(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2009-0024256
(43) 공개일자 2009년03월06일

(51) Int. Cl.
G06F 9/22 (2006.01) *G06F 9/50* (2006.01)
G06F 9/46 (2006.01) *G06F 9/38* (2006.01)
(21) 출원번호 10-2009-7000686
(22) 출원일자 2009년01월13일
심사청구일자 없음
번역문제출일자 2009년01월13일
(86) 국제출원번호 PCT/US2007/012746
국제출원일자 2007년05월29일
(87) 국제공개번호 WO 2007/149188
국제공개일자 2007년12월27일
(30) 우선권주장
11/471,466 2006년06월19일 미국(US)

(71) 출원인
디스키퍼 코포레이션
미국 캘리포니아주 91504 버뱅크 노쓰 글레녹스 불바드 7590
(72) 발명자
젠센 크래이그
미국 캘리포니아주 91011 라 캐나다 메사 비스타 드라이브 4245
스태퍼 앤드류
미국 캘리포니아주 91342 사일마 알타 비스타 웨이 13270
토마스 바실
미국 캘리포니아주 91342 사일마 윌로우그린 레인 14537
(74) 대리인
김태홍, 신정건

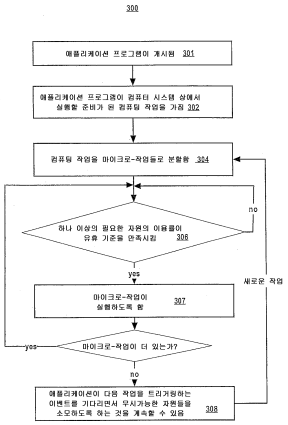
전체 청구항 수 : 총 41 항

(54) 비파괴 시간에 실행하도록 컴퓨터 마이크로-작업을 스케쥴링하는 방법, 시스템 및 장치

(57) 요 약

컴퓨팅 작업을 마이크로-작업들로 분할하고 마이크로-작업들의 실행을, 필요한 자원이 하나 이상의 유휴 기준에 따를 때의 시간들에 할당하는 방법, 시스템 및 장치가 제공된다. 마이크로-작업들은 진행 과정에 맞춰 실행되지만, 마이크로-작업들에 의해 필요한 자원들이 다른 작업에 의해 필요하지 않을 때에만 실행된다. 이러한 방법론을 이용하는 소프트웨어 프로그램은 컴퓨터가 파워업되는 동안 동일한 컴퓨터 시스템 상에서 실행하는 다른 소프트웨어 프로그램들의 성능에 충격을 주지 않고 항상 실행될 수 있다.

대 표 도 - 도3



특허청구의 범위

청구항 1

복수의 마이크로-작업들 각각에 대하여,

특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따르는 때를 판단하는 단계와;

특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따른다는 판단에 적어도 부분적으로 기초하여 특정 마이크로-작업이 실행되도록 하는 단계

를 포함하는 머신 구현 방법.

청구항 2

제1항에 있어서, 컴퓨팅 작업을 복수의 마이크로-작업들로 분할하는 단계를 더 포함하는 머신 구현 방법.

청구항 3

제2항에 있어서, 컴퓨팅 작업을 복수의 마이크로-작업들로 분할하는 단계는, 컴퓨팅 작업을 개시하는 애플리케이션 프로그램에 의해 수행되는 것인 머신 구현 방법.

청구항 4

제2항에 있어서, 컴퓨팅 작업을 복수의 마이크로-작업들로 분할하는 단계는, 컴퓨팅 작업을 개시하는 애플리케이션 프로그램 외의 소프트웨어 프로세스에 의해 수행되는 것인 머신 구현 방법.

청구항 5

제1항에 있어서, 특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따르는 때를 판단하는 단계는, 오퍼레이팅 시스템에 의해 수행되는 것인 머신 구현 방법.

청구항 6

제1항에 있어서, 특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따르는 때를 판단하는 단계는, 오퍼레이팅 시스템의 외부에 있는 마이크로-작업 스케쥴러에 의해 수행되는 것인 머신 구현 방법.

청구항 7

제1항에 있어서, 각각의 마이크로-작업의 실행 사이를 기다리는 최소량의 시간을 특정하는 대기 시간에 기초하여 마이크로-작업들 중 적어도 하나의 마이크로-작업의 실행을 지연시키는 단계를 더 포함하는 머신 구현 방법.

청구항 8

제7항에 있어서, 애플리케이션 프로그램이 대기 시간을 특정하는 단계를 더 포함하는 머신 구현 방법.

청구항 9

제8항에 있어서, 상기 대기 시간을 특정하는 단계는, 마이크로-작업들 중 서로 다른 작업들에 대한 서로 다른 최소량의 시간을 특정하는 단계를 포함하는 것인 머신 구현 방법.

청구항 10

제1항에 있어서, 상기 하나 이상의 유휴 기준은 하나 이상의 자원 이용률 임계값에 기초하는 것인 머신 구현 방법.

청구항 11

제10항에 있어서, 상기 하나 이상의 자원 이용률 임계값을 특정하는 단계를 더 포함하는 머신 구현 방법.

청구항 12

제10항에 있어서, 마이크로-작업들 중 서로 다른 임계값들에 대한 서로 다른 자원 이용률 임계값을 특정하는 단계를 더 포함하는 머신 구현 방법.

청구항 13

제1항에 있어서, 상기 특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따른다는 판단에 적어도 부분적으로 기초하여 특정 마이크로-작업이 실행되도록 하는 단계는, 다른 작업들이 컴퓨터 자원들을 이용할 수 있기 전에 실행하도록 허용된 마이크로-작업들의 수를 제한하는 것에 추가로 기초하는 것인 머신 구현 방법.

청구항 14

제13항에 있어서, 상기 실행하도록 허용된 마이크로-작업들의 수를 제한하는 것은 애플리케이션 프로그램 외의 프로세스에 의해 컴퓨터 자원 이용률을 검사함으로써 판단되는 것인 머신 구현 방법.

청구항 15

제1항에 있어서, 컴퓨팅 작업을 개시하는 애플리케이션에 할당하는 메모리의 양을 판단하는 단계를 더 포함하며,

상기 메모리의 양을 판단하는 단계는, 전체적으로 애플리케이션에 대한 메모리 요구량보다는 애플리케이션의 마이크로-작업들을 실행하는데 필요한 보다 더 작은 메모리 양에 기초하는 것인 머신 구현 방법.

청구항 16

제15항에 있어서, 상기 메모리의 양을 판단하는 단계는, 컴퓨터 시스템에 이용가능한 메모리 양에 추가로 기초하는 것인 머신 구현 방법.

청구항 17

제1항에 있어서, 상기 마이크로-작업들은 컴퓨터 프로세스의 일부분이며, 각각의 마이크로-작업의 실행은 나머지의 컴퓨터 프로세스의 결과에 영향을 주지 않고 지연될 수 있는 것인 머신 구현 방법.

청구항 18

하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금, 복수의 마이크로-작업들 각각에 대하여,

특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따르는 때를 판단하는 단계와;

특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따른다는 판단에 적어도 부분적으로 기초하여 특정 마이크로-작업이 실행되도록 하는 단계

를 수행하도록 하는 명령의 하나 이상의 시퀀스를 전달하는 것인 컴퓨터 판독가능 기록 매체.

청구항 19

제18항에 있어서, 상기 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금 컴퓨팅 작업을 복수의 마이크로-작업들로 분할하는 단계를 수행하도록 하는 명령을 더 포함하는 컴퓨터 판독가능 기록 매체.

청구항 20

제18항에 있어서, 상기 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금, 각각의 마이크로-작업의 실행 사이를 기다리는 최소량의 시간을 특정하는 대기 시간에 기초하여 마이크로-작업들 중 적어도 하나의 마이크로-작업의 실행을 지연시키는 단계를 수행하도록 하는 명령을 더 포함하는 컴퓨터 판독가능 기록 매체.

청구항 21

제20항에 있어서, 각각의 마이크로-작업의 실행 사이를 기다리는 최소량의 시간을 특정하는 대기 시간에 기초하여 마이크로-작업들 중 적어도 하나의 마이크로-작업의 실행을 지연시키는 단계를 수행하는 명령은 마이크로-작업들 중 서로 다른 작업들에 대한 서로 다른 최소량의 시간을 특정하는 단계를 수행하는 명령을 포함하는 것인 컴퓨터 판독가능 기록 매체.

청구항 22

제18항에 있어서, 특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따르는 때를 판단하는 단계를 수행하는 명령은, 특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 자원 이용률 임계값에 기초하여 하나 이상의 유휴 기준에 따르는 때를 판단하는 단계를 수행하는 명령을 포함하는 것인 컴퓨터 판독가능 기록 매체.

청구항 23

제22항에 있어서, 상기 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금 하나 이상의 자원 이용률 임계값을 특정하는 단계를 수행하도록 하는 명령을 더 포함하는 컴퓨터 판독가능 기록 매체.

청구항 24

제22항에 있어서, 상기 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금 마이크로-작업들 중 서로 다른 마이크로-작업들에 대한 서로 다른 자원 이용률 임계값을 특정하는 단계를 수행하도록 하는 명령을 더 포함하는 컴퓨터 판독가능 기록 매체.

청구항 25

제18항에 있어서, 상기 특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따른다는 판단에 적어도 부분적으로 기초하여 특정 마이크로-작업이 실행되도록 하는 단계를 수행하는 명령은, 다른 작업들이 컴퓨터 자원들을 이용할 수 있기 전에 실행하도록 허용된 마이크로-작업들의 수를 제한하는 것에 추가로 기초하여 특정 마이크로-작업이 실행되도록 하는 단계를 수행하는 명령을 포함하는 것인 컴퓨터 판독가능 기록 매체.

청구항 26

제25항에 있어서, 상기 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금, 애플리케이션 외의 프로세스에 의해 컴퓨터 자원 이용률을 검사함으로써 다른 작업들이 컴퓨터 자원들을 이용할 수 있기 전에 실행하도록 허용된 마이크로-작업들의 수를 제한하는 단계를 수행하도록 하는 명령을 더 포함하는 컴퓨터 판독가능 기록 매체.

청구항 27

제18항에 있어서, 상기 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금 애플리케이션에 할당하는 메모리의 양을 판단하는 단계를 수행하도록 하는 명령을 더 포함하며,

상기 메모리의 양을 판단하는 단계는, 애플리케이션의 마이크로-작업들을 실행하기 위한 최소 메모리 요구량에 기초하는 것인 컴퓨터 판독가능 기록 매체.

청구항 28

제27항에 있어서, 상기 메모리의 양을 판단하는 단계를 수행하는 명령은, 컴퓨터 시스템에 이용가능한 메모리의 양에 기초하여 메모리의 양을 판단하는 단계를 수행하는 명령을 더 포함하는 것인 컴퓨터 판독가능 기록 매체.

청구항 29

제18항에 있어서, 상기 마이크로-작업들은 컴퓨터 프로세스의 일부분이며, 각각의 마이크로-작업의 실행은 나머지의 컴퓨터 프로세스의 결과에 영향을 주지 않고 지연될 수 있는 것인 컴퓨터 판독가능 기록 매체.

청구항 30

하나 이상의 프로세서와; 상기 하나 이상의 프로세서에 통신가능하게 결합된 컴퓨터 판독가능 기록 매체를 포함하는 시스템으로서,

상기 컴퓨터 판독가능 기록 매체는, 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금, 복수의 마이크로-작업들 각각에 대하여,

특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따르는 때를 판단하는 단계와;

특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따른다는 판단에 적어도 부분적으로 기초하여 특정 마이크로-작업이 실행되도록 하는 단계

를 수행하도록 하는 명령의 하나 이상의 저장된 시퀀스를 상부에 저장한 것인 시스템.

청구항 31

제30항에 있어서, 상기 컴퓨터 판독가능 기록 매체는, 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금, 컴퓨팅 작업을 복수의 마이크로-작업들로 분할하는 단계를 수행하도록 하는 명령을 더 포함하는 것인 시스템.

청구항 32

제30항에 있어서, 상기 컴퓨터 판독가능 기록 매체는, 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금, 각각의 마이크로-작업의 실행 사이를 기다리는 최소량의 시간을 특정하는 대기 시간에 기초하여 마이크로-작업들 중 적어도 하나의 마이크로-작업의 실행을 지연시키는 단계를 수행하도록 하는 명령을 더 포함하는 것인 시스템.

청구항 33

제32항에 있어서, 각각의 마이크로-작업의 실행 사이를 기다리는 최소량의 시간을 특정하는 대기 시간에 기초하여 마이크로-작업들 중 적어도 하나의 마이크로-작업의 실행을 지연시키는 단계를 수행하는 명령은 마이크로-작업들 중 서로 다른 작업들에 대한 서로 다른 최소량의 시간을 특정하는 단계를 수행하는 명령을 포함하는 것인 시스템.

청구항 34

제30항에 있어서, 특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따르는 때를 판단하는 단계를 수행하는 명령은, 특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 자원 이용률 임계값에 기초하여 하나 이상의 유휴 기준에 따르는 때를 판단하는 단계를 수행하는 명령을 포함하는 것인 시스템.

청구항 35

제34항에 있어서, 상기 컴퓨터 판독가능 기록 매체는, 상기 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금 하나 이상의 자원 이용률 임계값을 특정하는 단계를 수행하도록 하는 명령을 더 포함하는 것인 시스템.

청구항 36

제34항에 있어서, 상기 컴퓨터 판독가능 기록 매체는, 상기 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금 마이크로-작업들 중 서로 다른 마이크로-작업들에 대한 서로 다른 자원 이용률 임계값을 특정하는 단계를 수행하도록 하는 명령을 더 포함하는 것인 시스템.

청구항 37

제30항에 있어서, 상기 특정 마이크로-작업들을 실행하는데 필요한 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준에 따른다는 판단에 적어도 부분적으로 기초하여 특정 마이크로-작업이 실행되도록 하는 단계를 수행하는 명령은, 다른 작업들이 컴퓨터 자원들을 이용할 수 있기 전에 실행하도록 허용된 마이크로-작업들의 수를 제한하는 것에 추가로 기초하여 특정 마이크로-작업이 실행되도록 하는 단계를 수행하는 명령을 포함하는 것인

시스템.

청구항 38

제37항에 있어서, 상기 컴퓨터 판독가능 기록 매체는, 상기 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금, 애플리케이션 외의 프로세스에 의해 컴퓨터 자원 이용률을 검사함으로써 다른 작업들이 컴퓨터 자원들을 이용할 수 있기 전에 실행하도록 허용된 마이크로-작업들의 수를 제한하는 단계를 수행하도록 하는 명령을 더 포함하는 것인 시스템.

청구항 39

제30항에 있어서, 상기 컴퓨터 판독가능 기록 매체는, 상기 하나 이상의 프로세서에 의해 실행될 때 하나 이상의 프로세서로 하여금, 애플리케이션에 할당하는 메모리의 양을 판단하는 단계를 수행하도록 하는 명령을 더 포함하며,

상기 메모리의 양을 판단하는 단계는, 애플리케이션의 마이크로-작업들을 실행하기 위한 최소 메모리 요구량에 기초하는 것인 시스템.

청구항 40

제39항에 있어서, 상기 메모리의 양을 판단하는 단계를 수행하는 명령은, 컴퓨터 시스템에 이용가능한 메모리의 양에 기초하여 메모리의 양을 판단하는 단계를 수행하는 명령을 더 포함하는 것인 시스템.

청구항 41

제30항에 있어서, 상기 마이크로-작업들은 컴퓨터 프로세스의 일부분이며, 각각의 마이크로-작업의 실행은 나머지의 컴퓨터 프로세스의 결과에 영향을 주지 않고 지연될 수 있는 것인 시스템.

명 세 서

기 술 분 야

<1> 본 발명은 컴퓨팅 환경에서 소프트웨어 애플리케이션들을 실행하는 것에 관한 것이다. 보다 자세하게는, 본 발명의 실시예들은 마이크로-작업들의 실행이 다른 애플리케이션들 및 작업들의 실행에 상당히 충격을 주지 않도록, 애플리케이션의 컴퓨팅 작업 또는 입력-출력 작업을 마이크로-작업들로 분할하고 낮은 컴퓨터 자원 이용의 기간들(period)에 마이크로-작업들의 실행을 할당하는 것에 관한 것이다.

배 경 기 술

<2> 메인프레임 컴퓨터의 초기에는, 작업들의 묶음들로 소프트웨어 프로그램들을 실행시키는 개념이 표준이였다. 컴퓨터의 수가 제한되었기 때문에, 사용자들은 컴퓨터가 일부 다른 더 중요한 작업들에 이용되지 않았을 때 컴퓨터 상에서 실행하도록 자신들의 작업(들)을 스케쥴링해야 했다. 이러한 시스템에서, 각각의 작업은 다음 작업에 이어지고 그 후 그 다음 작업이 이어져 중단 없이 완료될 때까지 실행하도록 스케쥴링되었다. 제한된 이용가능 컴퓨터 시간은 더 높은 우선 순위 애플리케이션들을 지연시키지 않도록 더 낮은 우선 순위 작업들을 "비번시(off-hours)"에 실행시킬 것을 요한다.

<3> 보다 최근에는, 멀티-태스킹(multi-tasking) 컴퓨터 시스템들이 단일의 CPU에 의해 2이상의 작업들의 동시 실행 또는 인터리브된 실행을 가능하게 하였다. 멀티-태스킹 컴퓨터 시스템은 많은 애플리케이션들이 동일한 일반 기간에 실행할 수 있도록 한다. 통상적으로, 멀티-태스킹 시스템들은 복잡한 내부 스케쥴링 알고리즘들을 갖고 있으며, 여기서, 프로세스들이 할당된 우선순위들에 따라 스케쥴링된다. 그러나, 이 애플리케이션들은 여전히 컴퓨팅 자원들에 대해 경합한다. 자원 경합을 완화시키기 위해, 멀티-태스킹 시스템에서의 애플리케이션이 오퍼레이터 스케쥴링에 맞춰 "비번시"에 런오프될 수 있다.

<4> 비번시에 실행되는 애플리케이션들은 백업, 인덱싱, 소프트웨어 업데이트, 바이러스 및 악성 소프트웨어 스캔 및 조각 모으기와 같은 메인터넌스 작업들을 포함할 수 있다. 비번시 프로세싱을 위한 후보들은 또한 리포트들을 실행시키고 금융 계산들을 수행하는 등의 소프트웨어 애플리케이션들을 포함할 수 있다. 그러나, 인덱서들과 같은 일부 애플리케이션들은 생산 시간 동안에 실행되어야 한다. 따라서, 모든 애플리케이션들이 비번시 실행을 위한 양호한 후보인 것은 아니다.

<5> 비번시에 실행하도록 작업을 스케쥴링하는데 있어 다른 문제는 작업이 실행하도록 스케쥴링된 시간에 컴퓨터가 꺼질 수 있다는 점이다. 추가의 문제는, 일부 머신들이 명확하게 식별된 비번시간을 갖지 못한다는 점이다. 예를 들어, 많은 컴퓨터 시스템들은 컴퓨팅 액티비티가 중요한 기간 동안에 인터럽트되지 않아야 할 정도로 충분히 중요하게 여겨지는 컴퓨팅 액티비티를 위해 하루에 24시간 동안 이용된다. 따라서, 어느 작업을 스케쥴링하는데 있어 "비번시간"이 없다. 다른 추가의 문제는, 일반적으로 작업이 비번시 컴퓨팅을 위해 스케쥴링되어야 할 때를 사용자가 결정해야 한다는 점이다. 따라서, 스케쥴을 설정하는 것은 사용자의 시간을 차지하고 사용자가 에러를 범하기 쉽다.

<6> 이전에 설명된 바와 같이, 컴퓨팅 작업을 실행하는 것은 컴퓨터를 이용하는 사용자의 능력과 상충할 수 있고 가능한 더 긴급한 다른 애플리케이션들 및 작업들로부터 자원을 가져올 수 있다. 스로틀링(Throttling)은 이들 부정적인 충격을 최소화하는 기술이다. 스로틀링은 애플리케이션 또는 작업이 할당된 양의 자원들 보다 더 많이 이용하는 것을 방지한다. 스로틀링의 유형들은 디스크 I/O 스로틀링, CPU 스로틀링 및 네트워크 스로틀링을 포함한다. 예를 들어, CPU 스로틀링은 애플리케이션에 대한 목표 CPU 이용 한계값을 설정하고, 애플리케이션들이 목표 한계값을 초과하는 경우에 애플리케이션이 작업하는 것을 중지하도록 하는 것을 포함할 수 있다. 스로틀링은 메인터넌스 애플리케이션들 또는 보다 덜 중요한 컴퓨팅 작업들을 위한 컴퓨터 자원들에 때때로 적용된다. 스로틀링이 이점을 갖고 있지만, 컴퓨팅 작업의 자원 이용은 다른 작업들 및 애플리케이션들에 대하여 전체적으로 투명하지 못하다.

<7> 동시에, 심지어 긴급한 최상 우선순위 작업의 처리 동안에도 상당한 컴퓨팅 자원들이 이용하지 않게 되는 것도 주목할만하다. CPU들, 메모리, 디스크 드라이브 및 네트워크들의 속도에서의 큰 차이는 일반적으로 이들 컴포넌트 중 하나 이상의 컴포넌트를 유휴상태로 놓여지게 하는 한편, 다른 컴포넌트들 중 하나는 완전하게 소모된다. 예를 들어, 3GHz CPU는 종종 밀리초 단위로 측정된 평균 액세스 시간에서 데이터를 검색하는 디스크 드라이브를 대기하는 동안 유휴 상태에 놓여있다.

<8> 그 외에 손실된 이들의 자원을 복구하여 이용하기 위해 필요한 것은 다른 작업들 또는 애플리케이션들에 상당히 충격을 주지 않고 컴퓨터 시스템에서 하나 이상의 작업들이 실행할 수 있도록 하는 기술이다. 이 기술은 작업을 스케쥴링하는데 있어 사용자의 시간을 소모하지 않아야 하거나 또는 작업이 실행중일 때 컴퓨터 시스템과의 사용자의 상호작용에 부정적으로 충격을 주지 않아야 한다. 이 기술은 비번시에 실행하도록 작업을 스케쥴링하는 것을 필요로 하지 않아야 한다. 이 기술은 비번시간을 갖지 않는 컴퓨터 시스템에 유리해야 하고 컴퓨터 시스템에 의해 이용가능해야 한다.

<9> 이 섹션에서 설명된 이 접근 방식들은 추구될 수 있는 접근 방식들이지만, 반드시 이전에 인식되거나 또는 추구되었던 접근방식일 필요가 있는 것은 아니다. 따라서, 달리 나타내어지지 않았다면, 이 섹션에서 설명된 접근 방식들 중 어느 것도 이 섹션에서의 접근 방식들의 포함에 의해 단지 종래 기술로서 간주되는 것으로서 추정되지 않아야 한다.

발명의 상세한 설명

<10> 마이크로 작업을 실행시키는데 필요한 자원이 유휴 상태에 있을 때 컴퓨팅 작업을 마이크로-작업들로 분할하고 마이크로-작업들을 실행시킴으로써 이들의 이용되지 않는 컴퓨터 자원들을 이용하기 위한 방법, 시스템 및 장치가 개시된다.

실 시 예

<17> 다음 설명에서, 본 발명의 철저한 이해를 제공하기 위해 설명의 목적으로 다수의 특정 세부 내용이 설명된다. 그러나, 본 발명은 이들 특정 세부 내용 없이도 실시될 수 있음이 명백하다. 다른 경우에, 본 발명이 불필요하게 모호해지는 것을 피하기 위해 잘 알려진 구조 및 장치들이 블록도로 도시되어 있다.

<18> 개요

<19> 대부분의 컴퓨터들은 자신의 시간의 자원 용량 100% 모두를 이용하는 것은 아니다. 이것은 매일의 일부분 동안에만 켜져있는 컴퓨터 뿐만 아니라 하루에 24 시간, 일주일에 7일을 끊김없이 높게 이용하는 컴퓨터 조차도 실제 그러하다. 따라서, 컴퓨터 시간 및 자원이 낭비된다. 예를 들어, 24시간 기간에 걸쳐 매우 심하게 이용되고 액티비티시 잠깐의 스파이크를 가질 수 있는 컴퓨터 시스템은 자신의 자원의 약 5% 내지 20%만을 평균적으로 이용할 수 있다.

<20> 마이크로 작업을 실행시키는데 필요한 자원이 유휴 상태에 있을 때 컴퓨팅 작업을 마이크로-작업들로 분할하고 마이크로-작업들을 실행시킴으로써 이들의 이용되지 않는 컴퓨터 자원들을 이용하기 위한 방법, 시스템 및 장치가 여기에 개시된다. 여기에서 이용될 때, 용어, 마이크로-작업은 나머지 프로세스의 결과에 영향을 주지 않고 그 실행이 지연될 수 있는 컴퓨터 프로세스의 일부분이다. 여기에 이용될 때, "유휴 자원", "유휴 시간" 등은 자원이 100%보다 작게 이용될 때의 시간, 즉, 이용되고 있지 않는 자원의 일부가, 자원의 일부 다른 부분이 이용되는 경우에도, "유휴 상태"인 것으로 간주될 때의 시간을 의미한다.

<21> 따라서, 마이크로-작업들은 진행 과정에 맞춰 실행되지만, 가능한 빨리 완료하는 컴퓨팅 작업을 얻기 위한 시도 없이 실행된다. 이러한 방법론을 이용하는 소프트웨어 프로그램은, 동일한 컴퓨터 상에서 동시에 실행하는 다른 소프트웨어 프로그램들의 성능에 거의 무시할만한 충격으로 컴퓨터가 파워업된 상태에서 항상 실행될 수 있어, 상당히 더 유용한 작업이 시간 단위 당 행해지도록 한다.

<22> 다른 작업들 및 애플리케이션들에 상당히 충격을 주는 것을 피하기 위해, 주어진 현재 자원 구속조건에서 가능한 빨리 작업을 실행하도록 시도하는 것 또는 "비번시"에 실행하도록 작업을 스케쥴링하는 것과 대조적으로, 이 작업은 컴퓨터 상에서 진행 과정에 맞춰 실행되지만, 그 작업이 사용자 또는 다른 컴퓨터 작업들에 대해 인식불가능할 수 있는 미세한 피스들로 실행된다. 따라서, 작업은 다른 작업들 및 애플리케이션들에 대해 그리고 사용자에 대해 완전히 투명할 수 있다. 사용자는 작업을 스케쥴링할 필요가 없으며, 이러한 방법으로, 작업은 성능 결정적인 시간 동안을 포함한 어떠한 시간에도 실행될 수 있다.

<23> 이 설명에서 전반적으로 이용될 바와 같이, 다른 작업들, 프로세스들 및 애플리케이션들에 대한 용어 "투명한"은 다른 작업들 및 애플리케이션들이 마이크로-작업들의 실행으로 인한 상당히 부정적인 어떠한 성능 충격도 겪지 않고 실행할 수 있음을 의미한다. 부정적인 충격은 테스트 작업이 독립적으로 실행하는데 얼마나 오래 걸리는지와, 마이크로-작업들로 분할된 컴퓨팅 작업이 실행하는 동안을 비교하여 결정될 수 있다. 이들 2가지 경우에 대한 테스트 작업의 실행 시간에는 상당한 차이가 없어야 한다.

<24> 마이크로-작업 스케쥴러(MJS)는 본 발명의 일 실시예에 따라 마이크로-작업들이 실행되어야 하는 때를 판단한다. 일 실시예에서, 애플리케이션 프로그램은 자신의 컴퓨팅 작업들을 복수의 마이크로-작업들로 분할한다. 여기에 이용된 바와 같이, 용어, 복수는 1보다 큰 임의의 수를 의미한다. MJS는 마이크로-작업들이 실행해야 할 때를 판단한다. MJS로 작업하도록 실행되는 애플리케이션들은 여기서 MJS-실행가능 애플리케이션으로 언급된다.

<25> 일 실시예에서, MJS는 메모리 매니저를 갖는다. MJS-실행가능 애플리케이션은 오퍼레이팅 시스템으로부터 메모리를 요청하는 것에 반대로, MJS 메모리 매니저로부터 메모리를 요청한다. 일 실시예에서, MJS-실행가능 애플리케이션은 (MJS-실행가능 애플리케이션에 할당되고, MJS-실행가능 애플리케이션이 실행하는 메모리의 양을 의미하는) 매우 작은 인-메모리 풋프린트를 갖고 있다. 작은 풋프린트를 실현하기 위해, MJS를 이용하는 프로그램이 이용하는 메모리의 양에 대해 한계값을 둔다.

<26> 다른 실시예에서, 컴퓨터 자원 이용률(utilization)은 자원 이용률이 하나 이상의 유휴 기준에 따르는지를 판단하기 위해 모니터링되고 분석된다. MJS는 하나 이상의 유휴 기준이 만족될 때 마이크로-작업들이 실행될 수 있게 한다. 특정 마이크로-작업이 실행하는데 필요한 시간이 특정 마이크로-작업에 의해 이용된 자원에 대한 유휴 시간의 통상의 윈도우보다 작기 때문에, 자원이 다른 작업에 필요해지기 전에 자원이 양도된다. 따라서, 마이크로-작업에 의한 자원 이용률은 통지되지 않은 상태로 될 수 있고 마이크로-작업은 자신의 애플리케이션 환경에 대해 가시가능하지 않을 수 있다.

<27> MJS-실행가능 애플리케이션은 본 발명의 일 실시예에 따라 자신의 자원 이용률이 산출되어야 하는 조건들을 정의하기 위해 MJS에 자원 이용률 임계 파라미터들을 전송한다. 이들 자원은 이들로 한정되는 것은 아니지만, 디스크 I/O, CPU 및 네트워크 이용률을 포함한다. 예를 들어, MJS-실행가능 애플리케이션은 상기 3개의 자원들의 임의의 조합에 대한 최소 임계 레벨들까지 마이크로-작업이 실행될 것을 요청할 수 있다.

<28> 일 실시예에서, 애플리케이션은 저장 매체 디프래그멘터이다. 디프래그멘터는 일일 스케쥴링된 조각모으기 작업을 완료하는데 12분의 벽시간(wall clock time)이 걸릴 수 있다. MJS-실행가능한 디프래그멘터는 디프래그멘터에 의해 필요한 자원들이 임의의 다른 작업 또는 애플리케이션에 의해 이용되고 있지 않을 때의 시간들을 선택하여, 조각모으기 작업을 한번에 수 밀리초 실행될 수 있는 많은 마이크로-작업들로 분할한다. MJS-실행가능 디프래그멘터는 12분 작업을 더 오랜 기간에 걸쳐 실행하는 마이크로-작업들로 분할하여, 컴퓨터 시스템이 온 상태에 있을 때는 언제든지 실행할 수 있다.

<29> 특정예로서, MJS-실행가능 디프래그멘터는 제1 마이크로-작업을 실행시켜 파일이 조각화되었는지의 여부를 판단할 수 있다. 제1 마이크로-작업의 실행은 다른 유휴 자원들만을 소모한다. 즉, 이용률이 하나 이상의 유휴 기준에 따르는 자원들만을 소모한다. 그 결과, 마이크로-작업의 실행은 다른 애플리케이션들에 대하여 투명하다. 또한, 메모리 이용률은 낮게 유지되며, 디프래그멘터와 관련된 메모리 할당이 마이크로-작업의 실행 이전에 또는 실행에 뒤이어 변경될 필요가 없다.

<30> 각각의 마이크로-작업을 실행하기 전에, MJS-실행가능 디프래그멘터는 컴퓨터 자원 이용률이 하나 이상의 유휴 기준에 따르는지의 판단을 행한다. 따라서, MJS-실행가능 디프래그멘터는 컴퓨터 자원 이용률이 마이크로-작업이 진행될 수 있을 정도로 현재 충분히 낮은지를 판단하였다. 자원 이용률이 너무 높다면, 마이크로-작업의 실행은 연기된다. 제1 마이크로-작업이 실행된 후, MJS-실행가능 디프래그멘터는 제2 마이크로-작업을 반드시 즉시 실행할 필요가 있는 것은 아니다. 다만, 후속하는 마이크로-작업들의 실행은 다른 애플리케이션들이 동일한 자원을 필요로 하는 경우에 시간에 걸쳐 분산될 수 있다.

<31> 아키텍쳐 개요

<32> 도 1은 본 발명의 일 실시예에 따라 마이크로-작업들을 실행하기 위한 아키텍쳐(100)의 블록도를 나타낸다. 각각의 MJS-실행가능 애플리케이션(115(1) - 115(n))은 자신의 컴퓨팅 작업(또는 작업들)을 실행을 위한 마이크로-작업들(125)로 분할한다. 예를 들어, 애플리케이션 프로그래머는 마이크로-작업(125)을 실행시키기 위해 MJS(110)로부터의 허가를 요청하는 애플리케이션 코드에서의 적절한 위치들에 호출들(call)을 위치시킬 수 있으며, 이는 실제로 컴퓨팅 작업을 마이크로-작업들(125)로 분할한다. 마이크로-컴퓨팅 작업들은 일 실시예에 따라 다음의 마이크로-작업(125)이 실행할 때까지 실행 중 일시적인 중지에 대해 안전하게 허용하는 한편 단일 유닛으로서 완료될 수 있는 실질적으로 더 작은 (예를 들어, 최소의) 작업 유닛들이다. 실행시 중단에 대해 안전하게 허용함으로써, 마이크로-작업들 모두의 실행으로부터 야기되는 결과에 영향을 주지 않고 특정 마이크로-작업의 실행이 지연될 수 있음이 의미된다. 마이크로-작업들(125)을 작게 유지함으로써, MJS-실행가능 애플리케이션(115)이 한번에 작은 양의 컴퓨터 자원들만을 이용할 수 있게 된다. 따라서, 마이크로-작업(125)의 실행은 본 발명의 일 실시예에 따라 컴퓨터 시스템에서의 다른 애플리케이션의 성능에 상당한 충격을 주지 않도록 상당히 작은 양의 자원들을 소모한다. 예로서, MJS-실행가능 애플리케이션(115(1) - 115(n))은 백업, 인덱싱, 소프트웨어 업데이트, 바이러스 및 악성 소프트웨어 스캔, 및 조각 모으기와 같은 메인터넌스를 수행할 수 있다. 그러나, MJS-실행가능 애플리케이션(115(1) - 115(n))은 또한 메인터넌스 외의 소프트웨어일 수 있다.

<33> 마이크로-작업 스케쥴러(MJS; 110)는 마이크로-작업들(125)이 실행될 수 있을 때를 판단한다. 이 실시예에서, MJS(110)는 특정 MJS-실행가능 애플리케이션(예를 들어, 115(1))으로 하여금 하나 이상의 마이크로-작업들(125)이 실행을 허용받을 수 있도록 요청할 수 있게 하기 위한 애플리케이션 프로그램 인터페이스(API; 130)를 갖는다. API(130)는 또한, 아래 보다 자세히 설명될 바와 같이, MJS-실행가능 애플리케이션(115)으로 하여금 얼마나 많은 마이크로-작업들(125)이 분산될 수 있는지에 의해 특정할 수 있도록 한다. 예시적인 API가 여기서 아래에 포함된다. 그러나, 아키텍쳐(100)는 예시적인 API로 한정되지 않는다.

<34> MJS(110)가 어느 마이크로-작업(125)이 다음에 실행되는 것을 허용받아야 하는지를 결정할 수 있도록 마이크로-작업 스케쥴러(110)는 마이크로-작업 대기열(queue)을 유지한다. 마이크로-작업들의 실행은 다른 애플리케이션에 무시가능한 충격을 주도록 MJS(110)에 의해 타이밍된다. 일 실시예에서, MJS(110)는 마이크로-작업들이 유휴 자원들만을 이용하도록 마이크로-작업들을 스케쥴링한다. MJS(110)는 스케쥴링 결정을 행하기 위해 자원 이용률이 하나 이상의 유휴 기준에 따르는지를 판단한다. 즉, 이 실시예에서, MJS(110)는 자원 기준으로 된다. 마이크로-작업들의 실행은 API 호출에서의 MJS-실행가능 애플리케이션에 의해 또는 애플리케이션과 MJS 사이의 다른 통신 방법에 의해 특정될 수 있다. 스케쥴러(105)가 다음 마이크로-작업(125)이 다른 작업들에 충격을 주지 않고 실행할 수 있다고 판단한 경우, MJS(110)는 MJS-실행가능 애플리케이션(115(1))에 마이크로-작업(125)을 실행하도록 명령함으로써 MJS-실행가능 애플리케이션(115)에 응답한다. MJS는 일 실시예에서, 자원 이용률에 기초하여 언제 작업들을 스케쥴링하는지에 대해 자신의 판단에 기초한다. 일례로서, MJS는 디스크 액티비티를 분석할 수 있다. 마이크로-작업을 가진 애플리케이션 외의 애플리케이션이 디스크를 이용하는 경우, MJS는 다른 애플리케이션이 마이크로-작업을 스케쥴링하는 것을 완료할 때까지 기다린다. MJS는 디스크 I/O 이용률을 모니터링하는 것을 계속하고, 다른 어떤 애플리케이션이 디스크 I/O에 액세스하는 것을 시도하지 않는다면 다른 마이크로-작업이 스케쥴링될 수 있게 한다. 그러나, 다른 애플리케이션이 디스크 I/O의 이용을 시도한다면, MJS는 다른 마이크로-작업이 스케쥴링될 수 없게 할 것이며, 여기서, 다른 애플리케이션이 디스크 I/O를 이용할 수 있다.

<35> 다른 예로서, MJS는 네트워크 액티비티를 분석할 수 있다. 네트워크 트래픽이 너무 높다면, 트래픽이 느려질(slow) 때까지, MJS는 어떠한 마이크로-작업들도 스케쥴링하지 않을 것이다. 네트워크 트래픽이 충분히 느려진다면, MJS가 실행을 위하여 마이크로-작업을 스케쥴링한다. MJS는 네트워크 트래픽이 충분히 느린 상태로 유지되는 것을 보장하도록 검사를 계속할 수 있다. 네트워크 트래픽이 충분히 느린 상태로 유지된다면, 다른 마이크로-작업이 스케쥴링될 수 있다. 그러나, 트래픽이 너무 높게 된다면, 추가의 마이크로-작업들이 실행하도록 스케쥴링되지 않는다.

<36> MJS는 임의의 유형의 계산 자원들 및 자원들의 임의의 조합에 기초하여 자원 기반 스케쥴링 결정들을 행할 수 있다. 일 실시예에서, MJS는 실행에 대한 허가(permission)를 대기하는 마이크로-작업들의 복수의 대기열을 갖는다. 각각의 대기열은 특정 자원에 대응할 수 있다. 예를 들어, 디스크 I/O를 이용하는 것을 필요로 하는 마이크로-작업들에 대한 대기열, 네트워크를 이용하는 것을 필요로 하는 마이크로-작업들에 대한 대기열, CPU를 이용하는 것을 필요로 하는 마이크로-작업들에 대한 대기열 등이 있을 수 있다. 또한, 자원들의 조합을 이용하는 마이크로-작업들에 대한 하나 이상의 대기열이 있을 수 있다. MJS는 특정 자원 또는 자원들의 조합이 이용가능할 때 마이크로-작업들을 전개한다. 특정 마이크로-작업은 2개의 자원의 이용을 필요로 할 수 있다. 예를 들어, 특정 마이크로-작업이 네트워크 자원의 이용 및 디스크 자원의 이용을 필요로 할 수 있다. 그러나, 특정 마이크로-작업은 CPU 자원을 필요로 하지 않는다. 심지어 CPU 자원 이용률이 현재 높은 경우에도, 특정 마이크로-작업은 여전히 스케쥴링되어 실행될 수 있다.

<37> MJS(110)가 도 1에서 MJS-실행가능 애플리케이션(115)과는 별도의 프로그램으로서 도시되어 있지만, MJS(110)는 MJS-실행가능 애플리케이션(115) 내에 통합될 수 있다. 따라서, MJS(110)는 일 실시예에 따라 마이크로-작업들(125)이 MJS-실행가능 애플리케이션(115(1))에 응답을 전송함이 없이 실행될 수 있게 한다. 따라서, API(130)는 선택적이다.

<38> 일 실시예에서, MJS(110)는 오퍼레이팅 시스템의 일부이다. 다른 실시예에서, MJS(100)는 오퍼레이팅 시스템으로부터 외부에서 실행한다. 일 실시예에서, MJS는 오퍼레이팅 시스템 외부에서 실행하는 경우, MJS는 자신의 자원 이용률에서 자체적으로 한정한다. 예를 들어, MJS(110)는 자신의 자원 이용률을 모니터링하고, 자신의 자원 이용률이 너무 높게 되면 MJS(110)는 MJS(110)를 스케쥴링하는 것을 일정 기간 동안에 정지하도록 오퍼레이팅 시스템에 요청을 행한다.

<39> MJS-실행가능 애플리케이션(115)은 본 발명의 일 실시예에 따라 자원 이용률을 제어하도록 MJS(110)에 파라미터를 전송한다. 자원 이용률의 제어는 이들에 한정되는 것은 아니지만, 디스크 I/O, CPU 및 네트워크를 포함한다. 예를 들어, MJS-실행가능 애플리케이션(115)은 위의 3개의 자원들의 임계 레벨들의 임의의 조합 까지 실행된 마이크로-작업을 요청할 수 있다. 또한, MJS-실행가능 애플리케이션(115)은 서로 다른 마이크로-작업들(125)에 대하여 서로 다른 자원 임계 레벨들을 특정할 수 있다. 예를 들어, MJS-실행가능 애플리케이션(115)은 일 실시예에 따라 각각의 마이크로-작업(125)에서의 다른 자원 임계 레벨을 특정한다. 따라서, 미세-단위로 된(fine-grained) 자원 관리가 가능하다. MJS(110)가 자원 이용률을 계산하는 경우, 본 발명의 일 실시예에 따라, 이것은 측정된 MJS-실행가능 애플리케이션 외의 프로세스들의 자원 이용률이다. CPU 이용률 임계값이 20%로 설정되는 다음 예가 설명을 위해 이용된다. MJS-실행가능 애플리케이션이 실행할 수 있기 전에 CPU 이용률이 20% 미만인 경우에, 마이크로-작업(들)이 실행할 때 CPU 이용률이 20% 이상으로 증가할 수 있다. 이 예에서, 20%를 초과하는 증가는 CPU 자원 이용률 위반으로 간주되지 않는다. 유사한 원리들이 네트워크 및 디스크 I/O 자원들에 적용된다.

<40> MJS(110)는 일 실시예에서, 또한 메모리 매니저(140)를 갖는다. MJS(110)가 초기화되는 경우 MJS(110)에는 오퍼레이팅 시스템에 의해 메모리가 할당되며, 메모리의 일부를 MJS(110)가 자신의 목적을 위해 사용하며, 메모리의 일부를 MJS-실행가능 애플리케이션(115)에 할당한다. MJS-실행가능 애플리케이션(115(1))이 개시하는 경우, MJS-실행가능 애플리케이션(115(1))은 MJS(110)로부터 메모리 할당을 요청한다. MJS(110)는 모든 프로세스들에 의한 현재 컴퓨터 시스템 메모리 이용률 및 MJS-실행가능 애플리케이션(115(1))의 수요와 같은 인자들에 기초하여, 얼마나 많은 메모리를 MJS-실행가능 애플리케이션(115(1))에 할당할 것인지를 판단할 수 있다. 메모리 요구량들은 각각의 MJS-실행가능 애플리케이션(115)에 고유할 수 있으며, 컴퓨터 소프트웨어 프로그래머에 의해 MJS-실행가능 애플리케이션(115) 내에 프로그래밍될 수 있다.

<41> 도 2는 본 발명의 일 실시예에 따라 MJS-실행가능 애플리케이션 메모리 풋프린트(204(1) - 204(n))에 비교되는 통상의 애플리케이션 메모리 풋프린트(202)의 비교를 나타낸다. MJS-실행가능 애플리케이션(115(1))은 마이크로-작업들(125)을 실행시키기 때문에, 메모리 할당(204(1))이 매우 작을 수 있다. 또한, 메모리 할당(204(1))이

매우 작기 때문에, MJS-실행가능 애플리케이션(115(1))이 자신의 할당된 메모리(204(1))를 항상 양도하는 것이 반드시 필요한 것은 아닐 수 있다. 따라서, MJS-실행가능 애플리케이션(115)은 빈번한 메모리 할당 및 할당 해제(de-allocation)를 야기하지 않는다. 메모리 할당에서의 감소 및 제거는 왜 MJS-실행가능 애플리케이션(115)이 다른 애플리케이션 및 작업들에 상당한 충격을 주지 않는지의 한 이유이다.

<42> 프로세스 흐름

<43> 도 3은 본 발명의 일 실시예에 따라 마이크로-작업들을 이용하여 MJS-실행가능 애플리케이션을 실행하기 위한 프로세스(300)의 단계들을 나타내는 흐름도이다. 단계 301에서, MJS-실행가능 애플리케이션 프로그램이 개시된다. 일 실시예에서, 컴퓨터 시스템이 부팅될 때 MJS-실행가능 애플리케이션 프로그램이 개시된다. MJS-실행가능 애플리케이션이 수행할 작업을 갖고 있지 않은 경우, MJS-실행가능 애플리케이션은 실행할 작업을 가질 때까지 유휴 상태에 놓여있게 된다. 이 유휴 상태에서, MJS-실행가능 애플리케이션은 수시 모니터링과 같은 일부 기능들을 수행할 수 있다. 단계 302에서, MJS-실행가능 애플리케이션이 저장 매체를 조각모으기하는 것, 또는 바이러스들에 대한 스캐닝과 같은 수행할 작업을 갖는다. 작업은 단일 디스크 및 그 단일 디스크 상에 저장된 파일들을 조각모으기하는 것일 수 있고 여기서, MJS-실행가능 애플리케이션은 그 디스크를 진행 과정에 맞춰 조각모으기한다.

<44> MJS-실행가능 애플리케이션이 개시될 때 작은 양의 메모리가 MJS-실행가능 애플리케이션에 할당된다. MJS-실행가능 애플리케이션이 일반적으로 한번에 단일의 마이크로-작업을 실행하는 것만을 시도하기 때문에 할당된 양은 매우 작을 수 있다. 그러나, 일부 경우에, MJS-실행가능 애플리케이션은 실행할 다른 프로세스를 대기함이 없이 복수의 마이크로-작업들을 실행하는 것을 시도할 수 있다. 예를 들어, MJS가 필요한 컴퓨터 시스템 자원이 유휴 상태라고 판단한다면, MJS는 다른 프로세스가 마이크로-작업들에 의해 이용되는 자원들을 이용함이 없이 MJS-실행가능 애플리케이션이 복수의 마이크로-작업들을 연속적으로(in a row) 실행하게끔 할 수 있다.

<45> 단계 304에서, 컴퓨팅 작업은 마이크로-작업들로 분할된다. 마이크로-작업들은 본 발명의 일 실시예에 따라 컴퓨터 시스템에서의 다른 작업의 성능에 상당한 충격을 주지 않도록 마이크로-작업들의 실행이 충분히 작은 양의 자원들을 이용하게 하는 크기로 이루어진다. 컴퓨팅 작업을 마이크로-작업들로 분할하는 것은 MJS-실행가능 애플리케이션 내에서 명령에 의해 달성될 수 있다. 일반적으로, 이들 명령은 MJS-실행가능 애플리케이션에서의 결정 포인트들이다. 예를 들어, 명령들은 마이크로-작업을 실행하도록 허가를 요청하는 MJS에 대한 API 호출들일 수 있다. 그러나, MJS는 MJS-실행가능 애플리케이션과 통합될 수 있으며, 이 경우, 명령들은 MJS-실행가능 애플리케이션 내의 스케쥴링 기능에 대한 호출들일 수 있다. 컴퓨팅 작업을 마이크로-작업들로 분할하기 위해 다른 기술들이 이용될 수 있다.

<46> 단계 306에서, 각각의 마이크로-작업들에 대하여 특정 마이크로-작업에 의해 이용될 컴퓨터 시스템의 하나 이상의 자원들의 이용률이 하나 이상의 유휴 기준을 만족하는지에 관한 판단이 행해진다. 따라서, 마이크로-작업들은, 마이크로-작업들에 의해 요구되는 컴퓨터 시스템의 자원들이 충분히 유휴 상태에 있는 경우에만 때때로 실행된다. 일 실시예에서, 유휴 기준은 자원 임계값에 기초한다. 예를 들어, 자원 임계값이 이용될 수 있으며, 여기서, MJS-실행가능 애플리케이션의 마이크로 작업은 다른 프로세스들에 의한 자원 이용률이 MJS-실행가능 애플리케이션에 의해 특정된 임계값 미만인 경우에만 실행한다. 아래 설명된 예시적인 API는 일부 자원 임계 파라미터들의 일례를 포함한다. 그러나, 프로세스(300)는 이들 자원 임계 파라미터로 한정되는 것은 아니다. 단계 306은 각각의 마이크로-작업에 의해 요구되는 자원들의 이용가능성에 종속되는 시간에 걸쳐 마이크로-작업들의 실행을 분산(spread)시킨다. 따라서, 마이크로-작업들의 실행은 컴퓨터 시스템에서의 다른 작업들 및 애플리케이션들의 성능에 상당한 충격을 주지 않는다.

<47> 특정 자원에 대한 유휴 기준은 하나 이상의 인자들에 기초할 수 있다. 예를 들어, 일 실시예에서, CPU 이용률은 CPU 자원 이용률에 대한 유휴 기준으로서 이용된다.

<48> 단계 307에서, 마이크로-작업들은 실행되어지도록 된다. 일 실시예에서, MJS는 특정 마이크로-작업이 실행될 수 있도록 하는 허가를 MJS-실행가능 애플리케이션에 제공한다. 실행할 마이크로-작업들이 더 있다면, 제어가 단계 306로 진행하여, 다음 마이크로-작업에 의해 이용될 컴퓨터 시스템의 하나 이상의 자원의 이용률이 하나 이상의 유휴 기준을 만족하는지를 판단한다.

<49> 메인터넌스 유형 실시예에서, MJS-실행가능 애플리케이션은 진행 과정에 맞춰 실행하여, 컴퓨터 시스템이 부팅된 상태로 유지되는 한 실행을 계속한다. 따라서, MJS-실행가능 애플리케이션이 자신의 작업을 완료하는 경우에도, 애플리케이션은 단계 308에 나타낸 바와 같이 실행을 계속한다. 따라서, MJS-실행가능 애플리케이션은 일반

적으로 애플리케이션의 개시를 대표하는 추가적인 자원들을 소모하지 않는다. MJS-실행가능 애플리케이션이 해야 할 다른 작업을 갖는다고 판단한 경우에, 단계 304에서 MJS-실행가능 애플리케이션은 새로운 컴퓨팅 작업을 마이크로-작업들로 분할하고, 단계 306에서 마이크로-작업들이 시간에 걸쳐 실행된다.

<50> 예시적인 API

<51> 본 발명의 일 실시예는 MJS-실행가능 애플리케이션이 MJS와 인터페이스할 수 있도록 하는 API이다. 예시적인 API는 CPU, 디스크 및 네트워크에 대한 다음의 자원 임계 파라미터들을 갖는다.

<52> - CPU 이용률 임계값

<53> - 보류중인 디스크 I/O 카운트 임계값

<54> - 네트워크 이용률 임계값

<55> 상술한 파라미터들은 각각의 마이크로-작업에 대하여 특정될 수 있다. 즉, 서로 다른 마이크로-작업들에는 서로 다른 자원 임계 파라미터들이 할당될 수 있다. 예를 들어, 네트워크를 이용하는 마이크로-작업에 대하여, 네트워크 임계값이 이용될 수 있다. 그러나, 네트워크 임계값은 네트워크를 이용하지 않는 마이크로-작업들에 대하여 제로일 수 있다. 따라서, 미세 단위로 된 자원 관리가 본 발명의 일 실시예에 따라 제공된다.

<56> 특정 예로서, MJS-실행가능 애플리케이션은 CPU 이용률이 50% 미만이고 I/O 디스크 이용률이 40% 미만이고 네트워크 트래픽이 60% 미만인 경우에만 특정 마이크로-작업이 실행되도록 요청할 수 있다. 전혀 다른 것을 포함하지 않는 자원 임계 인자들의 임의의 조합이 이용될 수 있다. CPU 이용률 임계값은 본 발명의 일 실시예에 따라 임의의 다른 작업의 것과 대조적으로, MJS의 CPU 이용들을 구별한다.

<57> 다음 2개의 파라미터들은 얼마나 빈번하게 자원 이용률이 측정되어야 하는지를 특정하는데 이용된다.

<58> - CPU 이용률 윈도우

<59> - 네트워크 이용률 윈도우

<60> CPU 이용률 윈도우 파라미터는 CPU 이용률이 계산되는 시간 윈도우를 정의한다. 예를 들어, 마지막 n 밀리초에 걸친 CPU 이용률이 평균화된다. 네트워크 이용률 윈도우는 네트워크 이용률이 계산되는 시간 윈도우를 정의한다. 이들 파라미터는 MJS에 대하여 내부적일 수 있다. 그러나, MJS-실행가능 애플리케이션은 이들 파라미터를 오버라이드할 수 있다. 보류중인 디스크 I/O는 임의의 시점에서 절대적이며, 따라서, 이 I/O는 계산될 필요가 없다.

<61> 필수(mandatory) 유휴 시간 파라미터는 마이크로-작업들이 어떻게 시간에 걸쳐 분산될 수 있는지를 제어하도록, MJS-실행가능 애플리케이션으로부터 MJS 엔진에 전달될 수 있다. 필수 유휴 시간 파라미터는 선택적이다. 또한, 이용될 때, 필수 유휴 시간 파라미터는 제로 값을 가질 수 있다.

<62> - 필수 유휴 시간

<63> MJS는 모든 마이크로-작업들이 실행된 후에 시스템 유휴 시간으로서 정의되는 "유휴 시간"의 트랙을 유지시킨다. 이전에 설명된 바와 같이, MJS-실행가능 애플리케이션(들)은 MJS를 이용하여 마이크로-작업들을 대기열에 저장시킬 수 있다. MJS 대기열 상에 마이크로-작업들이 없는 경우, MJS는 특정된 필수 유휴 시간 동안 기다린 다음, MJS-실행가능 애플리케이션(들)을 기상시키고, MJS-실행가능 애플리케이션(들)에 추가적인 작업들을 수행하도록 권한을 부여한다. 예를 들어, MJS-실행가능 디프래그멘터가 디스크 드라이브를 조각모으기하도록 먼저 복수의 마이크로-작업들을 실행한 다음, MJS 마이크로-작업 스케쥴러에 의해 중단될 수 있다. 특정된 필수 유휴 시간 후에, MJS는 추가적인 작업에 권한을 부여하도록 MJS-실행가능 디프래그멘터를 호출한다. 예를 들어, MJS-실행가능 디프래그멘터는 메모리를 해제하는 것과 같은 클린업(clean-up) 작업을 실행할 수 있다. 필수 유휴 시간은 MJS-실행가능 애플리케이션에 의해 조정될 수 있는 디폴트 파라미터일 수 있다.

<64> 다음의 파라미터들은 자원 이용률이 임계값을 초과할 때 마이크로-작업을 실행하는 것을 대기하는 것에 관한 것이다.

<65> - 대기 시간

<66> - 최대 대기 시간

<67> MJS가, 자원 이용률이 현재 너무 높아 마이크로-작업을 실행할 수 없다고 판단한다면, MJS는 특정된 대기 시간

동안을 기다린 다음, 자원 이용률을 재검사한다. 대기 시간 파라미터는 MJS가 자원 이용률이 너무 높다고 판단할 때마다 증가될 수 있다. 예를 들어, MJS는 최대 대기 시간(Max Wait Time)에 도달할 때까지 대기 시간 파라미터를 증가시킬 수 있다. 이들 파라미터는, MJS-실행가능 애플리케이션이 처음 시작될 때, MJS-실행가능 애플리케이션에 의해 특정될 수 있다. MJS-실행가능 애플리케이션은 자신의 실행 시간 동안에 이들 파라미터를 조정할 수 있다.

<68> 디프래그멘터 실시예

<69> 일 실시예에 따르면, MJS-실행가능 애플리케이션은 디프래그멘터이다. 도 4는 본 발명의 일 실시예에 따라 마이크로-작업들을 이용한 MJS 디프래그멘터의 프로세스(400)의 단계들을 나타낸다. 프로세스(400)는 디프래그멘터의 적어도 일부분이 마이크로-작업 개념을 이용할 수 있는 한 방법의 일례이다. 이 예에서, MJS-실행가능 애플리케이션은 API를 통하여 MJS와 인터페이스한다. 그러나, 이전에 설명된 바와 같이, MJS는 MJS-실행가능 애플리케이션 내에 통합될 수 있으며, 여기서, API는 반드시 필수적인 것인 아니다.

<70> 조각 모으기의 프로세스는 조각화된 파일들에 대한 디스크 드라이브를 스캐닝하는 것을 포함한다. 이 스캐닝은 파일 기록을 얻은 마이크로-작업들과, 그 파일이 조각화된 것인지를 판단하는 마이크로-작업들인 개별 마이크로-작업들로 나누어질 수 있다. 디스크 드라이브를 스캐닝하는 것은 도 4의 단계들 402 - 408로 나타내어진다.

<71> 단계 402에서, MJS-실행가능 디프래그멘터는 마이크로-작업을 실행하도록 MJS로부터 허가를 요청하기 위해 마이크로-작업 API를 호출한다. API 호출은 이 마이크로-작업에 대한 자원 이용률 임계 파라미터들을 특정할 수 있다. 다른 방법으로, 이전에 정의된 파라미터들이 이 마이크로-작업에 대해 제공될 수 있다. 요청을 수신하는 것에 응답하여, MJS는, 마이크로-작업이 실행될 수 있을 때를 판단한다. 이 판단은 자원 이용률 임계 파라미터들에 기초할 수 있다.

<72> 단계 404에서, MJS-실행가능 애플리케이션이 허가를 수신한 후, MJS 디프래그멘터는 마이크로-작업을 실행하며, 이 경우, 마이크로-작업은 다음 파일 기록을 얻는 것이다. 단계 406에서, MJS-실행가능 애플리케이션은 다시 마이크로-작업 API를 호출한다. MJS-실행가능 애플리케이션이 MJS으로부터 실행하도록 허가를 수신하는 경우, MJS-실행가능 애플리케이션은 단계 404로부터의 파일이 조각화된 것인지를 판단한다. 파일이 조각화된 것이 아니라면, 프로세스(400)는 단계 402로 복귀한다.

<73> 파일이 조각화된 것이라면, 단계 410 -416로 나타낸 바와 같이, MJS-실행가능 디프래그멘터가 파일을 조각모으기할 수 있다. 단계 410은 마이크로-작업 API에 대한 호출이다. 단계 412는 파일에 대한 자유 디스크 공간을 찾고 그 자유 공간의 할당을 획득하는 것이다.

<74> 마이크로-작업 API를 호출하는 단계 414와, 파일의 피스를 이동시키는 단계 416은 전체 파일이 이동될 때까지 반복된다. 예를 들어, 이동될 파일은 다른 애플리케이션의 성능에 상당한 충격을 야기하지 않도록 충분히 작은 피스들로 이동될 수 있다.

<75> 따라서, 마이크로-작업 개념은 진행 과정에 맞춰 조각화를 모니터링하고 조각화가 발생하자마자 파일들을 조각모으기하는 동적 디프래그멘터를 제공한다. 이는 스케쥴링된 조각모으기 실행 시간을 대기함이 없이 조각화의 발생시 즉시 조각화된 파일들을 조각모으기하는 MJS-실행가능 디프래그멘터를 가져온다. 따라서, MJS-실행가능 디프래그멘터는 본 발명의 일 실시예에 따르면 실시간 디프래그멘터이다.

<76> 쉘 마이크로-작업 스케쥴러

<77> 일 실시예에서, MJS는 컴퓨팅 작업을 마이크로-작업들로 자동으로 분할한다. 예를 들어, MJS는 MJS-실행되지 않는 애플리케이션 프로그램들 주변을 랩핑(wrap)하는 쉘로서 역할을 한다. 따라서, 쉘 MJS는 임의의 실행가능한 애플리케이션이 실행될 수 있는 완벽한 소프트웨어 애플리케이션이다. 쉘 MJS는 실행가능한 애플리케이션으로부터의 컴퓨팅 작업을 마이크로-작업들로 자동으로 분할한다. 즉, 이 실시예에서, 애플리케이션 프로그래머는 애플리케이션을 마이크로-작업들로 분할할 필요가 없다.

<78> 일 실시예에서, 쉘 MJS는 자원 이용률에 기초하여 실행가능 애플리케이션으로부터의 컴퓨팅 작업을 마이크로-작업들로 분할한다. 쉘 MJS는 애플리케이션과, 애플리케이션이 어떤 자원을 이용하는지를 알도록 어떻게 애플리케이션이 실행하는지를 분석할 수 있다. 예를 들어, MJS는 어떤 자원을 애플리케이션이 이용하는지를 분석하고 애플리케이션이 어느 정도로 자원을 이용하는지를 분석한다. 예를 들어, 디스크 디프래그멘터가 실행할 때 쉘 MJS는 어떤 자원들(예를 들어, CPU, 네트워크, 디스크 I/O)을 애플리케이션이 이용하는지를 판단할 수 있다. 일 실시예에서, 쉘 MJS는 이 분석에 기초하여 어떻게 애플리케이션을 마이크로-작업들로 분할할지를 자동으로 판단한

다. 쉘 MJS는 또한 이 분석에 기초하여 어떻게 마이크로-작업들을 스케쥴링할지를 판단할 수 있다.

<79> 쉘 MJS는 어떻게 컴퓨팅 작업을 마이크로-작업들로 분할할지를 판단하고 및/또는 어떻게 실행을 위해 마이크로-작업들을 스케쥴링할지를 판단하도록 여러 파라미터들을 이용할 수 있다. 이들 파라미터는 사용자 입력에 기초할 수 있거나 또는 쉘 MJS에 의해 설정될 수 있다. 예를 들어, 사용자는 특정 애플리케이션이 높은 우선 순위를 가짐을 특정할 수 있다.

<80> 하드웨어 개요

<81> 도 5는 본 발명의 일 실시예가 실시될 수 있는 컴퓨터 시스템(500)을 나타내는 블록도이다. 프로세스(300)와 프로세스(400)의 단계들은 시스템(500)의 하나 이상의 컴퓨터 판독가능 매체 상에 명령으로서 저장되어 있고 컴퓨터 시스템(500)의 프로세서 상에서 실행된다. 컴퓨터 시스템(500)은 정보를 통신하기 위한 버스(502) 또는 다른 통신 메카니즘 및 정보를 처리하기 위하여 버스(502)와 결합된 프로세서(504)를 포함한다. 컴퓨터 시스템(500)은 또한 프로세서(504)에 의해 실행될 명령 및 정보를 저장하기 위해 버스(502)에 결합된, 랜덤 액세스 메모리(RAM) 또는 다른 동적 저장 장치와 같은 메인 메모리(506)를 포함한다. 메인 메모리(506)는 또한 프로세서(504)에 의해 실행될 명령의 실행 동안에 임시 변수들 또는 다른 중간 정보를 저장하기 위하여 이용될 수 있다. 컴퓨터 시스템(500)은 프로세서(504)에 대한 정적 정보 및 명령을 저장하기 위하여 버스(502)에 결합된 판독 전용 메모리(ROM; 508) 또는 다른 정적 저장 장치를 더 포함한다. 자기 디스크 또는 광 디스크와 같은 저장 장치(510)가 제공되어, 정보 및 명령을 저장하기 위하여 버스(502)에 결합된다. 컴퓨터 시스템(500)은 임의의 수의 프로세서(504)를 가질 수 있다. 예를 들어, 일 실시예에서, 컴퓨터 시스템(500)은 멀티-프로세서 시스템이다. 프로세서(504)는 임의의 수의 코어들을 가질 수 있다. 일 실시예에서, 프로세서(504)는 멀티-코어 프로세서(504)이다. 컴퓨터 시스템(500)은 하이퍼 스레디드 머신(hyper threaded machine)에 이용될 수 있다.

<82> 컴퓨터 시스템(500)은 컴퓨터 사용자에 정보를 표시하기 위해 버스(502)를 통하여 음극선관(CRT)과 같은 디스플레이(512)에 결합될 수 있다. 영숫자 및 다른 키들을 포함한 입력 장치(514)는 프로세서(504)에 정보 및 커맨드 선택들을 통신하기 위하여 버스(502)에 결합된다. 다른 유형의 사용자 입력 장치는 프로세서(504)에 방향 정보 및 커맨드 선택을 통신하고 디스플레이(512) 상에서 커서 이동을 제어하기 위한 마우스, 트랙볼, 또는 커서 방향 키들과 같은 커서 제어부(516)이다. 이 입력 장치는 일반적으로 장치가 평면에서의 위치들을 특정할 수 있게 하는 2개의 축인 제1 축(예를 들어, x) 및 제2 축(예를 들어, y)으로 2개의 자유도를 갖는다.

<83> 본 발명은 여기에서 설명된 기술을 구현하기 위한 컴퓨터 시스템(500)의 이용에 관한 것이다. 본 발명의 일 실시예에 따르면, 이들 기술은 프로세서(504)가 메인 메모리(506)에 포함된 하나 이상의 명령의 하나 이상의 시퀀스들을 실행시키는 것에 응답하여 컴퓨터 시스템(500)에 의해 수행된다. 이러한 명령은 저장 장치(510)와 같은 다른 머신 판독가능 매체로부터 메인 메모리(506) 내에 읽어 넣을(read into) 수 있다. 메인 메모리(506)에 포함된 명령들의 시퀀스들의 실행은 프로세서(504)로 하여금 여기에 설명된 프로세스 단계들을 수행하도록 한다. 대안의 실시예들에서, 하드-와이어형 회로(hard-wired circuitry)가 본 발명을 실시하기 위해 소프트웨어 명령과 결합하여 또는 소프트웨어 명령 대신에 이용될 수 있다. 따라서, 본 발명의 실시예들은 하드웨어 회로 및 소프트웨어의 임의의 특정 조합으로 제한되지 않는다.

<84> 여기에 이용된 용어 "머신 판독가능 매체"는 특정 방식으로 머신이 동작하도록 하는 데이터를 제공하는데 참여하는 임의의 매체를 의미한다. 컴퓨터 시스템(500)을 이용하여 구현된 일 실시예에서, 예를 들어, 여러 머신 판독가능 매체가 실행을 위해 프로세서(504)에 명령을 제공하는데 참여된다. 이러한 매체는 이들로 한정되는 것은 아니지만, 비휘발성 매체, 휘발성 매체 및 전송 매체를 포함한 많은 형태를 취할 수 있다. 비휘발성 매체는 예를 들어, 저장 장치(510)와 같은 광 디스크 또는 자기 디스크를 포함한다. 휘발성 매체는 메인 메모리(506)와 같은 동적 메모리를 포함한다. 전송 매체는 버스(502)를 포함하는 와이어를 포함한, 동축 케이블, 구리선 및 광섬유를 포함한다. 전송 매체는 또한 라디오 파 및 적외선 데이터 통신 동안에 발생되는 것과 같은 음향파 또는 광파들의 형태를 취할 수 있다. 이러한 모든 매체는 매체에 의해 전달되는 명령을, 머신 내에 명령을 읽어 넣는 물리적 메카니즘에 의해 검출가능하도록 실체적이어야 한다.

<85> 예를 들어, 일반적인 형태들의 머신 판독가능 매체는 예를 들어, 플로피 디스크, 플렉시블 디스크, 하드 디스크, 자기 테이프, 또는 임의의 다른 자기 매체, CD-ROM, 임의의 다른 광학 매체, 펀치카드(punchcard), 페이퍼테이프, 홀들의 패턴들을 가진 임의의 다른 물리적 매체, RAM, PROM 및 EPROM, FLASH-EPROM, 임의의 다른 메모리 칩 또는 카트리지, 이후에 설명될 바와 같은 반송파 또는 컴퓨터가 판독할 수 있는 임의의 다른 매체를 포함한다.

<86> 여러 형태의 머신 판독가능 매체는 실행을 위해 프로세서(504)에 하나 이상의 명령의 하나 이상의 시퀀스들을 전달하는데 참여될 수 있다. 예를 들어, 명령은 원격 컴퓨터의 자기 디스크 상에서 초기에 전달될 수 있다. 원격 컴퓨터는 자신의 동적 메모리 내에 명령을 로딩시킬 수 있고 모뎀을 이용하여 전화선을 통해 명령을 전송할 수 있다. 컴퓨터 시스템(500)에 국부적인 모뎀은 전화선 상에서 데이터를 수신하고 적외선 송신기를 이용하여 데이터를 적외선 신호로 변환할 수 있다. 적외선 검출기는 적외선 신호로 전달된 데이터를 수신할 수 있고 적절한 회로가 버스(502) 상에 데이터를 위치시킬 수 있다. 버스(502)는 프로세서(504)가 명령을 검색하고 실행하는 메인 메모리(506)에 데이터를 전달한다. 메인 메모리(506)에 의해 수신된 명령은 프로세서(504)에 의한 실행 전에 또는 후에 저장 장치(510) 상에 선택적으로 저장될 수 있다.

<87> 컴퓨터 시스템(500)은 또한 버스(502)에 결합된 통신 인터페이스(518)를 포함한다. 통신 인터페이스(518)는 로컬 네트워크(522)에 접속된 네트워크 링크(520)에 결합한 양방향 데이터 통신을 제공한다. 예를 들어, 통신 인터페이스(518)는 전화선의 대응 유형에 데이터 통신 접속을 제공하기 위한 종합 정보 통신망(integrated services digital network; ISDN) 카드 또는 모뎀일 수 있다. 다른 예로서, 통신 인터페이스(518)는 호환가능 LAN에 데이터 통신 접속을 제공하기 위한 근거리 통신망(LAN) 카드일 수 있다. 무선 링크들이 또한 구현될 수 있다. 이러한 임의의 구현예에서, 통신 인터페이스(518)는 여러 유형의 정보를 나타내는 디지털 데이터 스트림을 전달하는 전기 신호들, 전자기 신호들 또는 광학 신호들을 전송 및 수신한다.

<88> 네트워크 링크(520)는 일반적으로 하나 이상의 네트워크를 통하여 다른 데이터 장치들에 데이터 통신을 제공한다. 예를 들어, 네트워크 링크(520)는 로컬 네트워크(522)를 통해 호스트 컴퓨터(524)에 또는 인터넷 서비스 공급자(ISP; 526)에 의해 조작되는 데이터 기기에 접속을 제공할 수 있다. ISP(526)는 이어서 현재 일반적으로 "인터넷"(528)이라 불리는 월드 와이드 패킷 데이터 통신 네트워크를 통하여 데이터 통신 서비스들을 제공한다. 로컬 네트워크(522)와 인터넷(528) 양쪽 모두는 디지털 데이터 스트림을 전달하는 전기 신호, 전자기 신호 또는 광학 신호를 이용한다. 여러 네트워크를 통한 신호들 및 네트워크 링크(520) 상에서 그리고 컴퓨터 시스템(500)으로부터 그리고 컴퓨터 시스템(500)에 디지털 데이터를 전달하는 통신 인터페이스(518)를 통하는 신호들은 정보를 전송하는 반송파의 예시적인 형태이다.

<89> 컴퓨터 시스템(500)은 네트워크(들), 네트워크 링크(520) 및 통신 인터페이스(518)를 통하여 메시지들을 전송하고, 프로그램 코드를 포함한 데이터를 수신할 수 있다. 인터넷 예에서, 서버(530)는 인터넷(528), ISP(526), 로컬 네트워크(522) 및 통신 인터페이스(518)를 통하여 애플리케이션 프로그램에 대하여 요청된 코드를 전송할 수 있다.

<90> 수신된 코드는 수신될 때 프로세서(504)에 의해 실행될 수 있고, 및/또는 이후의 실행을 위하여 저장 장치(510) 또는 다른 비휘발성 스토리지에 저장될 수 있다. 이러한 방식으로, 컴퓨터 시스템(500)은 반송파의 형태로 애플리케이션 코드를 획득할 수 있다.

<91> 상술한 설명에서, 본 발명의 실시예들은 구현예마다 다를 수 있는 복수의 특정한 세부 내용을 참조로 설명되었다. 따라서, 무엇이 본 발명인지 그리고 무엇이 출원인에 의해 본 발명인 것으로 의도되었는지에 관한 단 하나의 유일한 표시는 임의의 후속적인 수정안을 포함하여 본 출원에서 도출된 특정 형태의 청구범위 세트이다. 이와 같은 청구범위내에 수록된 용어들에 대해 상세한 설명에서 명백하게 기술한 임의의 정의들은 청구범위내에서 사용되는 이와 같은 용어들의 의미를 결정한다. 따라서, 청구범위에서 명백하게 언급하고 있지 않은 어떠한 제한, 구성요소, 특성, 특징, 이점 또는 속성도 상기 청구범위의 범위를 어떠한 방식으로든지 한정시켜서는 안된다. 따라서, 본 명세서 및 도면은 본 발명 범위를 한정시키고자 제시된 것이기 보다는 본 발명의 설명을 목적으로 제시된 것으로서 간주되어야 한다.

도면의 간단한 설명

<11> 본 발명은 첨부된 도면부의 도면들에서 제한이 아닌 일례로서 설명되어 있으며, 이 도면에서 동일한 도면 부호는 동일한 요소를 의미한다.

<12> 도 1은 본 발명의 일 실시예에 따라 마이크로-작업들을 실행시키기 위한 아키텍처도를 나타낸다.

<13> 도 2는 본 발명의 일 실시예에 따라 애플리케이션 메모리 풋프린트에 비교되는 통상의 애플리케이션 메모리 풋프린트의 비교를 나타낸다.

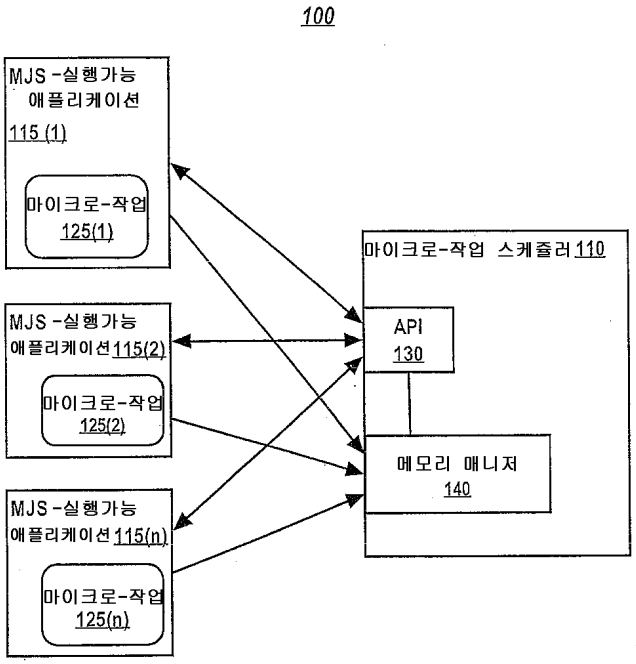
<14> 도 3은 본 발명의 일 실시예에 따라 마이크로-작업들을 이용하여 마이크로 작업 스케쥴러 실행가능 애플리케이션(micro-job scheduler enabled application)을 실행시키는 프로세스의 단계들을 나타내는 플로우차트이다.

<15> 도 4는 본 발명의 일 실시예에 따라 마이크로-작업들을 이용하여 저장 매체를 조각모으기하는 프로세스의 단계들을 나타낸다.

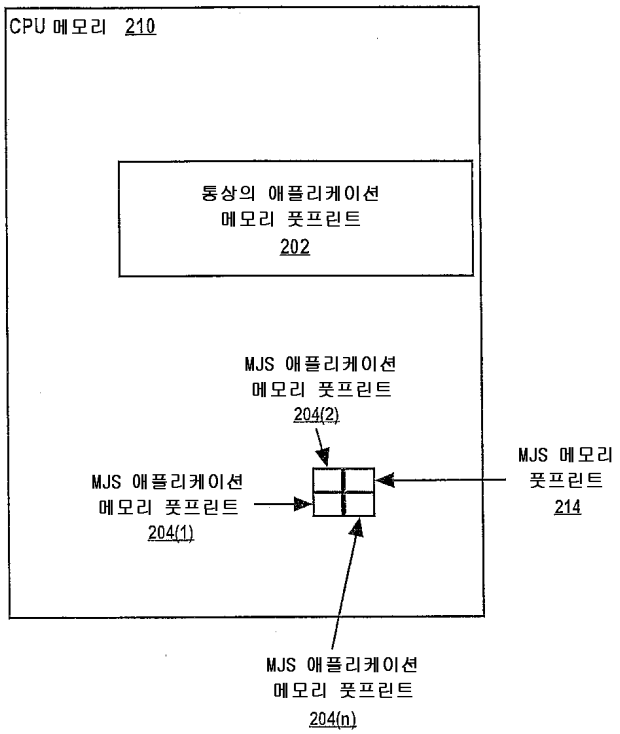
<16> 도 5는 본 발명의 일 실시예가 실시될 수 있는 컴퓨터 시스템을 나타내는 블록도이다.

도면

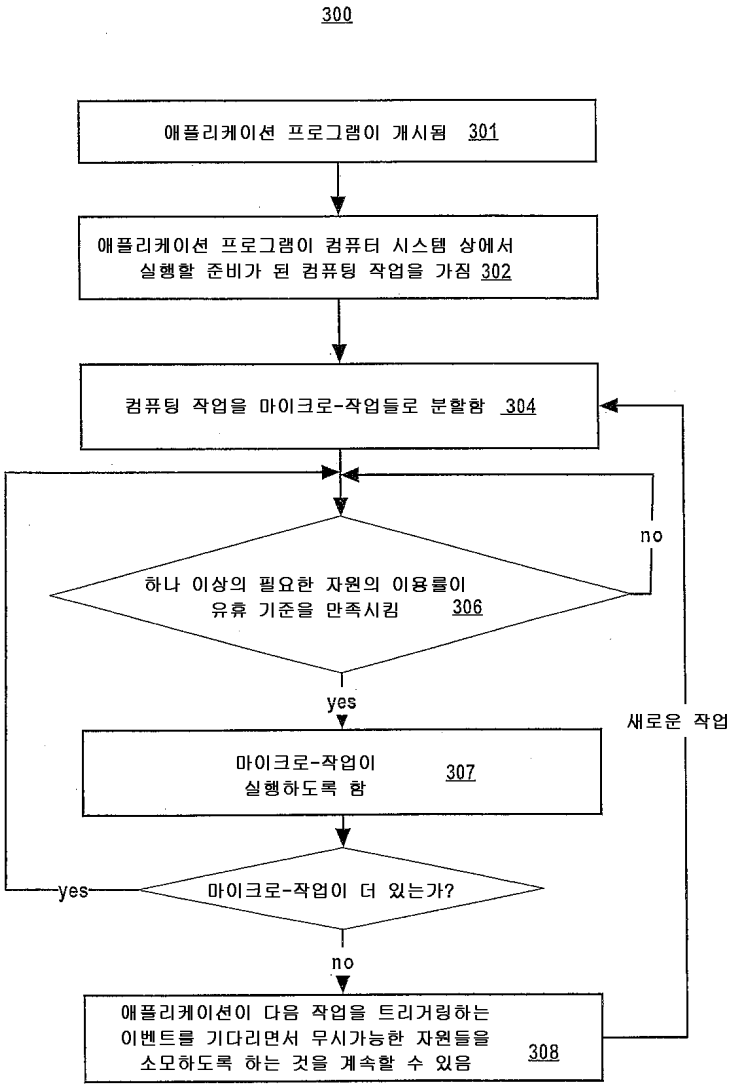
도면1



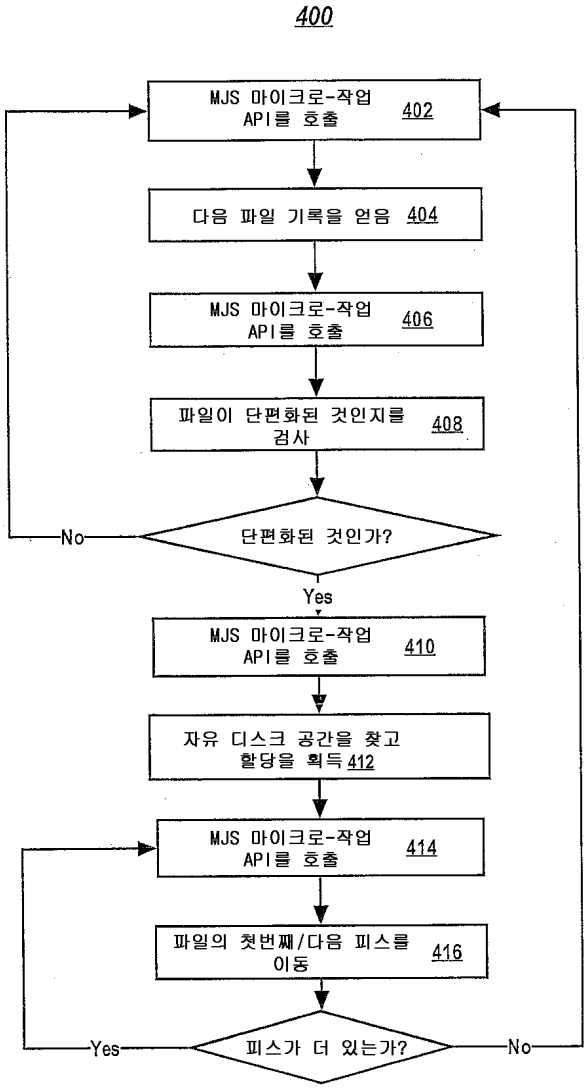
도면2



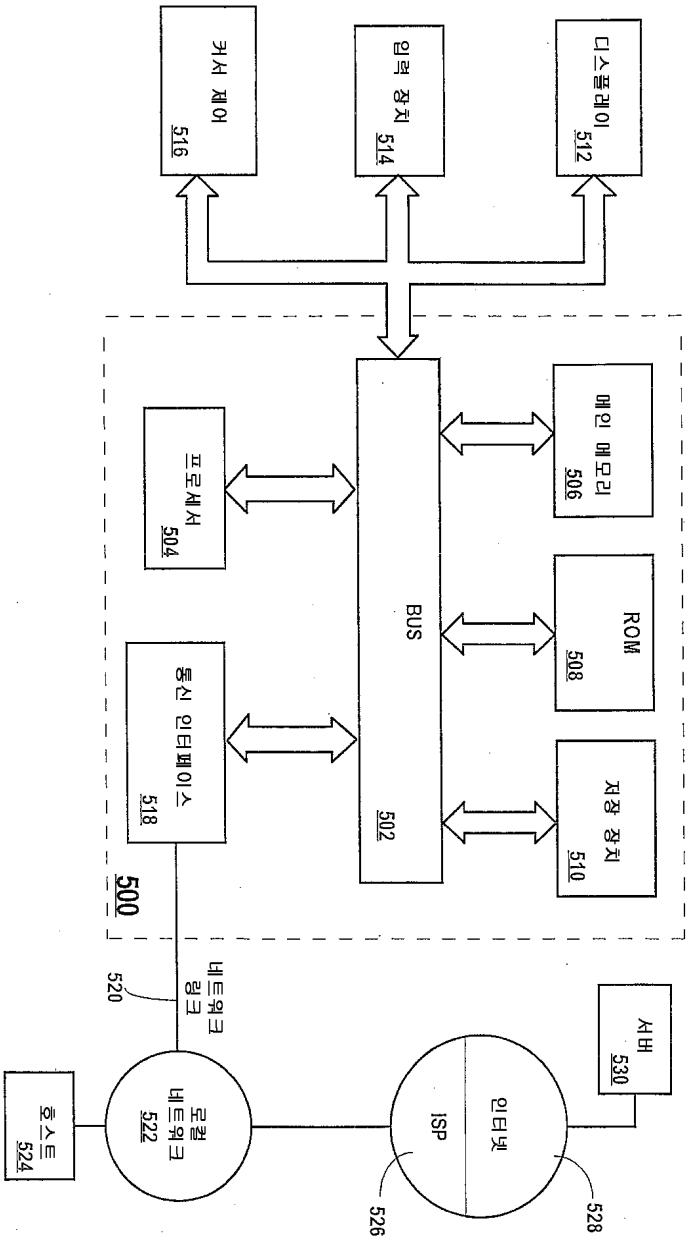
도면3


300
애플리케이션 프로그램이 개시됨 301
애플리케이션 프로그램이 컴퓨터 시스템 상에서 실행할 준비가 된 컴퓨팅 작업을 가짐 302
컴퓨팅 작업을 마이크로-작업들로 분할함 304
하나 이상의 필요한 자원의 이용률이 유휴 기준을 만족시킴 306
no
yes
새로운 작업
마이크로-작업이 실행하도록 함 307
마이크로-작업이 더 있는가?
yes
no
애플리케이션이 다음 작업을 트리거링하는 이벤트를 기다리면서 무시가능한 자원들을 소모하도록 하는 것을 계속할 수 있음 308

도면4


400
MJS 마이크로-작업 API를 호출 402
다음 파일 기록을 얻음 404
MJS 마이크로-작업 API를 호출 406
파일이 단편화된 것인지를 검사 408
단편화된 것인가?
No
Yes
MJS 마이크로-작업 API를 호출 410
자유 디스크 공간을 찾고 할당을 획득 412
MJS 마이크로-작업 API를 호출 414
파일의 첫번째/다음 피스를 이동 416
피스가 더 있는가?
Yes
No

도면5


디스플레이 512
입력 장치 514
커서 제어 516
메인 메모리 506
프로세서 504
BUS 502
ROM 508
통신 인터페이스 518
저장 장치 510
500
네트워크 링크 520
로컬 네트워크 522
호스트 524
ISP 526
인터넷 528
서버 530