(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2015-0143918
(43) 공개일자 2015년12월24일

(51) 국제특허분류(Int. Cl.)
G06F 9/46 (2006.01)
(21) 출원번호 10-2014-0072001
(22) 출원일자 2014년06월13일
심사청구일자 2014년06월13일

(71) 출원인
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
유혁
서울 강남구 광평로39길 3, 801동 101호 (수서동, 동익아파트)
박현찬
서울 성북구 인촌로7가길 34, 401호 (안암동2가)
(74) 대리인
김등용

전체 청구항 수 : 총 13 항

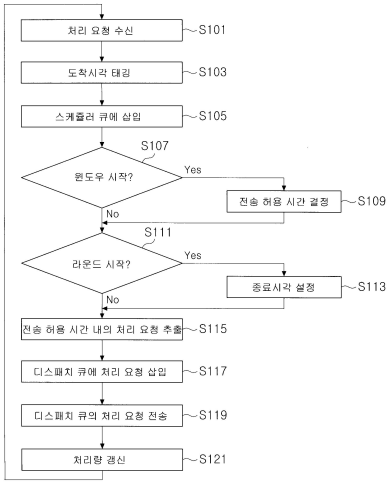
(54) 발명의 명칭 **요구 성능 보장 방법 및 요구 성능 보장 장치**

(57) 요 약

본 발명은 복수의 스케쥴러 큐 각각의 전송 허용 시간을 목표 요구 사항에 기초하여 결정하는 단계 및 각각의 결정된 전송 허용 시간 내에서 각각의 스케쥴러 큐의 처리 요청을 추출하는 단계를 포함하고 복수의 스케쥴러 큐 중 하나의 전송 허용 시간은 다른 하나의 전송 허용 시간과는 상이한 요구 성능 보장 방법에 관한 것이다.

본 발명을 이용함으로써 사용자 SLA에 맞춘 I/O 성능 차등화를 제공하고 동시에 SSD 사용률을 높여 전체 성능을 높일 수 있도록 하는 효과가 있다.

대 표 도 - 도4



명 세 서

청구범위

청구항 1

(a) 복수의 스케쥴러 큐 각각의 전송 허용 시간을 목표 요구 사항에 기초하여 결정하는 단계; 및

(b) 각각의 결정된 전송 허용 시간 내에서 각각의 스케쥴러 큐의 처리 요청을 추출하는 단계;를 포함하며,

복수의 스케쥴러 큐 중 하나의 전송 허용 시간은 다른 하나의 전송 허용 시간과는 상이한,

요구 성능 보장 방법.

청구항 2

제1항에 있어서,

상기 단계 (a)는 지정된 제1 시간 단위로 수행되고, 상기 단계 (b)는 상기 제1 시간 단위와는 상이한 제2 시간 단위로 상기 제1 시간 단위의 구간 내에서 복수 회 수행되며, 전송 허용 시간은 제2 시간 단위의 구간 내의 시간인,

요구 성능 보장 방법.

청구항 3

제1항에 있어서,

(c) 추출된 처리 요청을 디스패치 큐에 삽입하는 단계; 및

(d) 디스패치 큐의 처리 요청을 장치 큐로 순차적으로 전달하는 단계;를 더 포함하는,

요구 성능 보장 방법.

청구항 4

제3항에 있어서,

(e) 처리 요청의 처리 완료에 따라 목표 요구 사항에 대응하는 처리량을 갱신하는 단계;를 더 포함하며,

상기 단계 (a)는 반복적으로 수행되고 상기 목표 요구 사항, 이전 반복에서의 상기 처리량 및 전송 허용 시간에 기초하여 후속하는 반복에서의 전송 허용 시간을 결정하는,

요구 성능 보장 방법.

청구항 5

제1항에 있어서,

상기 단계 (b) 이전에, 복수의 가상 머신(Virtual Machine)으로부터 처리 요청을 수신하고 상기 처리 요청의 도착시각을 태깅하는 단계; 및

상기 처리 요청을 복수의 가상 머신 각각에 대응하는 스케쥴러 큐에 삽입하는 단계;를 더 포함하며,

추출된 처리 요청은 SSD(Solid State Drive)의 장치 큐로 전송되는,

요구 성능 보장 방법.

청구항 6

제5항에 있어서,

상기 단계 (b)는 각각의 스케쥴러 큐의 처리 요청에 태깅된 도착시각과 각각의 스케쥴러 큐에 대해 동적으로 결

정되는 종료시각의 비교로 처리 요청의 추출 여부를 결정하고 결정에 따라 상기 처리 요청을 추출하는,

요구 성능 보장 방법.

청구항 7

제6항에 있어서,

상기 단계 (b)는 상기 종료시각이 상기 도착시각 보다 큰 경우 상기 처리 요청을 추출하는,

요구 성능 보장 방법.

청구항 8

제4항에 있어서,

각 반복에서 결정되는 상기 전송 허용 시간은 이전 반복에서의 처리량에 반비례하여 결정되며,

상기 목표 요구 사항은 초당 I/O 처리량, 대역폭, 응답 속도 또는 SSD 사용량을 포함하는,

요구 성능 보장 방법.

청구항 9

복수의 스케쥴러 큐 각각의 전송 허용 시간을 목표 요구 사항에 기초하여 결정하는 피드백 제어 모듈; 및

각각의 결정된 전송 허용 시간 내에서 각각의 스케쥴러 큐의 처리 요청을 추출하는 I/O 스케쥴러;를 포함하며,

복수의 스케쥴러 큐 중 하나의 전송 허용 시간은 다른 하나의 전송 허용 시간과는 상이한,

요구 성능 보장 장치.

청구항 10

제9항에 있어서,

장치 큐로 출력될 처리 요청을 임시로 저장하는 디스패치 큐;를 더 포함하며,

상기 I/O 스케쥴러는 추출된 처리 요청을 디스패치 큐에 삽입하는,

요구 성능 보장 장치.

청구항 11

제9항에 있어서,

복수의 가상 머신 각각으로부터의 처리 요청을 각각 저장하는 복수의 스케쥴러 큐;를 더 포함하며,

상기 I/O 스케쥴러는 복수의 가상 머신(Virtual Machine)으로부터 처리 요청을 수신하고 상기 처리 요청의 도착 시각을 태깅하는,

요구 성능 보장 장치.

청구항 12

제11항에 있어서,

상기 I/O 스케쥴러는 각각의 스케쥴러 큐의 처리 요청에 태깅된 도착시각과 각각의 스케쥴러 큐에 대해 상기 피드백 제어 모듈에 의해 동적으로 결정되는 종료시각의 비교로 처리 요청의 추출 여부를 결정하고 결정에 따라 상기 처리 요청을 추출하는,

요구 성능 보장 장치.

청구항 13

제11항에 있어서,

상기 요구 성능 보장 장치는 복수의 가상 머신 및 공용 스토리지를 포함하는 클라우드 시스템 환경하에서 복수의 가상 머신으로부터 상기 공용 스토리지로의 I/O 처리 요청을 내장된 SSD를 이용하여 처리하는,

요구 성능 보장 장치.

발명의 설명

기 술 분 야

[0001] 본 발명은 요구 성능 보장 방법 및 요구 성능 보장 장치에 관한 것으로서, 스토리지의 특성을 고려하여 다양한 유형의 요구 성능의 공평한 보장과 스토리지 성능 향상을 동시에 이룰 수 있도록 하는, 요구 성능 보장 방법 및 요구 성능 보장 장치에 관한 것이다.

배 경 기 술

[0002] 가상화(virtualization) 기법은 널리 활용되는 클라우드 컴퓨팅(Cloud Computing)의 기반이 되는 기술로서 여러 가상 머신(Virtual Machine, 이하 'VM'이라고도 함.)이 물리적 자원을 공유할 수 있도록 하는 기술이다. 클라우드 컴퓨팅의 물리적 인프라인 클라우드 데이터 센터(Cloud Data Center, 이하 'CDC'라고도 함.)는 가상화 기술을 기반으로 물리적 자원을 가상화하여 통합하는 서버 통합(Server Consolidation) 기법을 이용한다.

[0003] 서버 통합 기술을 이용한 가상화 환경에서는 여러 가상 머신이 하나의 물리 머신 상에서 수행된다. 각각의 VM은 개별적인 성능 요구 사항을 가지고 클라우드 컴퓨팅 사용자의 계약(Service Level Agreement, 이하 'SLA'라고도 함.)으로 정형화된다. 특정 VM의 사용자는 네트워크 응답 시간, 요청 처리 속도를 성능 기준으로 제시할 수 있고 다른 사용자는 초당 I/O 처리량(I/O Operation per Second, 이하 'IOPS'라고도 함.) 또는 대역폭(Bandwidth) 등을 성능 기준으로 제시할 수 있고 또 다른 사용자는 CPU 처리량 또는 CPU 사용률이 기준이 될 수 있다.

[0004] 알려진 상용 클라우드 시스템은 데이터 가용성(Data Availability)이나 I/O 처리 성능 또는 월별 서비스 가능 시간(Monthly Uptime Percentage)을 SLA로 제시하고 있고 SLA를 준수하도록 노력한다.

[0005] 한편 CDC는 기존 기계적 하드디스크 대신에 높은 IOPS, 대역폭 및 에너지 효율적인 SSD(Solid State Drive)를 도입하고 있다. SSD의 도입 형태는 크게 세 가지로 분류될 수 있다. 첫째 기존 하드디스크와 융합된 형태의 하이브리드 저장장치 또는 디스크 어레이를 사용하는 방식이 있다.

[0006] 둘째 호스트 서버 캐쉬(Host-Side Cache)로 사용하는 방식이 있다. CDC에서는 일반적으로 수 개의 어레이로 구성된 공유 스토리지(시스템)를 사용하여 실제 데이터와 물리 서버가 분리된 형태의 스토리지 구조를 많이 사용한다. 이 스토리지 구조는 많은 데이터 처리가 수반되고 네트워크 병목 현상이나 공유 스토리지 처리 한계 등으로 인해 성능 저하로 이어진다. 이러한 문제를 호스트 서버 캐쉬를 이용하여 해결될 수 있다. 호스트 서버를 담당하는 물리 서버에 SSD를 장착하고 이 서버에서 수신되는 I/O 요청을 공유 스토리지로 전달하기 전에 먼저 내장된 SSD에서 처리하도록 하고 이에 따라 각 VM에 대해 높은 I/O 성능을 제공할 수 있다.

[0007] 마지막으로 공유 스토리지의 모든 하드디스크를 SSD로 대체하는 방식이 있다. 이 방식은 비용 문제로 인해 그 적용에 한계가 존재한다.

[0008] 호스트 서버 캐쉬 방식은 비용과 처리 대비 효율적인 방식으로서 호스트 서버 캐쉬로 사용되는 SSD에 대해 SLA를 보장하는 공평 스케쥴링 기법이 필요하다.

[0009] 호스트 서버 캐쉬로 사용되는 SSD는 여러 VM에 의해서 공유되는 자원이고 가상 머신 관리 계층인 하이퍼바이저(Hypervisor) 또는 가상 머신 모니터(Virtual Machine Monitor)가 SLA에 따라 공유 자원인 SSD를 각 VM에 할당할 책임을 가진다. 이러한 장치 관리는 가상화 시스템 구조에 따라 다양한 형태를 가질 수 있다. 예를 들어 하이퍼바이저가 하드웨어를 직접 관리하는 타입, 호스트 역할을 하는 운영체제 위에서 하이퍼바이저가 수행되는 타입 등이 있다. 두 구조 모두 SSD 상위의 I/O 스케쥴러가 존재하고 SSD 자원을 SLA에 맞추어 VM에 제공해야 하는 역할을 맡는다. 따라서 I/O 스케쥴링 기법은 SSD 활용을 위한 중요한 기술 요소이다.

[0010] I/O 스케쥴링에 관련된 몇몇 연구(비특허문헌 1 내지 4 참조)가 알려져 있다.

[0011] 비특허문헌 1의 I/O 스케쥴러는 SSD의 전체적인 성능 향상을 목표로 하고 있으나 SLA에 맞추어 각 VM의 I/O 성

능을 차등화하는 기능을 제공하고 있지 않고 읽기 요청을 수행하는 VM에 대해 좋은 성능을 제공하여 다른 VM과의 공평성을 제공하고 있지 않다.

[0012] 비특허문헌 2는 SSD의 성능 향상과 공평성 제공을 목적으로 하나 I/O 성능에 대한 SLA 보장을 위해 사용되기 어렵고 SLA를 만족시키는 형태로 변형하기 대단히 어렵다.

[0013] 비특허문헌 3의 스케쥴러와 비특허문헌 4는 공평 큐잉 기반의 알고리즘으로서 SSD의 성능 향상과 각 VM 간의 공평성 보장을 동시에 이룰 수가 없다.

[0014] 이와 같이 여러 VM으로부터의 처리 요청에 응답해서 호스트 서버상에 탑재된 SSD의 처리 성능을 향상하는 동시에 SLA 에 따른 목표 요구 사항을 만족시킬 수 있도록 하는 요구 성능 보장 방법 및 요구 성능 보장 장치가 필요하다.

선행기술문헌

비특허문헌

[0015] (비특허문헌 0001) Kim, Jaeho, et al. "Disk schedulers for solid state drivers." Proceedings of the seventh ACM international conference on Embedded software. ACM, 2009.

(비특허문헌 0002) K. Shen, “FlashFQ : A Fair Queueing I / O Scheduler for Flash-Based SSDs,” in Proccedings of the 2013 USENIX Annual Technical Conference, 2013, pp. 67-78.

(비특허문헌 0003) Gulati, Ajay, Arif Merchant, and Peter J. Varman. "mClock: handling throughput variability for hypervisor IO scheduling." Proceedings of the 9th USENIX conference on Operating systems design and implementation. USENIX Association, 2010.

(비특허문헌 0004) D. Shue, M. M. J. Freedman, and A. Shaikh, “Performance isolation and fairness for multi-tenant cloud storage,” in Proc. 10th USENIX Conference on Operating Systems Design and Implementation, 2012, pp. 349-362.

발명의 내용

해결하려는 과제

[0016] 본 발명은 상술한 문제점을 해결하기 위해서 안출한 것으로서, 사용자 SLA에 맞춘 I/O 성능 차등화를 제공하고 동시에 SSD 사용률을 높여 전체 성능을 높일 수 있도록 하는 요구 성능 보장 방법 및 요구 성능 보장 장치를 제공하는 데 그 목적이 있다.

과제의 해결 수단

[0017] 상기와 같은 목적을 달성하기 위한 요구 성능 보장 방법은 복수의 스케쥴러 큐 각각의 전송 허용 시간을 목표 요구 사항에 기초하여 결정하는 단계 및 각각의 결정된 전송 허용 시간 내에서 각각의 스케쥴러 큐의 처리 요청을 추출하는 단계를 포함하고 복수의 스케쥴러 큐 중 하나의 전송 허용 시간은 다른 하나의 전송 허용 시간과는 상이하다.

[0018] 또한 상기와 같은 목적을 달성하기 위한 요구 성능 보장 장치는 복수의 스케쥴러 큐 각각의 전송 허용 시간을 목표 요구 사항에 기초하여 결정하는 피드백 제어 모듈 및 각각의 결정된 전송 허용 시간 내에서 각각의 스케쥴러 큐의 처리 요청을 추출하는 I/O 스케쥴러를 포함하고 복수의 스케쥴러 큐 중 하나의 전송 허용 시간은 다른 하나의 전송 허용 시간과는 상이하다.

발명의 효과

[0019] 상기와 같은 본 발명에 따른 요구 성능 보장 방법 및 요구 성능 보장 장치는 사용자 SLA에 맞춘 I/O 성능 차등화를 제공하고 동시에 SSD 사용률을 높여 전체 성능을 높일 수 있도록 하는 효과가 있다.

도면의 간단한 설명

[0020] 도 1은 본 발명에 따른 예시적인 클라우드 시스템을 도시한 도면이다.

도 2는 요구 성능 보장 장치의 예시적인 하드웨어 블록도를 도시한 도면이다.

도 3은 처리 요청을 스케쥴링 하기 위한 요구 성능 보장 장치의 예시적인 기능 블록도를 도시한 도면이다.

도 4는 요구 성능을 보장하기 위한 예시적인 처리 흐름을 도시한 도면이다.

도 5는 본 발명을 설명하기 위해 이용되는 기호의 정의를 나타내는 도면이다.

도 6은 처리 요청 수신시에 스케쥴링 큐에 삽입하기 위한 알고리즘을 도시한 도면이다.

도 7은 전송 허용 시간을 결정하기 위해 윈도우 시작시에 이용되는 알고리즘을 도시한 도면이다.

도 8은 k 번째 윈도우 내의 새로운 라운드 시작시에 적용되는 알고리즘을 도시한 도면이다.

도9는 전송 허용 시간을 이용하여 처리 요청의 추출과 디스패치 큐로의 삽입을 처리하는 알고리즘을 도시한 도면이다.

도 10은 한 라운드 내에서 차등화된 전송 허용 시간을 활용하여 예시적인 VM들의 스케쥴링 큐에 적재된 처리 요청을 처리하는 일 예를 도시한 도면이다.

도 11은 목표 요구 사항의 처리량을 갱신하기 위한 알고리즘을 도시한 도면이다.

발명을 실시하기 위한 구체적인 내용

[0021] 상술한 목적, 특징 및 장점은 첨부된 도면을 참조하여 상세하게 후술 되어 있는 상세한 설명을 통하여 더욱 명확해 질 것이며, 그에 따라 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자가 본 발명의 기술적 사상을 용이하게 실시할 수 있을 것이다. 또한, 본 발명을 설명함에 있어서 본 발명과 관련된 공지 기술에 대한 구체적인 설명이 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에 그 상세한 설명을 생략하기로 한다. 이하, 첨부된 도면을 참조하여 본 발명에 따른 바람직한 실시 예를 상세히 설명하기로 한다.

[0022] 도 1은 본 발명에 따른 예시적인 클라우드 시스템을 도시한 도면이다.

[0023] 도 1에 따르면 클라우드 시스템은 복수의 VM(100)과 요구 성능 보장 장치(200) 및 공유 스토리지(300)를 포함한다. 클라우드 시스템은 CDC 상에서 구성된다. CDC는 복수의 물리 서버로 구성되고 한 유형의 물리 서버들 각각은 하나 이상의 VM(100)의 기능을 수행하기 위해 할당되고 다른 물리 서버는 호스트 서버로 할당되어 본 발명에 따른 요구 성능을 보장하기 위한 요구 성능 보장 장치(200)의 기능을 수행한다. 다른 유형의 물리 서버는 여러 하드디스크를 포함하여 공유 스토리지(300)를 제공할 수 있다. 임의의 VM(100)은 요구 성능 보장 장치(200)로 근거리 또는 광대역 네트워크를 통해 처리 요청을 전달할 수 있다.

[0024] 각 구성 요소에 대해서 간단히 살펴보면, VM(100)은 클라우드 시스템을 이용하고자 하는 사용자와 클라우드 시스템 간의 계약에 따라 특정 기능을 수행한다. VM(100)은 예를 들어 파일 서버, 웹 서버, 하둡(Hadoop) 등을 처리하는 서버 등의 기능을 수행한다. VM(100)은 클라우드 시스템 간의 계약(SLA)에 따라 개별적인 성능 요구 사항을 가진다. 각 VM(100)의 성능 요구 사항은 수행 기능에 따라 다를 수 있다. 예를 들어 성능 요구 사항은 IOPS, 대역폭, 응답 속도 또는 사용량(예를 들어 CPU 사용량이나 SSD 사용량 등)이거나 이 조합으로 구성된다. 이하 달성하고자 하는 성능 요구 사항을 목표 요구 사항이라고 지칭한다. 각각의 VM(100)은 요구 성능 보장 장치(200)로 처리 요청을 가상화 기법이 제공하는 I/O(Input/Output) 전달 방식을 통해 하드디스크(205)로 전달한다.

[0025] 요구 성능 보장 장치(200)는 VM(100) 각각의 목표 요구 사항에 따라 VM(100)으로부터의 처리 요청을 내장된 하드디스크(205)를 이용하여 처리한다. 요구 성능 보장 장치(200)는 소위 호스트 서버일 수 있다. 요구 성능 보장 장치(200)는 내장된 하드디스크(205)를 활용하여 처리 성능의 극대화와 서로 차별화되는 목표 요구 사항이 공평하게 이루어지도록 구성된다.

[0026] 요구 성능 보장 장치(200)는 내장된 SSD(하드디스크(205))를 활용하여 캐쉬 서버로서 기능하고 공유 스토리지(300)에 기록할 데이터를 임시로 버퍼링하고 VM(100)으로 처리 완료의 응답을 전송하고 이후 공유 스토리지(300)에 기록할 수 있다.

[0027] 요구 성능 보장 장치(200)에 관련해서는 도 2 이하에서 상세히 살펴보도록 한다.

[0028] 공유 스토리지(300)는 복수의 VM(100)에 의해서 공유되는 자원으로서 각종 데이터를 저장한다. 공유 스토리지(300)는 물리적으로 분산될 수 있고 요구 성능 보장 장치(200)로부터의 처리 요청에 응답하여 공유 스토리지(300) 내에 데이터를 저장하고 추출하여 전달할 수 있다.

[0029] 여기서, VM(100)에서 요구 성능 보장 장치(200)로 전송되는 처리 요청은 공유 스토리지(300)로의 I/O(Input/Output) 처리 요청을 나타내고 캐쉬 서버 기능의 수행에 따라 요구 성능 보장 장치(200)에 내장된 SSD에 대한 I/O (Input/Output) 처리를 수반한다. 내장된 SSD를 활용함으로써 I/O 처리의 성능 극대화를 가능케 한다.

[0030]

[0031] 도 2는 요구 성능 보장 장치(200)의 예시적인 하드웨어 블록도를 도시한 도면이다. 도 2에 따르면 요구 성능 보장 장치(200)는 입력 인터페이스(201), 메모리(203), 하드디스크(205), 통신 인터페이스(207), 하나 이상의 프로세서(209) 및 시스템 버스/제어 버스(211)를 포함한다. 이 중 일부의 블록은 생략될 수 있고 예를 들어 입력 인터페이스(201)는 변형예에 따라 생략될 수 있다.

[0032] 각 블록에 대해서 살펴보면, 입력 인터페이스(201)는 요구 성능 보장 장치(200)의 관리자로부터의 사용자 입력을 수신할 수 있는 인터페이스이다. 입력 인터페이스(201)는 마우스, 키보드 등을 포함하여 마우스나 키보드의 입력에 따른 신호를 수신하도록 구성된다.

[0033] 메모리(203)는 각종 데이터와 프로그램을 임시로 저장하거나 영구히 저장한다. 메모리(203)는 예를 들어 휘발성 메모리이거나 비휘발성 메모리이거나 휘발성 메모리 및 비휘발성 메모리를 포함한다.

[0034] 하드디스크(205)는 각종 데이터와 프로그램을 저장하는 대용량 저장 매체이다. 특히 이 하드디스크(205)는 SSD로 구성되어 외부 VM(100)으로부터의 처리 요청을 캐싱하기 위한 용도로 이용된다.

[0035] 하드디스크(205)는 내부에 각종 콘트롤러와 메모리를 포함한다. 메모리는 하드디스크(205) 외부로부터 수신된 I/O 처리 요청을 큐잉하기 위해서 이용된다. 메모리에 구성되는 큐는 이하 '장치 큐'라 지칭한다. 하드디스크(205) 내부의 콘트롤러는 수신된 I/O 처리 요청을 장치 큐에 삽입하고 장치 큐의 I/O 처리 요청을 내부 상태에 따라 재 스케쥴링하여 I/O 처리 요청을 처리한다. 본 발명에 따른 하드디스크(205)는 호스트 서버 캐싱 영역으로 이용된다.

[0036] 여기서 특히 SSD의 콘트롤러는 SSD의 성능 극대화를 위해 복잡한 알고리즘을 구비한다. 복잡한 알고리즘에 따라 장치 큐의 I/O 처리 요청을 재스케쥴링하여 SSD 내부의 플래쉬(Flash) 메모리들의 액세스 성능을 극대화한다. 장치 큐에 대한 설명은 도 4를 통해서 좀 더 살펴보도록 한다.

[0037] 통신 인터페이스(207)는 VM(100) 또는 VM(100)을 수행하는 물리 서버와 통신할 수 있도록 구성된 인터페이스이다. 통신 인터페이스(207)는 MAC(Media Access Control) 레이어의 네트워크 패킷을 물리 레이어 상의 아날로그 신호를 변환할 수 있다. 통신 인터페이스(207)는 예를 들어 이더넷 패킷을 아날로그 신호로 변환하여 이더넷 케이블로 출력하고 이더넷 케이블로부터 수신된 아날로그 신호를 이더넷 패킷으로 변환하여 내부로 전달할 수 있다.

[0038] 프로세서(209)는 메모리(203)와 하드디스크(205)를 활용하여 설정된 각종 기능을 수행한다. 프로세서(209)는 내부에 프로그램 코드의 명령어들을 실행하기 위한 실행 유닛(Execution Unit)을 하나 이상 포함한다.

[0039] 프로세서(209)에서 수행되는 프로그램은 여러 유형이 있을 수 있다. 예를 들어 본 발명이 적용되는 가상화 시스템의 구조에 따라 SSD와 같은 하드디스크(205)를 직접 관리하는 하이퍼바이저나 가상 머신 모니터의 프로그램을 포함하거나 나아가 운영체제 프로그램을 더 포함할 수 있다. 하이퍼바이저 또는 운영체제 프로그램은 I/O 스케쥴러(253)를 적어도 포함하고 나아가 디바이스 드라이버를 더 포함할 수 있다.

[0040] I/O 스케쥴러(253)는 외부 VM(100)으로부터 I/O 처리 요청과 같은 처리 요청을 수신하고 처리 요청에 응답하여 하드디스크(205)를 활용해서 그 처리를 수행한다. I/O 스케쥴러(253) 등에서 이루어지는 처리 과정은 도 3 이하에서 살펴보도록 한다.

[0041] 시스템 버스/제어 버스(211)는 요구 성능 보장 장치(200) 내의 구성 요소간의 데이터나 제어 신호를 송수신할 수 있는 버스이다. 시스템 버스/제어 버스(211)는 예를 들어 병렬 버스, 시리얼 버스, GPIO(General Purpose

Input/Output) 또는 그 조합으로 구성된다.

[0042] 도 3은 처리 요청을 스케쥴링 하기 위한 요구 성능 보장 장치(200)의 예시적인 기능 블록도를 도시한 도면이다. 도 3에 따르면 요구 성능 보장 장치(200)는 복수의 스케쥴러 큐(251), I/O 스케쥴러(253), 디스패치 큐(255), 피드백 제어 모듈(257) 및 요구 사항 관리 모듈(259)을 포함한다. 스케쥴러 큐(251), I/O 스케쥴러(253), 디스패치 큐(255), 피드백 제어 모듈(257) 및 나아가 요구 사항 관리 모듈(259)은 하이퍼바이저 등의 프로그램상에 포함되거나 운영체제 프로그램상에 포함된다.

[0043] 각각의 기능 블록들은 프로세서(209)상에서 수행되는 프로그램과 메모리(203)를 이용하여 바람직하게 구성된다. 각각의 기능 블록들은 스케쥴러 큐(251)나 디스패치 큐(255)를 매개로 독립적으로 수행할 수 있다. 이에 따라 I/O 스케쥴러(253)와 피드백 제어 모듈(257) 및 요구 사항 관리 모듈(259)은 내부 스케쥴링이나 이벤트 발생에 따라 독립적으로 수행된다.

[0044] 각 기능 블록들에 대해서 살펴보면, 복수의 스케쥴러 큐(251)는 복수의 VM(100)에 맵핑되어 대응하는 VM(100)으로부터의 처리 요청을 저장한다. 스케쥴러 큐(251)는 I/O 스케쥴러(253)의 제어에 따라 먼저 저장된 처리 요청이 출력될 수 있는 선입선출 방식의 리스트이다.

[0045] I/O 스케쥴러(253)는 복수의 VM(100) 각각으로부터 처리 요청을 수신하고 수신된 처리 요청을 스케쥴러 큐(251)에 선입선출 방식에 따라 삽입한다. 삽입시 I/O 스케쥴러(253)는 처리 요청을 수신한 도착시각을 처리 요청에 태깅하여 스케쥴러 큐(251) 등에 같이 저장할 수 있다.

[0046] 또한 I/O 스케쥴러(253)는 복수의 스케쥴러 큐(251) 각각에 저장된 처리 요청을 각 스케쥴러 큐(251)에 대해서 일정한 주기에 따라 동적으로 결정된 전송 허용 시간의 범위 내에서 추출하고 추출된 처리 요청을 디스패치 큐(255)에 삽입한다. 특히 I/O 스케쥴러(253)는 처리 요청에 대해 태깅된 도착시각과 아래에서 살펴볼 각 스케쥴러 큐(251)별로 일정한 주기에 따라 결정되는 전송 허용 시간의 비교로 각 스케쥴러 큐(251)의 처리 요청의 추출 여부를 결정하고 그 결정에 따라 각 스케쥴러 큐(251)의 처리 요청을 추출하고 이를 디스패치 큐(255)에 삽입한다.

[0047] 디스패치 큐(255)는 I/O 스케쥴러(253)로부터의 처리 요청을 임시로 저장한다. 디스패치 큐(255)는 선입선출 방식에 따라 먼저 입력된 처리 요청이 하드디스크(205)의 장치 큐로 출력된다. 디스패치 큐(255)의 처리 요청은 장치 큐의 콘트롤러와의 인터페이스를 통해 출력 가능한 시점에 바로 출력되도록 구성된다. 디스패치 큐(255)는 예를 들어 디바이스 드라이버 상에 구성될 수 있다.

[0048] 요구 사항 관리 모듈(259)은 입력 인터페이스(201)나 통신 인터페이스(207)를 통해 각 VM(100) 별 목표 요구 사항을 수신한다. 또한 요구 사항 관리 모듈(259)은 피드백 제어 모듈(257)에 연결되어 피드백 제어 모듈(257)의 목표 요구 사항에 대응하는 처리량을 수신하고 이를 출력 인터페이스(도면 미도시)나 통신 인터페이스(207)를 통해 출력할 수 있다.

[0049] 피드백 제어 모듈(257)은 복수의 스케쥴러 큐(251) 각각의 전송 허용 시간을 요구 사항 관리 모듈(259)로부터의 목표 요구 사항에 기초하여 결정한다. 피드백 제어 모듈(257)은 반복적으로 수행할 수 있고 각 반복에서 전송 허용 시간을 이전 반복에서의 처리량의 변화에 따라 늘리거나 줄일 수 있다.

[0050] VM(100) 별 전송 허용 시간은 목표 요구 사항에 따라서 또는 목표 요구 사항에 대응하는 처리량의 변화에 따라 VM(100) 별로 상이할 수 있다.

[0051] 도 3의 요구 성능 보장 장치(200)는 복수의 가상 머신(100), 공용 스토리지를 포함하는 클라우드 시스템 환경하에서 복수의 가상 머신(100)으로부터 공용 스토리지로의 I/O 처리 요청을 내장된 하드디스크(205)를 이용하여 처리하도록 구성된다.

[0052] 도 3의 각 기능 블록들 중 I/O 스케쥴러(253)와 피드백 제어 모듈(257)의 처리 흐름은 도 4 이하에서 상세히 살펴보도록 한다.

[0053] 도 4는 요구 성능을 보장하기 위한 예시적인 처리 흐름을 도시한 도면이다.

[0054] 도 4의 설명에 앞서, SSD와 같은 하드디스크(205)에서 성능의 극대화와 각 VM(100) 별 목표 요구 사항을 공평하

게 충족시킴에 있어 이율 배반성(trade-off)이 존재하는 점에 대해서 이하 설명하도록 한다.

[0055] VM(100) 각각의 여러 목표 요구 사항에 부합하게 사용하기 위한 기법으로서 공평 큐잉(fair-queuing) 기법이 있다. 공평 큐잉 기법의 스케쥴러로서 대표적으로 SFQ(D), BFQ, YFQ, mClock, FlashFQ 등이 있다. mClock 스케쥴러는 가상화 시스템 환경하에서 하드디스크 기반의 스토리지를 사용할 때 목표 요구 사항에 부합하는 성능 차등화 기능을 제공한다. 하지만 SSD에서 발휘할 수 있는 전체 성능을 희생하면서 이 목적을 달성하고 있다. 따라서 목표 요구 사항에 따른 성능 차등화를 달성하면서도 SSD의 전체 성능을 향상시킬 수 있는 스케쥴러의 구성이 요구된다.

[0056] 스토리지를 위한 I/O 스케쥴링 기법에서 성능과 공평성간 이율 배반성이 존재한다는 것은 널리 알려져 있다. 이율 배반성의 근본 이유는 I/O 스케쥴러(253) 내부의 스케쥴러 큐(251)와 하드디스크(205) 내에 내장된 장치 큐 간의 관계에서 기인한다.

[0057] 공평성을 담보하기 위해서 mClock과 같은 기존 스케쥴링 기법은 I/O 처리 요청을 일정한 정책에 따라 정렬하고 장치 큐로 전달되는 순서와 양을 조절하여 목표 요구 사항을 달성시킬 수 있다. 이에 따라 충분한 수준의 공평성을 제공하기 위해서는 스케쥴러 큐(251)에 가능한 많은 양의 I/O 처리 요청이 적재되어야 한다. 일반적으로 mclock 등은 네트워크 처리 요청에 적합하게 동작한다.

[0058] 반면에 하드디스크(205) 등은 디스크 헤드(disk head)를 플래터(plater) 위로 이동시켜 I/O 요청이 요구하는 위치에 접근해야만 한다. 물리적인 이동 시간으로 인해 특히 임의(random) 접근 방식의 I/O 요청에 대해 접근 시간이 늘어나는 문제가 발생한다. 이에 따라 하드디스크(205)는 장치 큐에 적재된 I/O 요청을 접근 시간을 줄일 수 있도록 재정렬하고 디스크 헤드의 이동을 최소화하여 좋은 성능을 제공할 수 있다.

[0059] 이미 살펴본 바와 스케쥴러 큐(251)에서의 공평성을 극대화하기 위해서는 충분한 양의 I/O 처리 요청이 적재되어야 하고 마찬가지로 하드디스크(205)의 처리 성능을 극대화하기 위해서도 장치 큐에 I/O 처리 요청이 충분히 적재되어야 한다. 하지만 스케쥴러 큐(251)와 장치 큐 모두에 충분한 양의 처리 요청을 적재하기는 사실상 불가능하다. 일반적으로 알려진 기업용 서버에서 I/O 요청의 길이는 초당 평균 4.87 개 정도인 것으로 알려져 있어 성능 공평성과 하드디스크(205)의 성능 극대화 공히 달성하는 것이 사실상 불가능하다.

[0060] 더욱이 하드디스크(205)가 SSD인 경우에 이율 배반성은 심화된다. 일반 하드디스크에 비해 SSD의 장치 큐의 역할은 더욱 복잡해진다. 이는 SSD의 내부 구조에 기인한다. SSD는 내부에 여러 채널로 연결된 다수의 플래쉬 메모리 칩과 콘트롤러를 내장하고 있다. SSD는 여러 독립적인 채널을 이용하여 병렬적으로 플래쉬 메모리 칩에 액세스하여 병렬성을 극대화하여 SSD의 성능을 극대화한다.

[0061] SSD의 콘트롤러는 병렬성 극대화를 위해 복잡한 계층을 가지고 복잡한 알고리즘에 따라 I/O 처리 요청을 처리한다. 병렬성 극대화를 위해서는 여러 I/O 처리 요청이 장치 큐에 존재하여야만 각 채널의 유휴 시간없이 동작할 수 있다. 이에 따라 콘트롤러는 큰 용량의 장치 큐를 구비하는 것이 필요하고 I/O 처리 요청을 가능한 많이 적재하고 있는 것이 SSD 성능 향상의 중요 요인이다. 이는 일반 물리적인 헤드를 동작하는 하드디스크(205)에 비해서 더 큰 성능 차이를 보인다.

[0062] 이상에서 살펴본 바와 같이, 하드디스크(205) 상의 장치 큐를 고려하여 목표 요구 사항에 따른 공평성 보장과 하드디스크(205)의 성능 극대화라는 두 가지 목표를 달성하기 위한 스케쥴링 기법이 요구된다.

[0063] 이러한 목표를 달성하기 위해, 도 4의 처리 흐름을 도 5 내지 도 11을 이용하여 이하 살펴본다. 먼저 도 5는 본 발명을 설명하기 위해 이용되는 기호를 나타낸다. 도 5에서 알 수 있듯이 G_i는 i 번째(i 는 2 이상) VM(100)을 나타내고, Q_i는 G_i의 스케쥴러 큐(251)를 나타내며, R_i^n은 G_i의 n 번째(n 은 0 이상) 처리 요청을 나타낸다. Tag_i^n은 R_i^n의 도착시각(arrival time)을 나타내는 태그이고 DAT^k는 k 번째 윈도우에서의 전송 허용 시간을 나타낸다. 전송 허용 시간은 이하 DAT(Differently Allowed Time) 라고도 지칭한다. Exp_i는 Q_i에 대해서 한 라운드에서의 종료시각을 나타낸다. 종료시각은 DAT와 라운드의 시작시각(시점)으로 계산된다. $Used_i^k(metric)$은 특정 목표 요구 사항에 대해서 k 번째 윈도우에서 Q_i에 대해서 계

산된 성능을 나타낸다. 이 $Used_i^k(metric)$는 k 번째 윈도우에서 지속적으로 갱신되고 이하 처리량으로 지칭될 수 있다. 메트릭(metric)은 예를 들어 IOPS, 대역폭, 응답 속도 또는 CPU(또는 SSD) 사용량 등을 지칭하고 특정 메트릭은 특정 목표 요구 사항에 대하여 한 윈도우에서의 처리량에 대응한다. $Goal_i^k(metric)$은 k 번째 윈도우에서 Q_i의 특정 메트릭에 대한 목표 요구 사항을 나타낸다. 목표 요구 사항은 적어도 IOPS, 대역폭, 응답 속도 또는 사용량이거나 이 조합으로 구성된다.

[0064] 도 4에서, 먼저 I/O 스케쥴러(253)는 복수의 VM(100)으로부터 공유 스토리지(300) 입출력(I/O)을 위한 처리 요청을 수신(S101)한다.

[0065] 처리 요청의 수신에 따라 I/O 스케쥴러(253)는 도 6의 알고리즘과 같이 현재 시각을 도착시각으로 태깅(S103)하고 처리 요청을 발송한 VM(100)에 대응하는 스케쥴러 큐(251)에 삽입(S105)한다. 스케쥴러 큐(251)에 삽입된 처리 요청들은 본 발명에 따른 스케쥴링 기법에 따라 스케쥴링되며 특히 시간 단위를 이용하여 공평성과 성능 극대화를 이룰수 있도록 한다.

[0066] 본 발명에 따른 스케쥴링 기법은 적어도 두 개의 시간 단위를 이용한다. 하나의 시간 단위(이하 '제1 시간 단위'라고도 함)는 목표 성능(요구 사항)과 처리 성능(처리량)과의 차이에 따라 목표 요구 사항을 이룰 수 있도록 전송 허용 시간을 변경하는 데 이용된다. 이러한 제1 시간 단위의 구간을 윈도우라고 지칭한다. 다른 하나의 시간 단위(이하 '제2 시간 단위'라고도 함)는 각 VM(100)별로 상이한 전송 허용 시간을 수용할 수 있도록 구성된다. 이 다른 하나의 시간 단위의 구간을 라운드라 지칭한다. 윈도우는 일반적으로 초 단위로 구성되고 라운드는 수십 밀리 초 단위로 구성된다. 시간 단위 차이로 인해 윈도우 내에는 복 수개의 라운드를 포함한다. 각 라운드 내에서 각 VM(100)별로 상이한 전송 허용 시간 동안에 각 VM(100) 별 처리 요청이 I/O 스케쥴러(253)에 의해서 수행되도록 구성된다.

[0067] 특정 윈도우(예를 들어 k 번째 윈도우)의 시작시(S107)에 피드백 제어 모듈(257)은 복수의 스케쥴러 큐(251) 각각의 전송 허용 시간을 목표 요구 사항에 기초하여 결정(S109)한다. 아래 수학식 1은 VM(100) 각각에 대해 한 윈도우 내에서 이용될 전송 허용 시간을 결정하기 위한 기본 식을 나타낸다.

수학식 1

[0068]
$$DAT_i^k = \frac{Goal_i^k(metric)}{Used_i^{k-1}(metric)} \times DAT_i^{k-1}$$

[0069] 수학식 1과 같이 복수의 스케쥴러 각각의 전송 허용 시간(DAT)은 목표 요구 사항(Goal)과 이전 윈도우(k-1)에서의 전송 허용 시간 및 이전 윈도우에서의 처리량($Used_i^{k-1}(metric)$)에 기초하여 결정된다. 특히 단계 S109는 윈도우의 시작 시점마다 반복적으로 수행되고 이전 윈도우에서의 처리량에 반비례하여 전송 허용 시간을 결정한다. 각 스케쥴러 큐(251)별 전송 허용 시간은 수학식 1에 따라 다른 하나의 전송 허용 시간과 상이할 수 있다.

[0070] 도 7은 전송 허용 시간을 결정하기 위해 윈도우 시작시에 이용되는 알고리즘을 도시하고 있다. 도 7에서 알 수 있는 바와 같이 이 알고리즘은 다수의 반복문으로 구성된다. 라인 1에서 라인 8은 수학식 1을 이용하여 후보 전송 허용 시간을 계산하는 방법을 도시하고 있다. 이 후보 전송 허용 시간이 라운드 내에서 각 VM(100) 별 전송 허용 시간이 될 수도 있다.

[0071] 목표 요구 사항은 절대치로 또는 다른 VM(100)과의 상대치(weight)로 설정될 수 있다. 라인 9 내지 33은 이러한 절대치와 상대치가 동시에 설정될 때를 고려하여 전송 허용 시간을 계산하는 것을 나타낸다.

[0072] 예를 들어 VM(100)이 3 개가 있고, 각 VM(100) 별 요구 성능 비율이 1:1:1이고 하나의 VM(100)이 절대치 50MByte/S가 성능 요구 있는 경우에, 전체 성능이 300 MByte/S인 경우 성능 비율에 따라 100MByte 씩 할당받을 수 있다. 만일 절대치를 고려한다면 이 VM(100)은 수학식 1에 따라 50MByte/S에 맞추도록 전송 허용 시간이 낮아질 수 있다. 이러한 절대치와 상대치를 고려하여 라인 9 내지 33은 절대치로 계산된 전송 허용 시간과 상대치

로 계산된 전송 허용 시간을 비교하여 더 큰 전송 허용 시간을 할당하도록 한다. 참고로 라인 4와 라인 16의 'equation 1'은 수학식 1을 나타낸다.

[0073] 단계 S109에서 피드백 제어 모듈(257)은 또한 각종 변수 등을 초기화한다. 예를 들어 피드백 제어 모듈(257)은 목표 요구 사항의 특정 메트릭에 대한 처리량(도 7의 라인 34 참조)을 나타내는 변수를 초기화한다.

[0074] 단계 S109는 피드백 제어 모듈(257)에 의해서 윈도우의 주기에 따라 반복적으로 수행되고 각 반복에서 결정되는 전송 허용 시간은 이전 반복에서의 전송 허용 시간과는 다를 수 있다.

[0075] 윈도우 내에는 복수의 라운드를 포함하는 데, 라운드의 길이(시간)은 VM(100)들의 DAT 중 가장 긴 시간으로 결정된다. 라운드 길이의 결정은 I/O 스케쥴러(253)에 의해서 수행될 수도 있고 피드백 제어 모듈(257)에 의해서 수행될 수도 있다. I/O 스케쥴러(253)는 만일 라운드 시작 시점(S111)인 경우 복수의 스케쥴러 큐(251) 각각을 위한 종료시각을 설정(단계 S113)한다.

[0076] 도 8은 k 번째 윈도우 내의 새로운 라운드 시작시의 알고리즘을 도시하고 있는 데, I/O 스케쥴러(253)는 k번째 라운드의 스케쥴러 큐(251) 각각의 종료시각을 피드백 제어 모듈(257)에 의해서 윈도우 시작시에 계산된(결정된) 전송 허용 시간을 현재시각에 더하여 계산한다. 이와 같이 종료시각은 전송 허용 시간을 이용하여 시간의 경과에 따라 동적으로 결정된다. 도 8은 종료시각의 설정 이후에 I/O 스케쥴러(253)를 호출(도 8의 라인 6, 도9의 스케쥴러 모듈)하고 디스패치 큐(255)의 처리 요청을 하드디스크(205)의 장치 큐로 전달(도 8의 라인 7)할 수 있다.

[0077] 이후 I/O 스케쥴러(253)는 복수의 스케쥴러 큐(251)에 적재된 처리 요청들을 각 스케쥴러 큐(251)에 대해서 결정된 전송 허용 시간 내에서 추출(단계 S115)하고 추출된 처리 요청을 디스패치 큐(255)에 삽입(단계 S117)한다. 도 9는 전송 허용 시간을 이용하여 처리 요청의 추출과 디스패치 큐(255)로의 삽입을 처리하는 알고리즘을 나타낸다. 도 9에서 알 수 있는 바와 같이 I/O 스케쥴러(253)는 스케쥴러 큐(251) 각각의 처리 요청들을 태깅된 도착시간과 종료시각의 비교로 추출 여부를 결정하고 그 결정에 따라 해당 처리 요청을 추출한다. 만일 스케쥴러 큐(251)의 종료시각이 처리 요청의 도착시간보다 큰 경우에 이 처리 요청을 추출되고 그렇지 않은 경우에는 다음 라운드에서 추출될 수 있다. 단계 S115와 단계 S117은 반복적으로 수행되며 한 윈도우 내에서 라운드의 주기에 따라 반복적으로 복 수회 수행된다.

[0078] 도 10은 이러한 예를 도시한 도면인데, 라운드의 길이는 복수의 VM(100) 들의 전송 허용 시간 들 중 가장 긴 시간으로 결정된다. 전송 허용 시간은 윈도우별로 갱신되므로 이 라운드 길이 또한 윈도우별로 변경될 수 있다. 각각의 전송 허용 시간 DAT는 한 라운드 내에서의 시간을 나타낸다.

[0079] 도 10의 예에서 알 수 있는 바와 같이, 라운드의 길이는 가장 긴 스케쥴러 큐(251) Q_1의 전송 허용 시간으로 결정되고, I/O 스케쥴러(253)는 스케쥴러 큐(251)의 각 처리 요청의 도착시각과 종료시각을 비교하여 추출 여부를 결정한다. Q_2나 Q_3 는 종료시각이 도과함에 따라 특정 처리 요청이 다운 라운드에서 추출된다.

[0080] 이후 단계 S119에서 디스패치 큐(255)의 처리 요청들은 하드디스크(205)로의 전송 가능한 시기에 하드디스크(205)의 장치 큐로 바로 전송된다. 하드디스크(205)는 SSD 일 수 있고 단계 S119는 I/O 스케쥴러(253)에 의해서 수행되거나 또는 하드디스크(205)를 제어할 수 있는 디바이스 드라이버에 의해서 수행될 수 있다.

[0081] 단계 S119 이후에 하드디스크(205)는 장치 큐의 처리 요청을 처리하고 처리 완료를 나타내는 응답을 피드백 제어 모듈(257)로 반환한다. 피드백 제어 모듈(257)은 특정 처리 요청에 대한 처리 완료에 따라 목표 요구 사항에 대응하는 처리량을 갱신(S121)한다.

[0082] 도 11은 목표 요구 사항의 처리량을 갱신하기 위한 알고리즘을 도시하고 있다. 도 11에서 알 수 있는 바와 같이 대역폭, IOPS, 응답 속도(latency) 및 사용량(utilization) 등이 갱신된다. 이러한 처리량은 한 윈도우내에서 계속 갱신될 수 있고 이후 윈도우 반복시에 전송 허용 시간의 계산에 이용된다.

[0083] 사용량(utilization)은 하나의 처리 요청에 의해서 SSD에서의 사용량을 나타내고 다양한 방식으로 정의될 수 있다. 일반적으로 사용량은 I/O 처리 요청의 크기에 비례하여 결정된다.

[0084] 도 4의 단계 S101 내지 단계 S121은 반복적으로 수행되며 특히 단계109는 윈도우 주기에 따라 반복적으로 수행되고 단계 S113은 윈도우 주기 내에서 라운드의 주기에 따라 복수회 반복적으로 수행된다. 나아가 적어도 단계 S115 내지 단계 S117 또한 라운드 내의 각각의 전송 허용 시간 내에서 라운드별로 반복적으로 수행된다.

[0085] 이상 도 4의 처리 흐름을 통해, 요구 성능 보장 장치(200)는 VM(100) 각각의 목표 요구 사항을 전송 허용 시간을 이용하여 공평하게 차등화할 수 있고 디스패치 큐(255)의 처리 요청을 동시에 하드디스크(205)로 출력하여 SSD의 처리 성능을 극대화할 수 있다.

[0086] 이상에서 설명한 본 발명은, 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에게 있어 본 발명의 기술적 사상을 벗어나지 않는 범위 내에서 여러 가지 치환, 변형 및 변경이 가능하므로 전술한 실시 예 및 첨부된 도면에 의해 한정되는 것이 아니다.

부호의 설명

[0087] 100 : 가상 머신 200 : 요구 성능 보장 장치

201 : 입력 인터페이스 203 : 메모리

205 : 하드디스크 207 : 통신 인터페이스

209 : 프로세서 211 : 시스템 버스/제어 버스

251 : 스케쥴러 큐 253 : I/O 스케쥴러

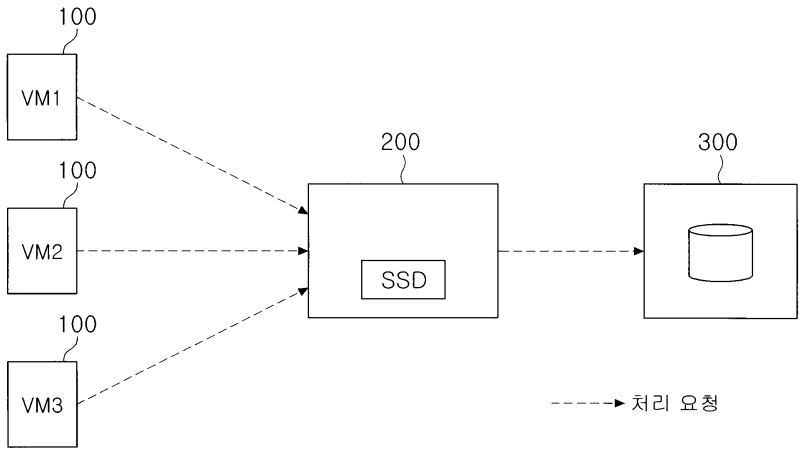
255 : 디스패치 큐 257 : 피드백 제어 모듈

259 : 요구 사항 관리 모듈

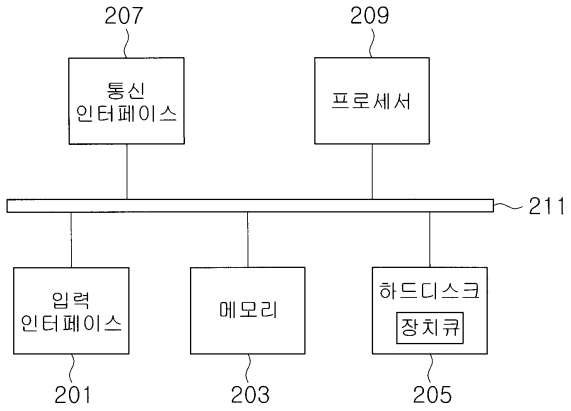
300 : 공유 스토리지

도면

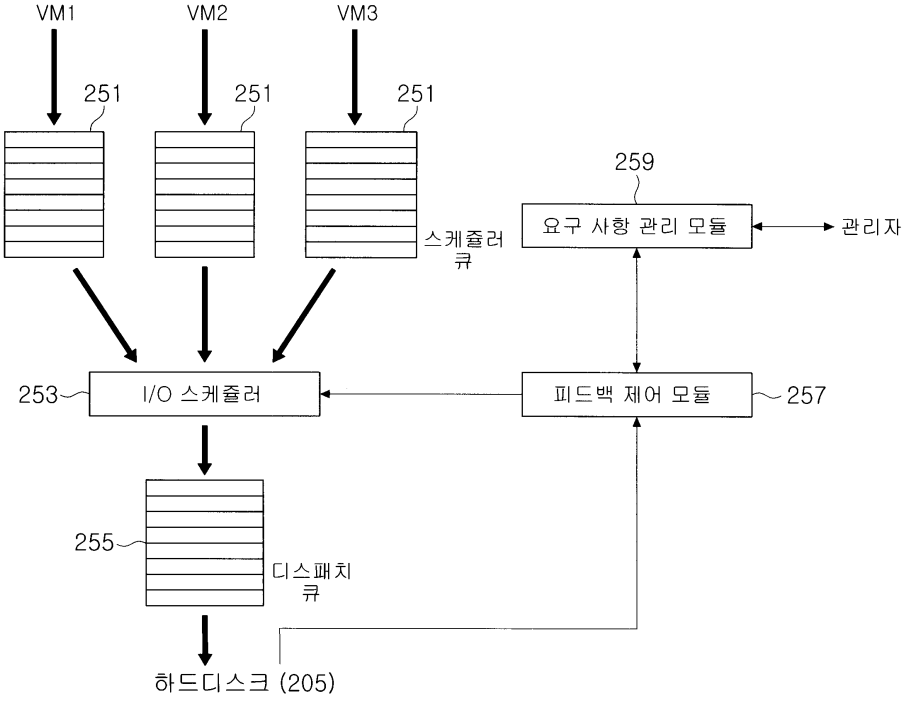
도면1



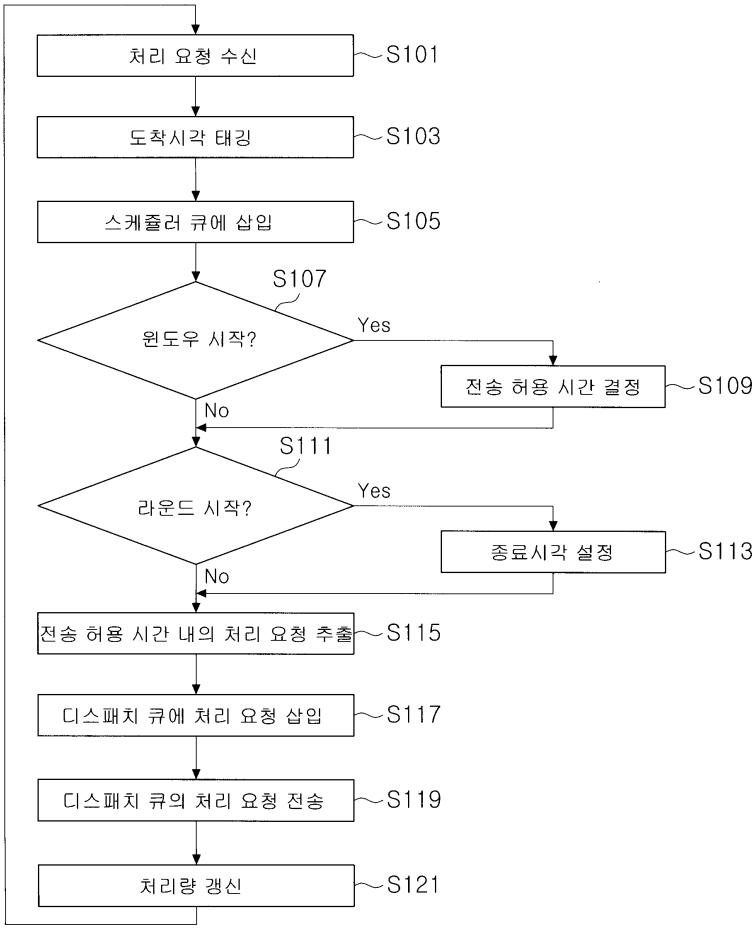
도면2


207
209
통신
인터페이스
프로세서
211
입력
인터페이스
메모리
하드디스크
장치큐
201
203
205

도면3


VM1
VM2
VM3
251
251
251
스케쥴러
큐
259
요구 사항 관리 모듈
관리자
253
I/O 스케쥴러
피드백 제어 모듈
257
255
디스패치
큐
하드디스크 (205)

도면4



도면5

기호	정의
G_i	the i-th guest VM
Q_i	The G-queue for G_i
R_i^n	n-th request of G_i
Tag_i^n	Tag of the request R_i^n
DAT_i^k	DAT of Q_i in the k-th window
Exp_i	Expiration time of Q_i in a round
$Used_i^k(metric)$	An amount of resource used for Q_i in the k-th window, is presented in $metric$
$Goal_i^k(metric)$	A goal for an amount of resource used for Q_i in the k-th window, is presented in $metric$

도면6

Algorithm : On arrival of R_i^n

1: Tag_i^n = current time
2: Enqueue R_i^n to Q_i

도면7

Algorithm : At start of k-th window

Variables: :
$metps$: The metric that is used for proportional sharing
$SetPS$: The set of G-queues for proportional sharing
$Total$: The total values of resources
$TotalWeight$: The total weights of Q_i
$CandiDAT_i$: The candidate PAT for Q_i
$Iter$: The number of iteration

/* Check and setup for QoS requirement */
1: **for** all Q_i **do**
2: **if** (Q_i has QoS requirement) **then**
3: Set $Goal_i^k(metric\ for\ the\ QoS\ requirement)$
4: Set $CandiDAT_i$ using the equation 1
5: **else**
6: $CandiDAT_i$ = 0
7: **end if**
8: **end for**
/* Proportional sharing starts */
9: $Iter$ = Number of G-queues with QoS requirement
10: $Total$ = Sum of all $USED_i^{k-1}(metps)$
11: $SetPS$ = All Q_i
12: **while** ($ITER \geq 0$) **do**
13: $TotalWeight$ = Sum of all weights of Q_i in $SetPS$
14: **for** Q_i in $SetPS$ **do**
15: $Goal_i^k(metps) = Total \times \frac{weight\ of\ Q_i}{TotalWeight}$
16: Set DAT_i^k using the equation 1
17: **end for**
/* Comparison with candidate DATs */
18: **for** all Q_i **do**
19: **if** (Q_i has a reservation and $DAT_i^k < CandiDAT_i$) **then**
20: $DAT_i^k = CandiDAT_i$
21: $Total = Total - \frac{CandiDAT_i}{DAT_i^{k-1}} \times Used_i^{k-1}(metps)$
22: Exclude Q_i from $SetPS$
23: Break this for-loop
24: **end if**
25: **if** (Q_i has a limitation and $DAT_i^k > CandiDAT_i$) **then**
26: $DAT_i^k = CandiDAT_i$
27: $Total = Total - \frac{CandiDAT_i}{DAT_i^{k-1}} \times Used_i^{k-1}(metps)$
28: Exclude Q_i from $SetPS$
29: Break this for-loop
30: **end if**
31: **end for**
32: Decrease $ITER$ by 1
33: **end while**
34: Set all $Used_i^k(metric)$ to 0
35: **repeat**
36: Start a new round
37: **until** (the end of k-th window)

도면8

Algorithm : At start of a round in k-th window

Variables: :
$CurrTime$: A variable to store the start time of next round

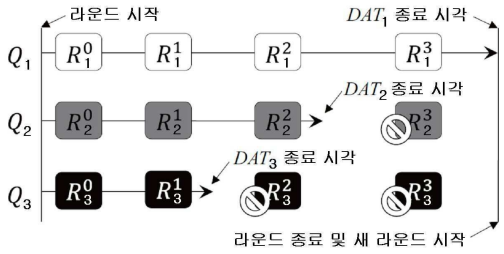
1: $CurrTime$ = current time
2: **for** all Exp_i **do**
3: $Exp_i = CurrTime + DAT_i^k$
4: **end for**
/* a next round starts */
5: **while** (the longest Exp_i < current time) **do**
6: Run the scheduler module
7: Flush the dispatch queue as much as possible
8: **end while**

도면9

Algorithm : The scheduler module

1: **for all** Q_i **do**
2: **for all** R_i^n in Q_i **do**
3: **if** $(Tag_i^n < Exp_i)$ **then**
4: Move R_i^n to dispatch queue
5: **end if**
6: **end for**
7: **end for**

도면10



도면11

Algorithm : On completion of R_i^n at k-th window

1: Increase $Used_i^k(bandwidth)$ by requested bytes of R_i^n
2: Increase $Used_i^k(iops)$ by 1
3: Increase $Used_i^k(latency)$ by a reciprocal of (current time - Tag_i^n)
4: Increase $Used_i^k$ (**utilization**) by get-**utilization**(R_i^n)