(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2020-0077115
(43) 공개일자 2020년06월30일

(51) 국제특허분류(Int. Cl.)
G06F 9/455 (2018.01)
(52) CPC특허분류
G06F 9/45558 (2013.01)
G06F 2009/4557 (2019.08)
(21) 출원번호 10-2018-0166315
(22) 출원일자 2018년12월20일
심사청구일자 없음

(71) 출원인
경희대학교 산학협력단
경기도 용인시 기흥구 덕영대로 1732 (서천동, 경희대학교 국제캠퍼스내)
(72) 발명자
허의남
경기도 용인시 기흥구 이현로29번길 86-21 201동401호(보정동,e편한세상대림아파트)
용찬호
경기도 수원시 권선구 수성로 47, 7동 1301호
(74) 대리인
김홍석

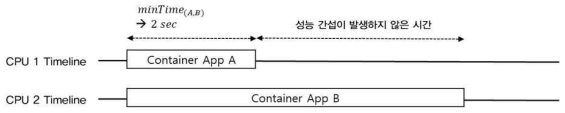
전체 청구항 수 : 총 1 항

(54) 발명의 명칭 **컨테이너 가상화 환경에서의 성능 간섭 최소화를 통한 최적의 처리 시간 보장 방법**

(57) 요 약

본 발명의 실시예에 따른 컨테이너 가상화 환경에서의 성능 간섭 최소화를 통한 최적의 처리 시간 보장 방법은 성능간섭수치를 측정하는 단계; 및 컨테이너 요청 배치단계를 포함한다.

대 표 도 - 도1



(52) CPC특허분류
G06F 2009/45591 (2019.08)

이 발명을 지원한 국가연구개발사업
과제고유번호 1711070399
부처명 과학기술정보통신부
연구관리전문기관 정보통신기술진흥센터
연구사업명 대학ICT연구센터육성지원사업
연구과제명 실시간 모바일 클라우드 서비스 플랫폼 연구개발
기 여 율 1/1
주관기관 경희대학교 산학협력단
연구기간 2018.01.01 ~ 2018.12.31

명 세 서

청구범위

청구항 1

컨테이너 가상화 환경에서의 성능 간섭 최소화를 통한 최적의 처리 시간 보장 방법으로서,

성능간섭수치를 측정하는 단계; 및

컨테이너 요청 배치단계를 포함하는 것을 특징으로 하는 컨테이너 가상화 환경에서의 상능 간섭 최소화를 통한 최적의 처리 시간 보장방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 컨테이너 가상화 환경에서의 성능 간섭 최소화를 통한 최적의 처리 시간 보장방법에 관한 것이다.

배 경 기 술

[0002] 서버 클러스터에서 수행되는 기존의 컨테이너 배치 방법은 대부분 서버 가용 자원, CPU 및 저장 장치 등의 하드웨어 종류, 컨테이너 애플리케이션의 종류 친화성 등을 기준으로 수행된다. 그러나 여러 개의 컨테이너들은 커널 등과 같은 호스트의 자원들을 공유해 사용하며, 이로 인해 컨테이너 간의 간섭이 발생해 성능이 저하될 수 있고, 컨테이너 애플리케이션의 처리 시간이 늘어날 수 있다는 문제점이 존재한다. 호스트의 공유 자원에 대한 격리는 컨테이너 가상화 기술에서 지원되지 않을 뿐만 아니라, 성능 간섭에 해당되는 공유 자원의 종류가 많고 다양해 이에 종합적으로 대응하기 어렵고, 다양한 컨테이너 애플리케이션 간의 성능 간섭 정도가 각각 상이하기 때문에, 이를 보완하기 위한 방법이 요구된다.

발명의 내용

해결하려는 과제

[0003] 본 발명은, 컨테이너 가상화를 사용하는 클라우드 환경에서, 단일 호스트 내에서 발생하는 복수 컨테이너 간 성능 간섭을 최소화해, 주어진 환경에서 최적의 처리 시간을 수행할 수 있도록 하는 기술을 제안한다.

과제의 해결 수단

[0004] 본 방법은 컨테이너의 처리 시간을 최소화하기 위해, 주어진 컨테이너 애플리케이션 간의 성능 간섭 수치를 측정하고, 이를 기반으로 각 컨테이너의 예상 수행 시간을 계산함으로써 성능 간섭으로 인한 처리 시간 증가 현상을 최소화한다. 이를 위해 본 방법은 총 두 단계로 구성되며, 이는 각각 (1) 다양한 컨테이너 애플리케이션 조합으로 인한 성능 간섭 수치를 측정하는 단계, (2) 이를 활용해 주어진 서버 클러스터 중 현재 요청된 컨테이너의 처리 시간이 최소화되거나, 전체 컨테이너 처리 시간이 최소화될 것으로 예상되는 서버를 선택해 요청을 할당하는 단계로 구성된다. 두 번째 단계에서는, 현재 요청된 컨테이너 애플리케이션의 종류와, 각 서버에서 기존에 실행 중인 컨테이너 애플리케이션의 종류를 서로 비교함으로써, 현재 요청된 컨테이너의 예상 수행 시간 및 기존에 실행 중인 컨테이너의 추가 수행 시간을 예측한다.

발명의 효과

[0005] 본 방법은 처리 시간에 민감한 서비스를 컨테이너 기반의 클라우드에서 연속적으로 요청 및 할당해 사용해야 할 때, 컨테이너 간 성능 간섭으로 인한 추가 수행 시간을 최소화함으로써 원래의 성능과 최대한 비슷한 수준의 성능을 제공할 수 있고, 이를 통해 주어진 처리 시간 내에 애플리케이션이 수행될 수 있도록 한다. 응답 시간에 민감한 V2X, 5G, 자동화 공정, IoT 환경 등에서 컨테이너 수준의 가상화를 차용해 사용할 때 본 방법을 적용할 수 있으며, 컨테이너 간의 성능 간섭을 최소화해 추가적인 처리 시간을 감소시킴으로써 서비스의 응답시간을 줄일 수 있을 것으로 예상된다.

도면의 간단한 설명

[0006] 도 1은 CPU에서 실행되는 컨테이너 애플리케이션의 타임라인이다.

도 2는 성능 간섭치 계산방법이다.

도 3은 컨테이너 애플리케이션의 예상 수행시간 계산방법이다.

도 4는 기존에 실행중인 컨테이너의 수행시간 갱신결과이다.

도 5는 컨테이너 요청의 배치 전략의 적용예시 개념도이다.

발명을 실시하기 위한 구체적인 내용

[0007] 상술한 본 발명의 특징 및 효과는 첨부된 도면과 관련한 다음의 상세한 설명을 통하여 보다 분명해 질 것이며, 그에 따라 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자가 본 발명의 기술적 사상을 용이하게 실시할 수 있을 것이다.

[0008] 본 발명은 다양한 변경을 가할 수 있고 여러 가지 실시 예를 가질 수 있는바, 특정 실시 예들을 도면에 예시하고 상세한 설명에 구체적으로 설명하고자 한다. 그러나 이는 본 발명을 특정한 실시 형태에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물 내지 대체물을 포함하는 것으로 이해되어야 한다.

[0009] 각 도면을 설명하면서 유사한 참조부호를 유사한 구성요소에 대해 사용한다.

[0010] 제1, 제2등의 용어는 다양한 구성요소들을 설명하는데 사용될 수 있지만, 상기 구성요소들은 상기 용어들에 의해 한정되어서는 안 된다. 상기 용어들은 하나의 구성요소를 다른 구성요소로부터 구별하는 목적으로만 사용된다.

[0011] 예를 들어, 본 발명의 권리 범위를 벗어나지 않으면서 제1 구성요소는 제2 구성요소로 명명될 수 있고, 유사하게 제2 구성요소도 제1 구성요소로 명명될 수 있다. 및/또는 이라는 용어는 복수의 관련된 기재된 항목들의 조합 또는 복수의 관련된 기재된 항목들 중의 어느 항목을 포함한다.

[0012] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미가 있다.

[0013] 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 의미를 가지는 것으로 해석되어야 하며, 본 출원에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않아야 한다.

[0014] 이하의 설명에서 사용되는 구성요소에 대한 접미사 모듈, 블록 및 부는 명세서 작성의 용이함만이 고려되어 부여되거나 혼용되는 것으로서, 그 자체로 서로 구별되는 의미 또는 역할을 갖는 것은 아니다.

[0015] 이하, 본 발명의 바람직한 실시 예를 첨부한 도면을 참조하여 당해 분야에 통상의 지식을 가진 자가 용이하게 실시할 수 있도록 설명한다. 하기에서 본 발명의 실시 예를 설명함에 있어, 관련된 공지의 기능 또는 공지의 구성에 대한 구체적인 설명이 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명을 생략한다.

[0016] 이하에서는, 본 발명에 따른 컨테이너 가상화 환경에서의 성능 간섭 최소화를 통한 최적의 처리 시간 보장 방법에 대해 살펴보기로 한다.

[0018] 컨테이너는 OS 레벨의 가상화를 적용함으로써 경량화된 가상 공간을 생성하는 가상화 기술이다. 컨테이너는 기존의 가상화 기술인 가상 머신과 달리 완벽한 운영 체제를 갖추지 않으며, 호스트 머신과 커널을 공유해 사용하고, 프로그램 실행에 필요한 라이브러리만을 갖춘 가상 공간을 사용한다.

[0019] 컨테이너, 예를 들어 Docker (도커) 와 같은 오픈소스 프로젝트는 멀티 테넌시를 제공하기 위해 호스트의 자원을 가상화해 컨테이너 별로 할당한다. 이 때 cgroup과 같은 리눅스 커널에 내장된 자원 분리 기술을 컨테이너 기술에 적용할 수 있다. cgroup는 메모리, CPU Block I/O 등의 자원을 각 컨테이너에게 독립적으로 할당하는 기능을 제공한다.

[0020] cgroup 기반의 자원 가상화는 가상 머신에서 사용되는 하이퍼바이저와 달리, 커널에 내장되어 있는 자원 격리 기술이다. cgroup에서 제어되지 않는 호스트 자원, 예를 들어 CPU의 캐시 (L1, L2, LLC 등), 메모리 대역폭, I/O 장치 등은 모든 컨테이너가 공유해 사용하며, 특정 컨테이너가 공유 자원을 과도하게 사용하거나 점유할 경우, 해당 공유 자원을 사용하는 다른 컨테이너의 성능이 낮아지는 현상이 발생할 수 있다. 이로 인해 컨테이너 애플리케이션의 처리 속도가 낮아짐에 따라 서비스 요청의 응답시간 또한 증가하고, 전체적인 서비스의 품질이 낮아질 수 있다.

[0021] 이를 해결하기 위해, 본 방법은 주어진 컨테이너 애플리케이션 간의 성능 간섭 수치를 측정하고, 해당 수치 값들을 기반으로 컨테이너 요청을 최적의 서버에 할당하는 스케줄링을 진행함으로써 컨테이너의 처리 시간을 최적화하는 방법을 제안한다.

[0022] 본 방법에서는, 컨테이너 애플리케이션의 종류가 N개로 주어졌다고 가정하며, 각 컨테이너의 공유 자원 사용 정도는 컨테이너의 실행 시간 (요청 처리 시간) 동안 동일하다고 가정한다. 또한, 컨테이너 성능 간섭의 주요 원인으로 지목되는 CPU 캐시 미스 공유(메모리 대역폭)을 최소화하기 위해, 각 컨테이너는 cgroup을 통해 배타적으로 할당된 1개의 CPU Affinity를 사용함을 가정한다. 또한, 하나의 컨테이너는 하나의 애플리케이션만을 배타적으로 실행한다.

[0023] 이하 본 발명의 실시예에 따른 컨테이너 가상화 환경에서의 성능 간섭 최소화를 통한 최적의 처리 시간 보장 방법을 도면을 이용하여 상세히 설명한다.

[0024] 도 1은 CPU에서 실행되는 컨테이너 애플리케이션의 타임라인이다.

[0025] 도 2는 성능 간섭치 계산방법이다.

[0026] <1단계 - 성능간섭 수치 측정단계>

[0027] 성능 간섭을 계산하기 위해, 복수 개의 컨테이너 애플리케이션을 동시에 실행해 추가로 실행된 시간을 계산한 뒤, 해당 시간 값을 성능 간섭을 받지 않은 실행 시간으로 나누어 계산한다. 도 1, 도 2는 두 종류의 컨테이너 애플리케이션을 동시에 실행한 뒤, 성능 감소치를 계산하는 방법을 나타낸다.

[0028] 도 2의 '컨테이너 애플리케이션 당 기준 소요 시간' 은 해당 컨테이너가 성능 간섭 없이 실행될 때의 원래 처리 시간을 의미한다. 이 때, 도 1과 같이 A, B를 컨테이너로서 동시에 실행했을 때 각 컨테이너에서 추가로 실행된 시간 만큼을 계산하고, 해당 시간 만큼을 원래의 실행 시간으로 나누어 성능 간섭치를 계산한다. 예를 들어 A의 원래 수행 시간이 2초이고, A와 B를 동시에 실행했을 때 2.2초가 소요되었다면 0.2 / 2 = 0.1, 즉 10%의 성능 간섭이 발생하였으며, 성능 간섭 수치는 0.1로 계산된다. 이 때, 성능 간섭이 일어난 구간은 두 컨테이너가 동시에 실행된 시간에 한정되므로, 각 컨테이너 애플리케이션 중 최소의 실행시간: minTime 값을 구해 함께 저장하며, 성능 간섭이 발생하지 않은 구간에 대해서는 고려하지 않는다. 이와 같은 값들을 통해, 아래와 같은 성능 간섭 레코드를 생성한다.

[0029] **레코드 형식 :** {[애플리케이션 종류의 조합], 타겟 애플리케이션 종류, minTime, 성능 간섭 수치}

[0030] **데이터 예시 :** {[A,B], A, 2.0, 0.1 }

[0031] - 위 예시의 값은 각각 아래의 뜻을 의미한다.

[0032] (1) A와 B가 동시에 실행되었을 때 : **[A,B],**

[0033] (2) A 애플리케이션에 대해서 : **A,**

[0034] (3) 2.0초 동안 (minTime) : **2.0**

[0035] (4) 0.1 (10%) 만큼의 성능 간섭을 받았다 : **0.1**

[0036] - 단, 여러 종류의 애플리케이션이 동시에 실행될 경우 각 애플리케이션 종류에 대한 성능 간섭의 정도는 모두 다르기 때문에 위의 예시 중 (2)에서 특정 애플리케이션에 대한 타겟을 나타내었다. 이와 같은 방법으로, 도 2에서 타겟 애플리케이션이 B인 경우에 대해서도 아래와 같이 나타낼 수 있다.

[0037] **데이터 예시 :** {[A,B], B, 2.0, 0.05 }

[0038] - 위 예시의 값은 각각 아래의 뜻을 의미한다.

[0039] (1) A와 B가 동시에 실행되었을 때 : **[A,B],**

[0040] (2) B 애플리케이션에 대해서 : **B,**

[0041] (3) 2.0초 동안 (minTime) : **2.0**

[0042] (4) 0.05 (5%)만큼의 성능 간섭을 받았다 : **0.05**

[0044] - 이러한 방식을 통해, 주어진 모든 컨테이너 애플리케이션 조합에 대해 성능 간섭치를 계산해 데이터로서 저장한다. 예를 들어, 네 종류의 컨테이너 애플리케이션이 주어졌을 때, 출력되는 성능 간섭치의 경우는, 중복 조합인 H(4,2): 10, + H(4,3): 20, + H(4,4): 35 = 65가지의 레코드가 출력된다. 이 레코드에는, (1) 어떠한 애플리케이션 조합이 함께 실행되었을 때, (2) 특정 애플리케이션에 (3) 몇 초 동안 (4) 얼마나 성능 간섭이 발생했는지에 대한 정보가 담겨 있다.

[0045] - 각 중복 조합: H(n,r) 은 N개의 컨테이너 애플리케이션 종류가 주어졌을 때, r개의 컨테이너 애플리케이션이 동시에 실행 될 경우의 레코드 개수를 나타낸다. 각 경우의 예시는 아래와 같다.

[0047] - H(4,2) 의 경우에 대한 레코드 예시는 {[A,B], B, 2.0, 0.05 } 와 같다.

[0048] - H(4,3) 의 경우에 대한 레코드 예시는 {[A,B,C], B, 2.0, 0.15 } 와 같다.

[0049] - H(4,4) 의 경우에 대한 레코드 예시는 {[B,B,B,C], C, 2.0, 0.25} 와 같다.

[0050] <2단계 - 컨테이너 요청 배치 단계>

[0051] 도 3은 컨테이너 애플리케이션의 예상 수행시간 계산방법이다.

[0052] 도 4는 기존에 실행중인 컨테이너의 수행시간 갱신결과이다.

[0053] 도 5는 컨테이너 요청의 배치 전략의 적용예시 개념도이다.

[0054] 본 발명의 실시예에 따른 컨테이너 요청 배치단계는 모든 경우에 대해 성능 간섭 수치를 얻은 뒤, 컨테이너 요청에 대한 배치 스케줄링을 진행할 수 있다. 컨테이너 요청이 들어올 경우, 서버 클러스터 내에서 각 서버에 대한 성능 간섭률 (예상 수행 시간, 또는 추가 수행 시간)을 계산한 뒤, 특정 전략을 사용해 컨테이너 요청을 할당한다.

[0055] 2-1과정 : 새로 요청된 컨테이너의 수행시간 예측

[0056] 클러스터 내의 모든 서버에 대해 새로 요청된 컨테이너의 예상 요청 시간을 계산한다. 새로 요청된 컨테이너의 예상 수행 시간을 계산하기 위해, 첫 번째 단계에서 출력된 성능 간섭 수치 데이터: 특정 조합의 컨테이너 애플리케이션이 함께 수행될 때의 성능 간섭 수치를 사용하며, 이에 대한 예시는 도 3과 같다.

[0057] 2-2과정 : 각 서버에서 기존에 실행중인 컨테이너의 예측 수행시간 갱신

[0058] 새로운 컨테이너 요청이 특정 서버에 할당될 경우, 새로운 컨테이너의 실행으로 인해 기존에 실행 중이던 컨테이너 또한 성능 저하가 발생한다. 따라서 새로 요청된 컨테이너의 예상 수행 시간을 각 서버에 대해 계산함과 동시에, 해당 서버에서 기존에 이미 실행 중인 컨테이너의 연장되는 추가 수행 시간을 함께 예측해 갱신한다. 새로운 컨테이너 요청 시, 기존에 이미 실행 중인 컨테이너의 연장되는 수행 시간의 예시는 도 4와 같다.

[0059] 2-3과정 : 컨테이너 요청의 배치전략

[0060] - 즉, 새로운 컨테이너에 대한 요청이 도착할 경우, 클러스터 내의 모든 서버에 대해 두 가지 값을 미리 계산할 수 있으며, 해당 값은 아래와 같다.

[0061] **[예상 수행 시간 A] :** 새로운 컨테이너 요청이 해당 서버에 할당될 경우, 해당 컨테이너의 예상 수행 시간

[0062] **[예상 수행 시간 B] :** 새로운 컨테이너 요청이 해당 서버에 할당될 경우, 해당 서버에서 기존에 실행 중인 컨테이너들에 미치는 성능 간섭 정도 (추가 수행 시간)

[0063] - 위의 두 값을 이용하여, 컨테이너 워크로드에 대한 배치 전략을 구상할 수 있다. 본 방법에서 제안하는 전략

은 두 가지로, 이는 아래와 같이 구성된다.

[0065] **[전략 1]** : 연속된 컨테이너 요청에서 전체 처리 시간을 최소화 하는 전략

[0066] - 일련의 컨테이너 요청이 연속적으로 도착할 때, **[예상 수행 시간 A]** 와 **[예상 수행 시간 B]**의 합이 가장 최소가 되는 서버를 선택한다. 특정 서버에서 계산된 **[예상 수행 시간 A]** 의 값이 크더라도, **[예상 수행 시간 B]** 의 값이 작다면 해당 서버가 선택될 수 있다. 이는 전체적인 컨테이너 수행 시간을 최소화할 수 있지만, 현재 요청 받은 컨테이너의 수행 시간이 최소화되지는 않을 수 있다.

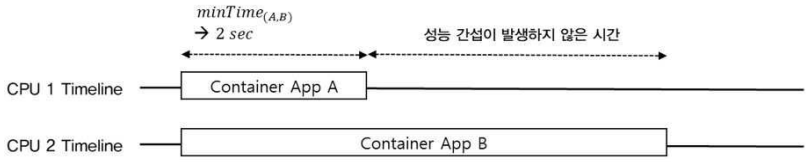
[0068] **[전략 2]** : 현재 요청 받은 컨테이너의 수행 시간을 최소화하는 전략

[0069] - 이전에 요청 받은 컨테이너 요청들로 인해 각 서버에 이미 컨테이너가 실행 중일 때, 현재 요청 받은 컨테이너에 대한 **[예상 수행 시간 A]**이 값이 최소가 되는 서버를 선택한다. 결과적으로, 현재 도착한 컨테이너 요청을 빠르게 수행할 수는 있지만, 다른 컨테이너에 가해지는 추가적인 성능 간섭을 고려하지 않기에, 전체적인 컨테이너 수행 시간의 합은 증가할 수 있다.

[0071] - 도 5는 위 두개의 전략을 적용한 예시를 나타낸다. 새로운 컨테이너의 요청이 발생했을 때, 각 서버에 대해 예상 수행 시간을 계산한다. 전체 컨테이너 처리 시간을 최소화하는 **[전략 1]**는, **[새롭게 요청된 컨테이너의 수행 시간 + 기존에 실행 중인 컨테이너의 추가 수행 시간]**의 합이 최소: 6.5초가 되는 **서버 1**을 선택한다. 현재 요청된 컨테이너의 수행 시간을 최소화하는 **[전략 2]**은, 두 개의 서버 중 4.0초가 소요되는 **서버 2**을 선택한다. 이 때, [전략 2]는 컨테이너 간 성능 간섭의 정도가 높아짐에 따라 전체적인 수행 시간이 증가할 수도 있지만, 현재 요청된 컨테이너의 수행 시간은 짧아질 수 있다.

도면

도면1



도면2

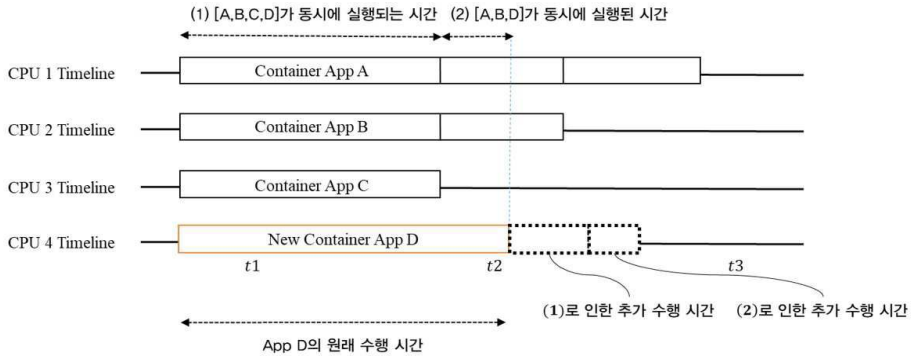
워크로드	A	B
소요 시간	2.0 sec	5.0 sec

컨테이너 애플리케이션의 기준 소요시간

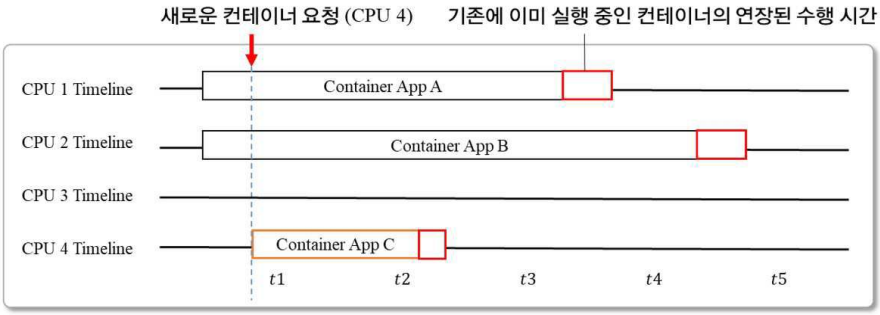
	수행 시간	추가 수행 시간	성능 감소치
A	2.2 sec	2.2 − 2.0 = 0.2	$\frac{0.2}{2.0} \approx 0.1$ (10%)
B	5.1 sec	5.1 − 5.0 = 0.1	$\frac{0.1}{5.0} \approx 0.05$ (5%)

A, B 를 동시에 실행 시, 성능 감소치 계산 방법

도면3



도면4



도면5

[전략 1] 전체 처리 시간 최적화 전략 → **서버 1를 선택**

[전략 2] 현재 요청된 컨테이너의 수행 시간 최소화 전략 → **서버 2을 선택**