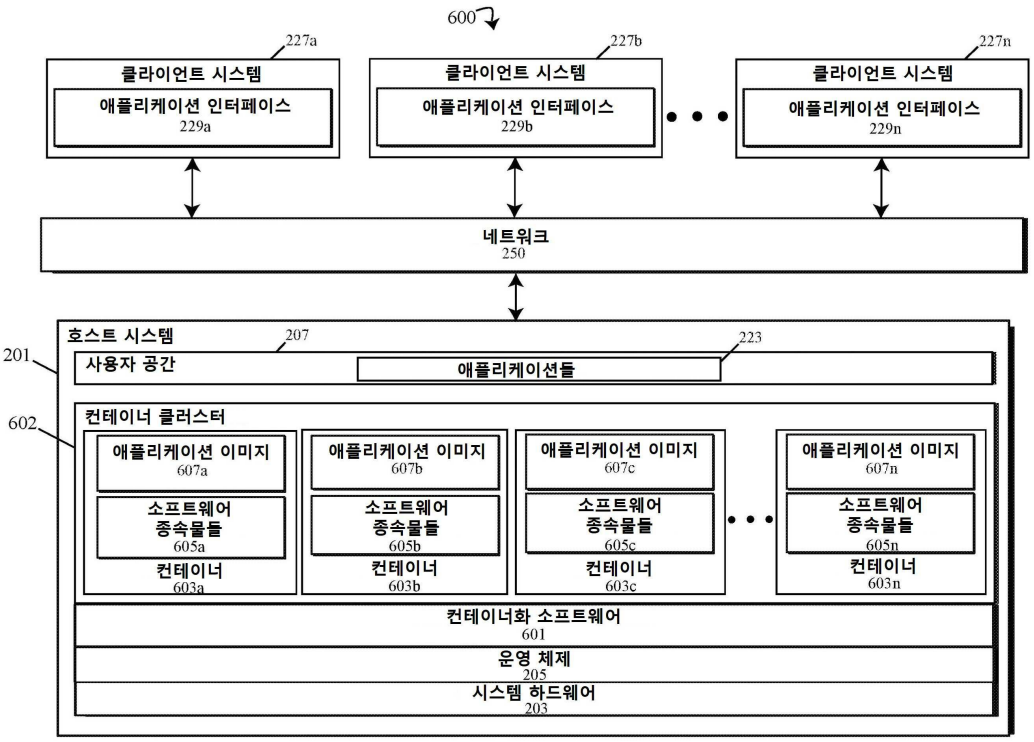

도면4b



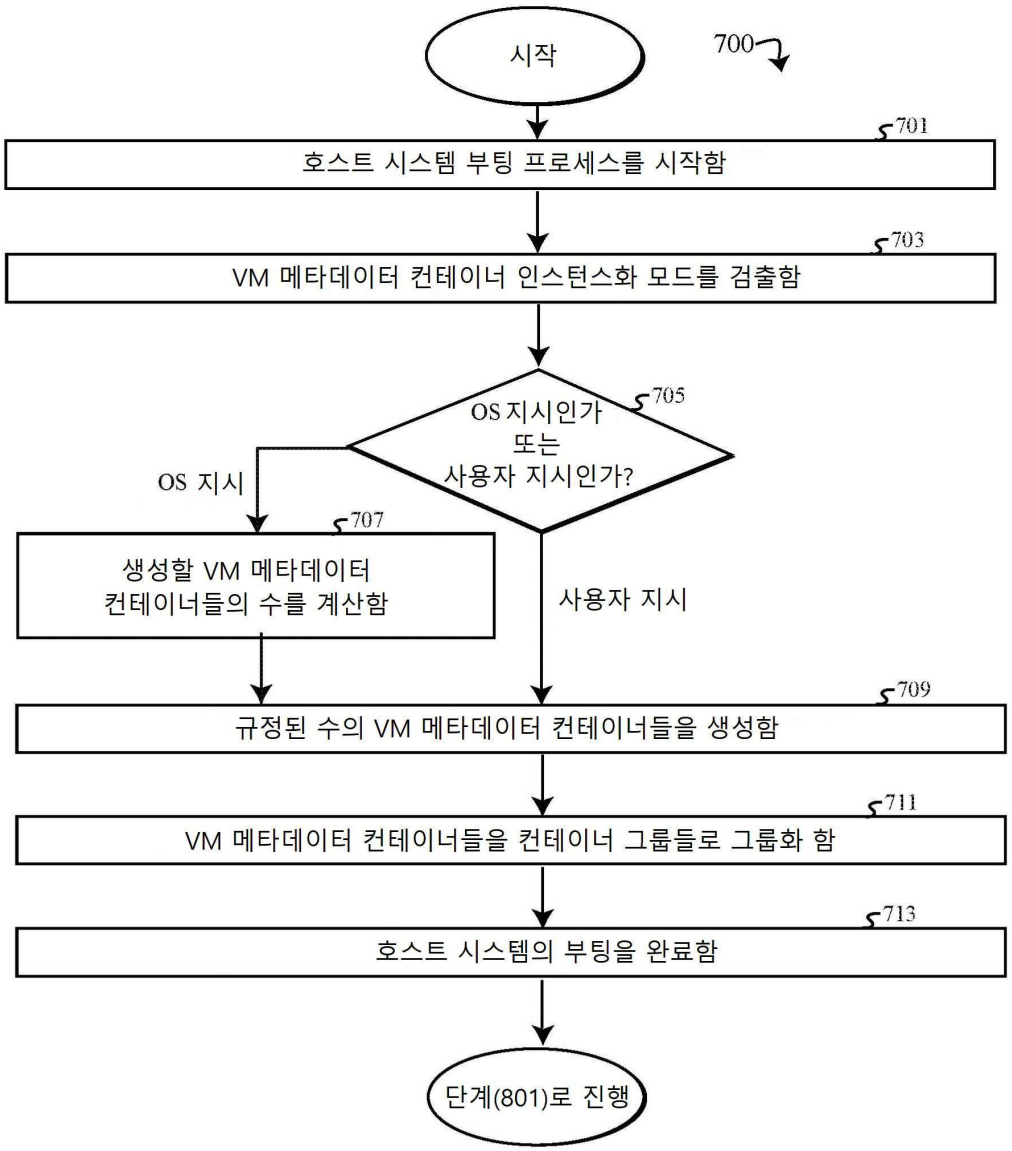

도면5



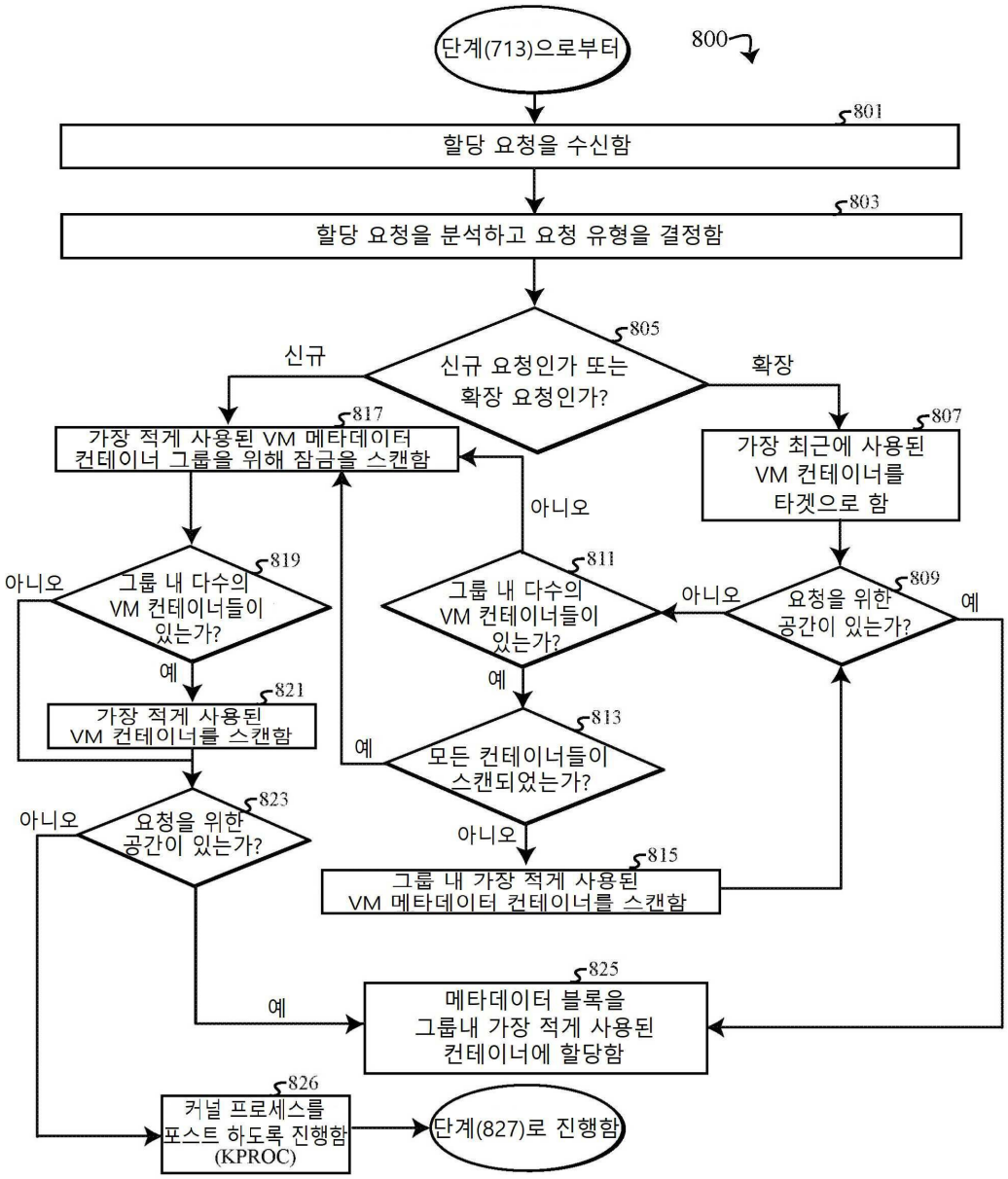
도면6


600
227a
클라이언트 시스템
애플리케이션 인터페이스
229a
227b
클라이언트 시스템
애플리케이션 인터페이스
229b
227n
클라이언트 시스템
애플리케이션 인터페이스
229n
네트워크
250
호스트 시스템
207
223
201
사용자 공간
애플리케이션들
602
컨테이너 클러스터
애플리케이션 이미지
607a
소프트웨어 종속물들
605a
컨테이너
603a
애플리케이션 이미지
607b
소프트웨어 종속물들
605b
컨테이너
603b
애플리케이션 이미지
607c
소프트웨어 종속물들
605c
컨테이너
603c
애플리케이션 이미지
607n
소프트웨어 종속물들
605n
컨테이너
603n
컨테이너화 소프트웨어
601
운영 체제
205
시스템 하드웨어
203

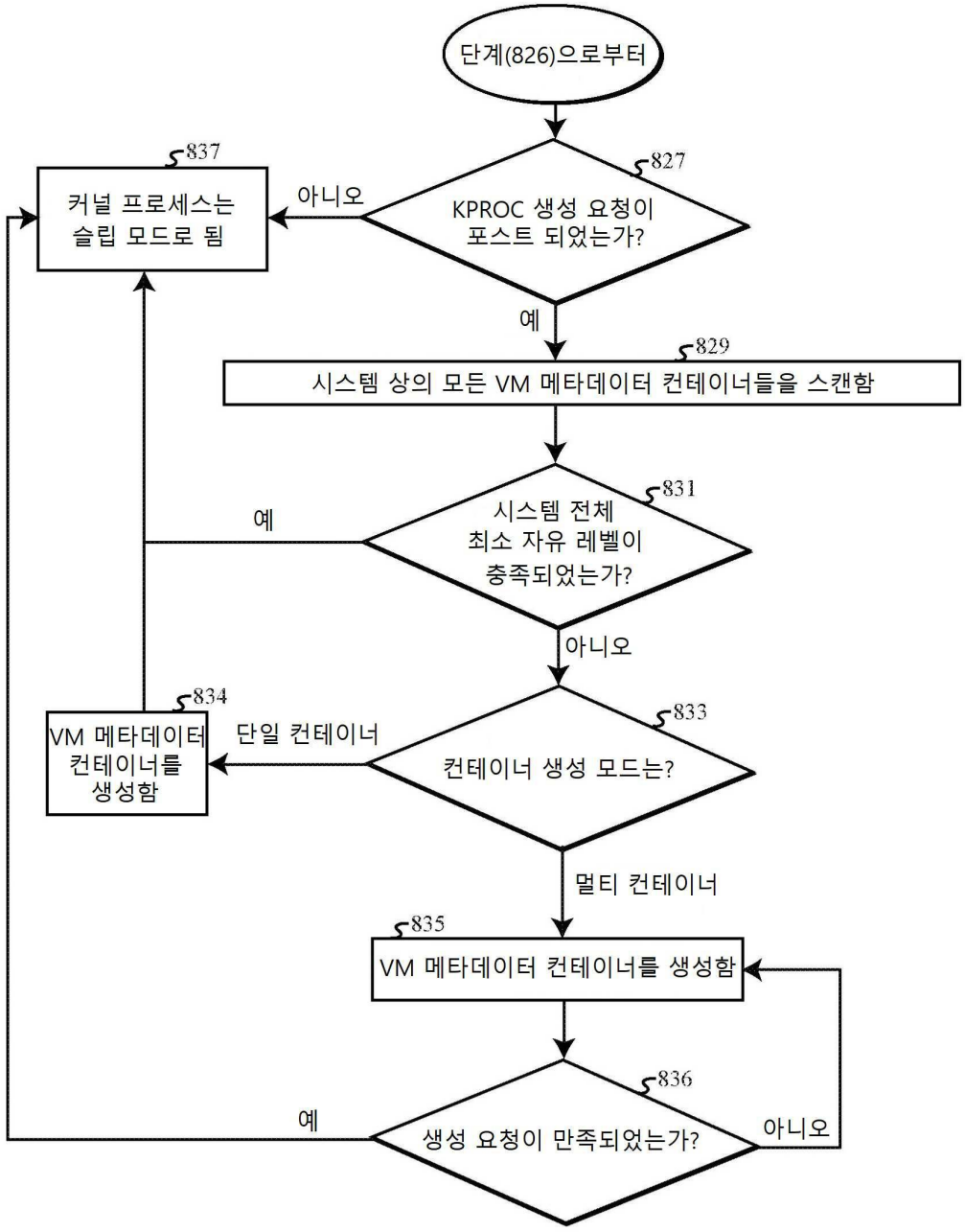
도면7


시작
700
701
호스트 시스템 부팅 프로세스를 시작함
703
VM 메타데이터 컨테이너 인스턴스화 모드를 검출함
705
OS 지시인가
또는
사용자 지시인가?
OS 지시
707
생성할 VM 메타데이터
컨테이너들의 수를 계산함
사용자 지시
709
규정된 수의 VM 메타데이터 컨테이너들을 생성함
711
VM 메타데이터 컨테이너들을 컨테이너 그룹들로 그룹화 함
713
호스트 시스템의 부팅을 완료함
단계(801)로 진행

도면8a


단계(713)으로부터
800
801
할당 요청을 수신함
803
할당 요청을 분석하고 요청 유형을 결정함
805
신규 요청인가 또는 확장 요청인가?
신규
확장
817
가장 적게 사용된 VM 메타데이터 컨테이너 그룹을 위해 잠금을 스캔함
807
가장 최근에 사용된 VM 컨테이너를 타겟으로 함
아니오
819
그룹 내 다수의 VM 컨테이너들이 있는가?
811
그룹 내 다수의 VM 컨테이너들이 있는가?
809
요청을 위한 공간이 있는가?
아니오
예
예
아니오
예
821
가장 적게 사용된 VM 컨테이너를 스캔함
813
모든 컨테이너들이 스캔되었는가?
예
아니오
823
요청을 위한 공간이 있는가?
아니오
815
그룹 내 가장 적게 사용된 VM 메타데이터 컨테이너를 스캔함
예
825
메타데이터 블록을 그룹내 가장 적게 사용된 컨테이너에 할당함
826
커널 프로세스를 포스트 하도록 진행함 (KPROC)
단계(827)로 진행함

도면8b




【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 청구항 25

【변경전】

제 24항에 있어서, 상기 방법은:

상기 컴퓨터 시스템의 상태를 쿼리하는 단계;

상기 컴퓨터 시스템에 의한 사용을 위해 이용 가능한 컴퓨팅 자원들의 최소 양을 유지하기 위해 상기 컴퓨터 시스템의 상태를 운영 체제의 시스템 관리 계획과, 상기 프로세서에 의해서, 비교하는 단계;

상기 컴퓨터 시스템의 상태를 운영 체제의 시스템 관리 계획과 비교하여, 이용 가능한 컴퓨팅 자원들의 최소 양을 유지하는 것이 충족되지 않았다고 결정하는 것에 응답하여, 상기 하나 또는 그 이상의 신규 VM 메타데이터

컨테이너들을 인스턴스화하는 단계를 더 포함하는

컴퓨터-판독가능 스토리지 매체.

【변경후】

제 24항에 있어서, 상기 방법은:

상기 컴퓨터 시스템의 상태를 쿼리하는 단계;

상기 컴퓨터 시스템에 의한 사용을 위해 이용 가능한 컴퓨팅 자원들의 최소 양을 유지하기 위해 상기 컴퓨터 시스템의 상태를 운영 체제의 시스템 관리 계획과, 프로세서에 의해서, 비교하는 단계;

상기 컴퓨터 시스템의 상태를 운영 체제의 시스템 관리 계획과 비교하여, 이용 가능한 컴퓨팅 자원들의 최소 양을 유지하는 것이 충족되지 않았다고 결정하는 것에 응답하여, 상기 하나 또는 그 이상의 신규 VM 메타데이터 컨테이너들을 인스턴스화하는 단계를 더 포함하는

컴퓨터-판독가능 스토리지 매체.