(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2025-0089281
(43) 공개일자 2025년06월18일

(51) 국제특허분류(Int. Cl.)
G06F 9/455 (2018.01) *G06F 9/48* (2018.01)
G06N 3/04 (2023.01)
(52) CPC특허분류
G06F 9/45558 (2013.01)
G06F 9/4881 (2013.01)
(21) 출원번호 10-2023-0179020
(22) 출원일자 2023년12월11일
심사청구일자 2023년12월11일

(71) 출원인
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
김명현
경기도 수원시 영통구 청명남로4번길 21-6, 104호
이재학
경기도 성남시 분당구 돌마로 486번길 12, 216동 901호
유헌창
서울특별시 노원구 노원로 214, 6동 202호
(74) 대리인
김등용

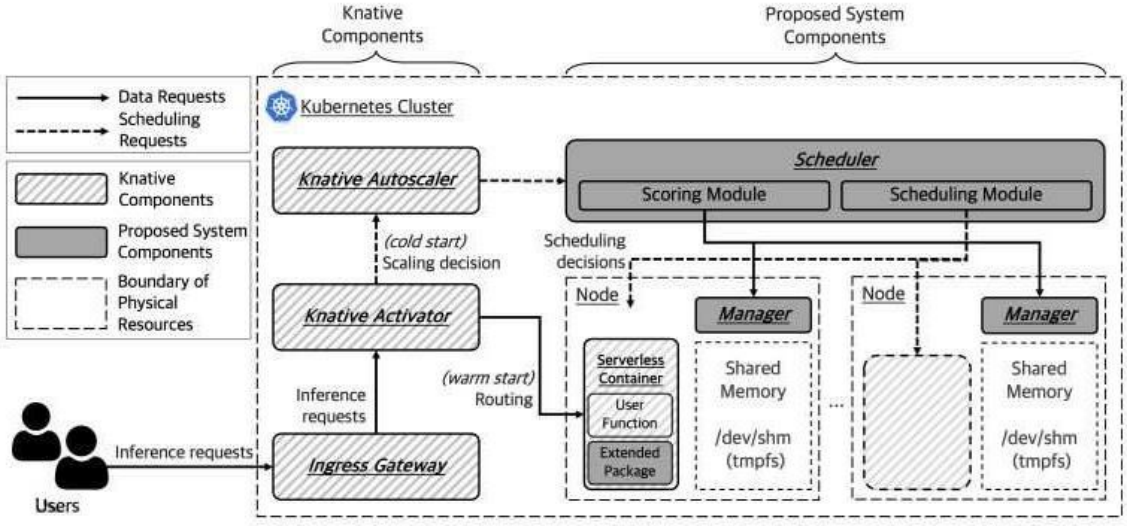
전체 청구항 수 : 총 7 항

(54) 발명의 명칭 **컨테이너 환경에서 심층 신경망 모델의 인메모리 공유 방법**

(57) 요 약

컨테이너 환경에서 심층 신경망 모델의 인메모리 공유 방법이 개시된다. 쿠버네티스 클러스터를 구성하는 쿠버네티스 장치와 복수의 워커 노드들을 포함하는 쿠버네티스 시스템에서 수행되는 심층 신경망(Deep Neural Network, DNN) 모델의 인메모리 공유 방법으로, 상기 쿠버네티스 장치가 사용자 단말로부터 추론 요청을 수신하는 단계, 상기 쿠버네티스 장치가 상기 복수의 워커 노드들 중에서 상기 추론 요청에 대응하는 컨테이너를 배치할 워커 노드를 선택하는 단계, 선택된 워커 노드가 상기 컨테이너를 배치하는 단계, 상기 선택된 워커 노드가 상기 선택된 워커 노드의 공유 메모리를 모니터링하는 단계, 및 상기 선택된 워커 노드가, 상기 공유 메모리의 사용량이 임계치를 초과하는 경우, 상기 공유 메모리에 저장된 공유 모델들 중 적어도 하나를 삭제하는 단계를 포함한다.

대 표 도



(52) CPC특허분류
G06N 3/04 (2023.01)
G06F 2009/45562 (2013.01)
G06F 2009/4557 (2019.08)
G06F 2009/45583 (2019.08)
G06F 2009/45591 (2019.08)
G06F 2201/81 (2013.01)

이 발명을 지원한 국가연구개발사업
과제고유번호 1711160486
과제번호 2022-0-00983-001
부처명 과학기술정보통신부
과제관리(전문)기관명 정보통신기획평가원
연구사업명 디지털전환K-SW기술개발(R&D)
연구과제명 사용자 별 오토모티브 헬스케어 서비스를 지원하는 엣지 클라우드 기반 차량 공유 플랫폼 개발
기 여 율 1/1
과제수행기관명 고려대학교 산학협력단
연구기간 2022.04.01 ~ 2023.12.31

명 세 서

청구범위

청구항 1

쿠버네티스 클러스터를 구성하는 쿠버네티스 장치와 복수의 워커 노드들을 포함하는 쿠버네티스 시스템에서 수행되는 심층 신경망(Deep Neural Network, DNN) 모델의 인메모리 공유 방법에 있어서,

상기 쿠버네티스 장치가 사용자 단말로부터 추론 요청을 수신하는 단계;

상기 쿠버네티스 장치가 상기 복수의 워커 노드들 중에서 상기 추론 요청에 대응하는 컨테이너를 배치할 워커 노드를 선택하는 단계;

선택된 워커 노드가 상기 컨테이너를 배치하는 단계;

상기 선택된 워커 노드가 상기 선택된 워커 노드의 공유 메모리를 모니터링하는 단계; 및

상기 선택된 워커 노드가, 상기 공유 메모리의 사용량이 임계치를 초과하는 경우, 상기 공유 메모리에 저장된 공유 모델들 중 적어도 하나를 삭제하는 단계를 포함하는, 심층 신경망 모델의 인메모리 공유 방법.

청구항 2

제1항에 있어서,

상기 컨테이너를 배치할 워커 노드를 선택하는 단계는,

상기 복수의 워커 노드들 각각의 스코어를 산출하는 단계; 및

가장 높은 스코어를 갖는 워커 노드를 상기 선택된 워커 노드로 선택하는 단계를 포함하는,

심층 신경망 모델의 인메모리 공유 방법.

청구항 3

제2항에 있어서,

상기 스코어를 산출하는 단계는, 공유 메모리에 상기 추론 요청에 대응하는 공유 모델이 저장되어 있는 워커 노드는 미리 정해진 스코어 구간 중 가장 높은 스코어를 갖는 것으로 결정하는,

심층 신경망 모델의 인메모리 공유 방법.

청구항 4

제3항에 있어서,

상기 스코어를 산출하는 단계는, 공유 메모리의 요청된 사이즈 보다 작은 여유 공간을 갖는 워커 노드는 상기 미리 정해진 스코어 구간 중 가장 낮은 스코어를 갖는 것으로 결정하는,

심층 신경망 모델의 인메모리 공유 방법.

청구항 5

제4항에 있어서,

상기 스코어를 산출하는 단계는, 수학식을 이용하여 스코어를 계산하고,

상기 수학식은 $\left(1-\frac{shm_{used}+req}{shm_{capacity}}\right)*100$ 이고,

상기 shm_{used}는 사용된 공유 메모리 용량이고, 상기 req는 요청된 사이즈이고, 상기 $shm_{capacity}$는 공유 메모리의 용량인,

심층 신경망 모델의 인메모리 공유 방법.

청구항 6

제5항에 있어서,

상기 선택된 워커 노드는 공유 메모리에 저장된 공유 모델의 사용에 관한 LRU(Least Recently Used) 기반의 큐를 유지하는,

심층 신경망 모델의 인메모리 공유 방법.

청구항 7

제5항에 있어서,

상기 삭제하는 단계는, 가장 오래전에 이용된 적어도 하나의 공유 모델을 삭제하는,

심층 신경망 모델의 인메모리 공유 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 컨테이너 환경에서 심층 신경망(Deep Neural Network, DNN) 모델의 인메모리 공유 방법에 관한 것이다.

배 경 기 술

[0002] 딥 러닝(Deep Learning) 기반 워크로드는 학습(Training)과 추론(Inference) 워크로드로 구분할 수 있다. 추론 워크로드는 단발적이고 수명이 길지 않은 특징이 있다. 이러한 특성은 최근 유행하는 서버리스 컴퓨팅의 장점과 잘 어울리는데, 서버리스는 사용자의 요청을 트리거로 작업을 시작하는 FaaS(Function-as-a-Service) 형태의 모델이다. 그러나 딥 러닝 모델을 서버리스로 구동하는 것은 다음 두 가지 문제를 야기한다.

[0003] 먼저, 서버리스는 사용자의 요청을 컨테이너로 처리하며 요청이 많아질 경우 컨테이너 개수를 늘리는 방법으로 확장하는데, 이 과정에서 동일한 모델 데이터가 여러 컨테이너에 중복되는 문제가 발생할 수 있다. 또한, 딥 러닝 서비스를 컨테이너 환경에서 구동할 시 딥 러닝 모델이 수행되는데 필요한 프로그램 구동 환경 및 프로그램 간 무거운 의존성을 가지고 있어 컨테이너 시작이 지연되는 콜드 스타트 문제(Cold Start Problem)가 있다.

[0004] 본 발명은 이러한 문제들을 완화하는 서버리스 플랫폼을 제안한다. 본 발명에서는 딥 러닝 모델을 컨테이너 간 인-메모리로 공유하여 보다 더 경량의 메모리 격리 수준을 가진 시스템을 구현할 수 있다. 또한, 자원 제약적인 환경에서의 효율적인 자원 활용을 위해 모델 배치 및 축출 알고리즘을 제안한다.

선행기술문헌

특허문헌

[0005] (특허문헌 0001) 대한민국 공개특허 제2020-0109819호 (2020.09.23. 공개)

(특허문헌 0002) 대한민국 공개특허 제2022-0075194호 (2022.06.07. 공개)

(특허문헌 0003) 대한민국 공개특허 제2022-0048927호 (2022.04.20. 공개)

(특허문헌 0004) 대한민국 공개특허 제2022-0149418호 (2022.11.08. 공개)

발명의 내용

해결하려는 과제

[0006] 본 발명이 이루고자 하는 기술적인 과제는 컨테이너 환경에서 DNN 모델의 인메모리 공유 방법 및 그 시스템을 제공하는 것이다.

과제의 해결 수단

[0007] 본 발명의 일 실시예에 따른, 컨테이너 환경에서 심층 신경망 모델의 인메모리 공유 방법은 쿠버네티스 클러스터를 구성하는 쿠버네티스 장치와 복수의 워커 노드들을 포함하는 쿠버네티스 시스템에서 수행되는 심층 신경망(Deep Neural Network, DNN) 모델의 인메모리 공유 방법으로, 상기 쿠버네티스 장치가 사용자 단말로부터 추론 요청을 수신하는 단계, 상기 쿠버네티스 장치가 상기 복수의 워커 노드들 중에서 상기 추론 요청에 대응하는 컨테이너를 배치할 워커 노드를 선택하는 단계, 선택된 워커 노드가 상기 컨테이너를 배치하는 단계, 상기 선택된 워커 노드가 상기 선택된 워커 노드의 공유 메모리를 모니터링하는 단계, 및 상기 선택된 워커 노드가, 상기 공유 메모리의 사용량이 임계치를 초과하는 경우, 상기 공유 메모리에 저장된 공유 모델들 중 적어도 하나를 삭제하는 단계를 포함한다.

발명의 효과

[0008] 본 발명의 실시예에 따른, 컨테이너 환경에서 DNN 모델의 인메모리 공유 방법 및 그 시스템에 의할 경우, 동일한 모델 데이터가 여러 컨테이너에 중복되는 문제로 인한 시스템 가용 메모리 부족 현상과 컨테이너 시작이 지연되는 콜드 스타트 문제를 해결할 수 있는 효과가 있다.

도면의 간단한 설명

[0009] 본 발명의 상세한 설명에서 인용되는 도면을 보다 충분히 이해하기 위하여 각 도면의 상세한 설명이 제공된다.

도 1과 도 2는 오리지널 컨테이너와 본 발명에서 제안하는 기법이 적용된 컨테이너 사이의 성능 비교로써, 메모리 풋프린트의 성능 비교(도 1)와 각 단계(step)의 실행을 위한 경과 시간의 성능 비교를 나타낸다.

도 3은 제안하는 시스템 아키텍처의 개요를 도시한다.

도 4는 각 노드의 공유 메모리 사용량에 대한 응답 예제를 나타낸다.

도 5는 스케줄러, 매니저, 및 함수 컨테이너 간의 스케줄링 흐름을 도시한다.

도 6은 본 발명의 일 실시예에 따른 DNN 모델의 인메모리 공유 방법을 설명하기 위한 흐름도이다.

발명을 실시하기 위한 구체적인 내용

[0010] 본 명세서에 개시되어 있는 본 발명의 개념에 따른 실시예들에 대해서 특정한 구조적 또는 기능적 설명들은 단지 본 발명의 개념에 따른 실시예들을 설명하기 위한 목적으로 예시된 것으로서, 본 발명의 개념에 따른 실시예들은 다양한 형태들로 실시될 수 있으며 본 명세서에 설명된 실시예들에 한정되지 않는다.

[0011] 본 발명의 개념에 따른 실시예들은 다양한 변경들을 가할 수 있고 여러 가지 형태들을 가질 수 있으므로 실시예들을 도면에 예시하고 본 명세서에서 상세하게 설명하고자 한다. 그러나, 이는 본 발명의 개념에 따른 실시예들을 특정한 개시 형태들에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물, 또는 대체물을 포함한다.

[0012] 제1 또는 제2 등의 용어는 다양한 구성 요소들을 설명하는데 사용될 수 있지만, 상기 구성 요소들은 상기 용어들에 의해 한정되어서는 안 된다. 상기 용어들은 하나의 구성 요소를 다른 구성 요소로부터 구별하는 목적으로만, 예컨대 본 발명의 개념에 따른 권리 범위로부터 벗어나지 않은 채, 제1 구성 요소는 제2 구성 요소로 명명될 수 있고 유사하게 제2 구성 요소는 제1 구성 요소로도 명명될 수 있다.

[0013] 어떤 구성 요소가 다른 구성 요소에 "연결되어" 있다거나 "접속되어" 있다고 언급된 때에는, 그 다른 구성 요소에 직접적으로 연결되어 있거나 또는 접속되어 있을 수도 있지만, 중간에 다른 구성 요소가 존재할 수도 있다고 이해되어야 할 것이다. 반면에, 어떤 구성 요소가 다른 구성 요소에 "직접 연결되어" 있다거나 "직접 접속되어" 있다고 언급된 때에는 중간에 다른 구성 요소가 존재하지 않는 것으로 이해되어야 할 것이다. 구성 요소들 간의 관계를 설명하는 다른 표현들, 즉 "~사이에"와 "바로 ~사이에" 또는 "~에 이웃하는"과 "~에 직접 이웃하는" 등도 마찬가지로 해석되어야 한다.

[0014] 본 명세서에서 사용한 용어는 단지 특정한 실시예를 설명하기 위해 사용된 것으로서, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 명세서에서, "포함하다" 또는 "가지다" 등의 용어는 본 명세서에 기재된 특징, 숫자, 단계, 동작, 구성 요소, 부분품 또는

이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0015] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 의미를 갖는 것으로 해석되어야 하며, 본 명세서에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.

[0016] 이하, 본 명세서에 첨부된 도면들을 참조하여 본 발명의 실시예들을 상세히 설명한다. 그러나, 특허출원의 범위가 이러한 실시예들에 의해 제한되거나 한정되는 것은 아니다. 각 도면에 제시된 동일한 참조 부호는 동일한 부재를 나타낸다.

[0017] 인공 지능(Artificial Intelligence, AI) 기술이 대중화되면서, 기계 학습 기반의 서비스가 대중화되었다. 많은 기계 학습 모델들은 파이토치(PyTorch)나 텐서플로우(Tensorflow)와 같은 심층 신경망(Deep Neural Network, DNN) 프레임워크에 의해 학습되고, 추천 시스템, 이미지 분석, 및 자연어 처리 등과 같은 서비스에 이용되고 있다. 이러한 DNN 워크로드는 학습 워크로드와 추론 워크로드로 분류된다. 학습 워크로드는 여러 수준의 행렬-벡터 곱으로 구성되어 있어 몇 시간에서 몇 주가 소요된다. 반면에, 추론 워크로드는 이미 학습된 모델을 이용하여 고정된 개수의 행렬 연산을 수행한다. 또한, 추론 워크로드에 따른 연산 시간은 실시간 서비스를 제공할 수 있을 만큼 충분히 짧다.

[0018] 최근에는 많은 서비스가 클라우드 환경에서 이용되고 있으며, DNN 서비스에 대한 클라우드 인프라의 사용 사례도 증가하고 있다. 학습 워크로드는 연속적이고 고도의 계산 집약적인 작업이기 때문에, 전체적으로 하나 이상의 GPU 인스턴스를 임대하는 것이 합리적이다. 반면에, 추론 프로세스는, 추론 요청이 발생할 때만 GPU 인스턴스를 사용하는 일회성 워크로드이며, 결과를 반환한 후 인스턴스는 아이들 상태(idle state)로 돌아간다. 따라서, 요청이 덜 자주 발생하는 상황에서, 가상 머신 인스턴스를 계속 활성화 상태로 유지하면 많은 유휴 리소스(idle resources)가 발생한다. 가용 시간(available time)을 최대화하기 위해, 최근 많은 연구가 서버리스 방식으로 추론 워크로드를 수행하고 있다.

[0019] FaaS(Function-as-a-Service)로도 알려진 서버리스 아키텍처는 이벤트 기반(event-driven) 및 PPU(Pay-Per-Use) 컴퓨팅 모델이다. 서버리스 모델로, 사용자의 요청(requests)과 같은 이벤트에 의해 트리거된 함수 코드(function code)를 인스턴스화한다(instantiates). 그러나, 요청이 없다면, 시스템은 사용 가능한 컴퓨팅 리소스를 늘리기 위해 활성 인스턴스를 해제한다. 클라우드 서비스 제공자는, 또한, 사용자가 인스턴스를 사용하는 시간에 대해서만 요금을 부과하는 과금 체계를 사용한다. 대부분의 상용 클라우드 플랫폼에서 서버리스 서비스는 컨테이너 기술(container technology)로 구현되기 때문에, 요청이 증가함에 따라 거의 무한한 확장성(scalability)을 갖는다.

[0020] 이러한 장점에도 불구하고, 서버리스 추론은 여전히 DNN 모델과 같이 방대한 데이터를 처리하는 데 어려움이 있다. 서버리스에서는, 동일한 요청에 대한 컨테이너가 동일한 샌드박스에서 제공될 것이라는 보장이 없다. 따라서, 서버리스 제공자는, 개발자에게 자신들의 로직을 불안정(stateless)하고 일시적으로(ephemeral)으로 구현하도록 강제하며, 이러한 특성은 두 가지 주요 도전 과제를 초래한다. 첫째, 데이터 중복 문제(data duplication problem)이다. 이 문제는, 서버리스 플랫폼이 함수를 컨테이너로 래핑(wrap)하기 때문에 발생한다. 플랫폼이 동일한 DNN 모델에 대한 여러 요청을 처리할 때, 모델 데이터는 컨테이너들 사이에서 복제될 수 있으며, 이는 각 컨테이너에게 할당된 시스템 메모리 영역에 무분별하게 복제 데이터를 적재시킴으로써 시스템 가용 메모리 확보가 힘들다. 둘째, 콜드 스타트 문제(cold start problem)이다. 딥러닝 서비스는 복잡한 의존 트리(dependency tree)를 갖기 때문에, 컨테이너를 시작할 때 일정한 시간 지연이 발생할 수 있다.

[0021] 본 발명에서는, 오픈소스 서버리스 플랫폼인 케이네이티브(Knative)를 기반으로 한 확장된 서버리스 디자인을 제안한다. 제안된 시스템은 다음과 같은 요소를 포함한다.

[0022] 1) 서버리스 컨테이너 간에 DNN 모델을 공유하는 기술.

[0023] 2) 공유 모델에 대한 친밀도(affinity)를 기반으로 한 커스텀 스케줄러(Cumstom scheduler).

[0024] 3) 공유 메모리 영역을 효율적으로 사용하기 위한 공유 모델 제거 알고리즘(Shared-model eviction

algorithm).

[0025] 시스템을 소개하기 전에, 제안하는 기법이 적용된 컨테이너와 오리지널 컨테이너를 비교함으로써 공유 전략의 이점을 보인다. 각 컨테이너 환경을 단계별(step by step)로 비교하기 위해, 다섯 개의 연속적인 측정 지점이 설정되고 이는 표 1에 요약되어 있다. 각 단계(step)는 메모리 풋프린트(memory footprint)와 경과 시간(elapsed time)을 측정한다.

[0026] [표 1]

	Target of Measurement
Point 1	Container creation & Loading dependencies
Point 2	Loading DNN model
Point 3	Loading model weights
Point 4	Removing temporal data
Point 5	Inference

[0027]

[0028] 도 1과 도 2는 오리지널 컨테이너와 본 발명에서 제안하는 기법이 적용된 컨테이너 사이의 성능 비교로써, 메모리 풋프린트의 성능 비교(도 1)와 각 단계(step)의 실행을 위한 경과 시간의 성능 비교를 나타낸다. 후자의 컨테이너는 이미지 처리 모델(image-processing model)인 사전학습된 VGG16 모델의 가중치를 공유하는 환경이다. 도 1은, 공유 전략을 사용하는 컨테이너는, 피크 타임(peak time, Point 3)에서, 오리지널 컨테이너에 비교하여 메모리의 약 49%를 점유함을 보인다. 또한, 도 2는 각 측정 포인트에 대한 경과 시간을 나타낸다. 공유 전략을 사용하는 환경은 파일 I/O 동작(file I/O operations)이 없기 때문에, 모델 로드 시간에서 오리지널 환경보다 약 0.2ms 덜 소요된다.

[0029] 결과적으로, 모델 공유 전략(model-sharing strategy)은 메모리 활용(memory utilization)에 이점을 제공하며, 지연 오버헤드(latency overhead)를 초래하지 않는다. 따라서, 본 발명에서는 모델 공유 시스템을 제안한다.

[0030] 이하에서는, 제안하는 시스템의 기초인 쿠버네티스(Kubernates)와 케인이티브(Knative)를 설명한다.

[0031] **서버리스 컴퓨팅(Serverless Computing)**

[0032] 서버리스는 클라우드 상의 개발자가 서버를 관리하지 않아도 되도록 인프라를 추상화하고 제공하는 개발 모델이다. IssA(Infrastructure-as-a-service)나 온프레미스 환경(on-premise environments)을 사용하는 개발자들은 계속해서 서버를 모니터링하고 관리하여야 한다. 반면에, 서버리스 컴퓨팅은 자동 스케일링(automatic scaling)과 이벤트 기반 설계(event-driven design)를 특징으로 하기 때문에, 서버 인프라를 관리할 필요가 없다. 개발자는 단순하게 함수 코드(function codes)를 시스템에 업로드하고, 클라우드 제공자는 함수 코드를 컨테이너에 패키징한다. 또한, 서버리스는 실행 횟수와 각 함수 컨테이너의 실행 시간에 대한 비용만을 청구하는 사용량 기반의 과금 모델(pay-for-use billing model)을 채택하기 때문에, 전통적인 클라우드 컴퓨팅 서비스보다 비용적인 이점을 가지고 있다. 서버리스의 인기 이유를 요약하면, 향상된 하드웨어 사용, 운영 비용 절감, 및 더 빠른 스케일링에 있다.

[0033] 서버리스 컴퓨팅을 지원하는 상용 클라우드 서비스는 Google Cloud Functions, Microsoft Azure Functions, AWS Lambda를 포함하며, Apache Open Whisk, OpenFaaS, 및 Knative와 같은 오픈 소스 프로젝트도 존재한다. 대부분의 오픈 소스 서버리스 플랫폼은 요청이 발생할 때 함수 코드를 컨테이너화하고 쿠버네티스(Kubernetes)에서 실행함으로써 동작한다. 또한, 부하(load)가 발생할 때 더 많은 컨테이너를 생성할 수 있어 높은 확장성(scalability)을 보인다.

[0034] **쿠버네티스(Kubernetes)**

[0035] 쿠버네티스(Kubernetes)는 다중 노드 클러스터 환경(multi-node cluster environment)에서 컨테이너 워크로드의 자동 배포(automated deployment), 스케일링, 및 관리를 제공하는 사실상의 표준 컨테이너 오케스트레이션 시스템이다. 상술한 바와 같이, 대부분의 서버리스 플랫폼은 함수 코드를 패키징하기 위하여 컨테이너 기술을 이용한다. 따라서, 인바운드 요청이 급증할 때, 함수 컨테이너를 확장하는 것이 필요하다. 또한, 각 서버리스 컨테이너는 서로의 프로세스에 영향을 미치지 않아야 한다. 쿠버네티스(Kubernetes)는 이러한 모든 요구 사항을 충족하며 확장성을 지원하기 때문에, 다양한 서버리스 플랫폼의 기본 시스템으로 사용된다. 또한, 쿠버네티스(Kubernetes) 시스템은 제어 평면 노드(control plane node)를 갖는 다중 노드 클러스터(multi-node cluster)

와 다중 워크 노드들(multiple worker nodes)로 구성된다.

[0036] 쿠버네티스(Kubernetes) 관련 중요 구성 요소들로 파드(Pod), 배포(Deployment), 및 스케줄러(Scheduler)가 있다. 파드(Pod)는 하나 이상의 컨테이너화된 어플리케이션을 실행하는 논리적 호스트를 나타낸다. 배포(Deployment)는 원하는 상태로 파드(Pod) 인스턴스를 생성하고 유지하는 방법을 처리하는 쿠버네티스(Kubernetes) 자원이다. 마지막으로, 스케줄러(scheduler)는 새로운 파드(Pods)를 감시하고 파드(Pod)를 배포할 노드를 결정하는 역할을 담당한다.

[0037] 쿠버네티스(Kubernetes) 시스템은 구성 파일을 작성하여 시스템 관리자가 스케줄러를 커스텀화할 수 있도록 한다. 스케줄링 프로세스는 여러 단계에서 발생하며, 스케줄러는 각 단계를 확장 지점으로 노출한다. 확장 지점 집합에는 노드를 스코어링하기 위해 사용되는 preScore 및 score 확장 지점을 포함한다.

[0038] **케이네이티브(Knative)**

[0039] 케이네이티브(Knative)는, 쿠버네티스(Kubernetes) 클러스터 상에서 실행되는 가장 인기 있는 오픈소스 서버리스 플랫폼 중 하나이다. 케이네이티브(Knative)는 제로-오토스케일링(zero-autoscaling), 점진적 롤아웃(progressive rollouts), 간단한 추상화(simpler abstractions), 및 서비스 메시(service mesh)를 제공하기 때문에, 다른 플랫폼과 비교하여 더 널리 사용된다. 케이네이티브(Knative) 플랫폼에서, 서버리스 함수는 Service, Revision, 및 Route의 세 가지 유형으로 정의된다. Service 리소스는 함수의 전체 라이프사이클을 래핑(warps)하며, Route 및 Revisions을 관리한다. Revision은 구성 파일을 갖는 함수 코드의 스냅샷(snapshot)을 나타내며, 직접적으로 Pods(파드)를 관리한다. Route는 네트워크 엔트포인트(network endpoint)를 하나 이상의 Revisions에 매핑하며, 개발자가 다양한 방식으로 네트워크 트래픽을 제어할 수 있도록 한다.

[0040] 제로에서 무한대로 스케일링할 수 있는 능력은 서버리스 아키텍처의 가장 중요한 장점 중 하나이며, 케이네이티브(Knative)도 케이테이티브 오토스케일러(Knative Autoscaler)를 통해 편리하게 스케일을 조절할 수 있는 기능을 제공한다. 이는 Revisions을 배포하거나 종료함으로써 스케일을 조절한다. 특정 서비스에 대한 새로운 요청이 없는 경우, 오토스케일러는 서비스의 Revision 복제본(replicas) 수를 0으로 설정한다. 그러나, 시스템은 그러한 서비스를 완전히 제거하지는 않는다. 케이네이티브 액티베이터(Knative Activator)는, 다시 요청이 시작될 때, 이러한 제로 스케일 상태에서 서비스를 웜업하는 역할을 수행한다.

[0041] 본 발명에서, 케이테이티브(Knative)는 제안된 시스템의 기본 시스템으로 사용된다. 케이네이티브(Knative) 컨테이너는, 5 줄 미만의 코드를 변경함으로써, 공유 메모리를 통해 IPC(Inter-Process Communication, 프로세스 간 통신)를 수행할 수 있기 때문이다. 이는 최소한의 코드 변경으로 기존 시스템과의 호환성을 유지하기 위한 요구 사항을 충족한다.

[0042] **POSIX 공유 메모리**

[0043] 현대의 운영 체제(OS)는 프로세스 사이의 데이터를 공유하기 위한 공유 메모리 영역을 제공한다. 이 공유 메모리는 IPC(프로세스 간 통신)이나 공유 라이브러리에 사용된다. 공유 메모리의 APIs는 System V 및 POSIX 스타일로 분류된다. 리눅스(Linux)에서의 POSIX 공유 메모리는 tmpfs로 공유 메모리를 노출하며, 이는 가상 메모리 파일시스템으로서 인메모리 파일을 제공한다. 리눅스(Linux)에서, 공유 메모리 객체는 /dev/shm 하에 마운트된 tmpfs에서 생성된다. tmpfs에서 객체는 파일시스템에 위치한 파일로 취급된다. 프로세스가 tmpfs의 객체를 이용할 때, 객체는 프로세스의 주소 공간으로 매핑되어 사용된다.

[0044] 일부 중요한 POSIX 공유 메모리 APIs는 shm_open, ftruncate, 및 mmap이다. 기존 공유 메모리 객체에 액세스하거나 새로 생성할 때, shm_open이 이용된다. 이 함수는 /dev/shm tmpfs 하의 객체의 파일 디스크립터(file descriptor)를 반환한다. 객체는 파일로 취급되므로, ftruncate 함수로 크기가 조정될 수 있다. 마지막으로, 프로세서는 객체를 자신의 주소 공간에 매핑할 수 있다.

[0045] 이하에서는, 제안된 시스템의 전반적인 구조를 탐구하며, 공유 메모리 사용 구현, 추론 요청을 처리하는 사용자 스케줄러, 및 공유 모델 평가 알고리즘을 소개한다. 제안하는 시스템은 공유 메모리 특징과 통합되어 있으며, 쿠버네티스(Kubernetes) 클러스터 상의 케이테이티브(Knative)와 협력하도록 설계된다. 도 3은 제안하는 시스템 아키텍처의 개요를 도시한다. 시스템은 서버리스 컨테이너를 통해 파이토치(PyTorch) 기반의 DNN 모델을 공유하고, 공유 메모리 영역에 최적 개수의 모델을 유지한다. 이를 위해, 시스템은 다음과 같은 구성 요소를 구현한다.

[0046] 첫째는 공유 모델 매니저(shared-model manager, 도 3의 Manager)이다. 공유 모델은 DNN 모델의 추상화로, 가

중치가 각 호스트(워커 노드를 의미할 수 있음)의 공유 메모리 영역에 로딩된다. 공유 모델을 이용하는 서버리스 컨테이너는 모델 가중치를 제로-카피(zero-copy)로 사용할 수 있다. 공유 모델 매니저는 각 워커 노드에 배포되며(또는 구비되며) 리눅스 데몬(Linux daemons)으로 취급된다. 실시예에 따라, 공유 모델 매니저는 워커 노드의 제어부, 매니저 유닛, 또는 매니저부로 명명될 수도 있다.

[0047] 시스템의 두번째 구성 요소로써, 쿠버네티스(Kubernetes) 커스텀 스케줄러(custom scheduler, 도 3의 Scheduler)를 제안한다. 커스텀 스케줄러는 파드(pods)의 배치를 담당한다. 모든 서버리스 컨테이너는 파드(pod)로 래핑된다. 커스텀 스케줄러는 공유 메모리 배포 상태에 따른 스코어링 프로세스를 포함한다.

[0048] 마지막 구성 요소는 파이토치(PyTorch)를 기반으로 한 확장된 파이썬 패키지(extended python backage, 도 3의 Extended Package)이다. 가장 인기 있는 오픈소스 심층 신경망 프레임워크의 하나로 파이토치(PyTorch) 패키지는 연구 및 실제 제품에서 널리 사용된다. 본 발명에서는, 모델 저장 및 로드와 관련된 오리지널 PyTorch APIs를 래핑하는 몇 가지 인터페이스가 구현되었다.

[0049] 도 3은 제안하는 시스템 아키텍처의 고수준 개요을 도시한다. 인그레스 게이트웨이(Ingress Gateway)는 사용자로부터 외부 트래픽을 집계하고(aggregates) 요청을 케이네이티브 액티베이터(Knative Activator)로 전송한다. 타겟 서비스가 준비되면, 액티베이터(Activator)는 요청을 파드(pod) 내의 서버리스 컨테이너로 라우팅한다. 그렇지 않은 경우, 서비스가 0으로 축소되었다면, 케이네이티브 오토스케일러(Knative Autoscaler)는 서비스 스케일업(scaling-up)을 결정하고, 이는 콜드 스타트 딜레이를 야기한다. 그러면, 새로운 파드(pod)는 커스텀 스케줄러에 의해 배포된다. 제안하는 시스템의 스케줄러는 각 노드에서 공유 메모리 사용과 관련된 메트릭(metrics)을 수집한다. 스케줄러 내의 스코어링 모듈(scoring module)은 메트릭을 기반으로 각 노드의 가용성(availability)을 평가한다. 마지막으로, 스케줄링 모듈(scheduling module)은 파드(pod) 배치를 결정한다.

[0050] **파이토치(PyTorch)-기반 확장된 패키징**

[0051] 확장된 패키지는 사용자 코드와 제안된 시스템 간의 통신을 담당한다. 사용자가 시스템에 배포하려고 하는 서버리스 컨테이너는 확장된 패키지를 기반으로 설치되어야 한다. 이 패키지는 오리지널 파이토치(PyTorch) 프레임워크와 추가적인 인터페이스를 포함한다.

[0052] 표 2는 확장된 패키지의 네 가지 래퍼 인터페이스(wrapper interface)를 기술한다. 사용자는 PyTorch의 오리지널 함수 대신 이 래퍼 함수(wrapper functions)을 간단히 사용할 수 있다. create_shm 함수는 각 노드에서 공유 메모리 영역을 예약하고, 새 영역이 예약되었음을 매니저에게 알린다. save_states 함수는 타겟 모델의 가중치 텐서(weight tensors)를 평탄화하고 저장하며, 공유 모델의 메타데이터를 매니저에게 전송한다. 모델이 공유 메모리 영역에 업로드되면, 사용자는 모델의 메타데이터를 요청하기 위해 get_metadata 함수를 호출한다. 그러면, load_states 함수는 제로-카피(zero-copy)로 모델 가중치를 리드(reads)하고 이를 모델 아키텍처에 연결한다.

[0053] [표 2]

Function	Descrption
create_shm	Reserve the shared memory region.
save_states	Flatten and write the weight tensors to the shared memory region.
get_metadata	Request the shared-model metadata to the manager.
load_states	Connect the model weights from the shared memory region by zero-copy.

[0054]

[0055] **공유 모델 매니저(Shared-Model Manager)**

[0056] 공유 모델 매니저는 메모리 예약과 모델 삭제(model eviction)를 포함하는 공유 모델 관리를 담당하는 노드당 데몬이다. 또한, 매니저는 공유 모델에 대한 액세스 로그(access logs) 및 공유 메모리 사용과 관련된 메트릭(metrics)을 제공한다. 이러한 정보를 스케줄러에 노출시키기 위해, 매니저는 HTTP API 서버로 구현될 수 있다.

[0057] 매니저는 메타데이터, 모델 정보, 및 액세스 로그를 SQLite, 경량 SQL 데이터베이스 엔진(lightweight SQL database engine)으로 유지할 수 있다. 공유 모델은 공유 메모리 영역에 저장되고 휘발성(volatility)을 갖기 때문에, 모델을 위한 데이터베이스도 지속적(persistent)일 필요는 없다. 따라서, SQLite 엔진은 매니저의 파드(pod) 내에 내장될 수 있다.

[0058] 또한, 매니저는 메모리를 예약하는 역할을 수행한다. 시스템은 POSIX 공유 메모리 API를 채용할 수 있다. 새로운 공유 모델의 배포에 관한 요청이 발생하면, 매니저는, shm_open 함수를 이용하여, /dev/shm 하에 객체를 생

성한다. 파이토치(PyTorch)-기반 패키지는 가중치 텐서를 선형 플로트 어레이(linear float array)로 펼치고, 파일 I/O를 이용하여 공유 메모리 객체에 기록한다.

[0059] 그러나, tmpfs의 용량은 시스템의 물리적인 메모리의 절반으로 제한되어 있으며, 이는 공유 모델에 대한 할당량 제한을 의미한다. 이 문제를 해결하기 위해, 매니저는 공유 모델 삭제 알고리즘을 구현한다.

[0060] 특정 모델에 대한 액세스 요청이 발생할 때마다, 제안하는 시스템 내의 스케줄러는 매니저에게 요청을 알린다. 요청에 관한 정보는 로그로 변환되어 매니저의 데이터베이스 내에 저장된다.

[0061] 마지막으로, 공유 모델 매니저는 tmpfs의 사용과 연관된 메트릭을 제공한다. 이 메트릭은 /dev/shm의 용량 및 사용량과 각 공유 모델의 사용량을 포함한다. 도 4는 각 노드의 공유 메모리 사용량에 대한 응답 예제를 나타낸다. 공유 모델 매니저는 공유 메모리 영역의 상태를 지속적으로 모니터링한다. 도 4에서, 매니저는 model_name 및/또는 tag_name과 같은 모델 정보를 유지한다.

[0062] **컨테이너 배치(Container Placement)**

[0063] 케이테이티브(Knative)의 오리지널 디자인에서, 개발자는 사용자 함수 코드를 포함하는 서비스를 생성한다. 그런 다음, 케이네이티브 컨트롤러(Knative Controller)는 이를 쿠버네티스(Kubernetes) 배포로 변환하며, 처음에는 0으로 스케일링된다. 첫번째 추론 요청이 인그레스 게이트웨이(Ingress Gateway)에 도착하면, 액티베이터(Activator)는 오토스케일러(Autoscaler)에 새로운 서비스를 스케일 아웃하도록 요청한다. 스케일링 결정은 커스텀 스케줄러의 스케줄링 프로세스를 트리거하며, 세부 내용은 도 5에 표시되어 있다.

[0064] 도 5는 스케줄러(커스텀 스케줄러), 매니저(Manager), 및 함수 컨테이너 간의 스케줄링 흐름을 도시한다. 스케줄러는 우선 각 노드로부터 수집된 메트릭에 기초하여 스코어링 프로세스를 시작한다. 메트릭에는, 도 4에 설명된 모든 및 각 공유 모델의 예약된 공유 메모리 양이 포함된다. 스케줄러, 특히 스코어링 모듈은 알고리즘 1(Algorithm 1)을 사용하여 각 노드의 스코어를 계산한다.

[0065] [알고리즘 1]

Algorithm 1 Scoring algorithm for each node

Input: target model M, shm request $M.req$
nodes list N $(1, ..., i)$
$N_i.shm_{used}$, $N_i.shm_{capacity}$

initialize scores list $S := \{\}$
for N_i in N **do**
  if M has been deployed in N_i **then**
    $S_i \leftarrow 100$
  else if $M.req > (N_i.shm_{capacity} - N_i.shm_{used})$ **then**
    $S_i \leftarrow 0$
  else
    $S_i \leftarrow 100 * (1 - (N_i.shm_{used} + M.req)/N_i.shm_{capacity})$
  end if
end for
for S_i in S **do**
  $S_i \leftarrow 100 * S_i/max(S)$
end for

[0066]

[0067] 알고리즘 1에서, M은 타겟 모델을, M.req는 공유 메모리의 요청된 사이즈를 나타낸다. N_i는 특정 워커 노드를, $N_i \cdot shm_{used}$와 $N_i \cdot shm_{capacity}$는 각각 공유 메모리 사용량과 용량을 나타낸다.

[0068] 스코어링 모듈은 먼저 대상 모델이 노드에 배치되었는지 여부를 확인한다. 모델이 이미 노드에 배치되었다면, 해당 노드의 스코어 값은 100(가장 높은 스코어)이 될 수 있다. 모델이 노드에 비치되어 있지 않지만, 공유 메모리의 요청된 사이즈가 잔여 용량 보다 크다면 스코어 값은 0(가장 낮은 스코어)이 될 수 있다. 스코어의 값은 a에서 b까지의 범위(예컨대, 0에서 100까지의 범위)를 가질 수 있다. 노드가 새로운 모델을 배치할 충분한 공유 메모리를 가진다면, 스코어는 수학식 1을 통해 계산될 수 있다. 각 노드의 스코어 계산이 완료되면, 정규화 프로세스가 수행되어 스코어의 범위를 0에서 100까지 설정한다.

[0069] [수학식 1]

[0070] $$\left(1-\frac{shm_{used}+req}{shm_{capacity}}\right)*100$$

[0071] 다시 도 5를 참조하면, 알고리즘 1을 통한 스코어링 프로세스 이후에, 스케줄러는 새로운 함수 컨테이너를 배치할 노드를 결정한다. 이때, 가장 높은 스코어 값을 가진 노드가 새로운 함수 컨테이너를 배치할 노드로 결정될 수 있다. 도 5는 또한, 컨테이너 초기화 후에 공유 모델 데이터를 요청하기 위한 일련의 액션을 도시한다. 컨테이너는 공유 모델 매니저로부터, 공유 메모리 블록의 이름과 각 가중치 텐서의 크기를 포함하는 메타데이터를 획득하려고 시도한다. 클러스터 내에 공유 모델이 존재한다면, 컨테이너는 확장된 패키지의 load_states 함수를 호출하여 제로-카피로 공유된 가중치를 연결한다.

[0072] 그렇지 않으면, 타겟 모델은 공유 메모리 영역에 존재하지 않으며, 이는 추론 요청이 새로운 모델을 필요로 한다는 것을 의미한다. 마지막으로, 컨테이너는 모델 가중치를 네트워크를 통해 가져와서 새로운 공유 메모리 블록에 기록한다. 이 경우에는, 사용자가 네트워크 지연을 겪게 되지만, 이후에 동일한 모델을 사용하는 요청은 동일한 문제를 겪지 않아도 된다.

[0073] **공유 모델의 삭제(Eviction of Shared-Model)**

[0074] 엣지 클라우드와 같이 리소스 제약이 있는 환경에서는, 모든 DNN 모델을 각 노드의 공유 메모리 영역의 할당 정도가 중앙 클라우드보다 제한적이다. 이러한 관점에서, 제안하는 시스템은 공유 메모리가 과도하게 할당되었는지 여부를 감지하고, 제한적인 공유 메모리 크기에 따른 가용 메모리 확보 및 메모리 효율성 증대를 위해 빈번하게 사용되지 않는 공유 모델을 제거하는 대체 알고리즘이 필요하다.

[0075] 공유 모델 제거 알고리즘의 핵심 아이디어는 각 노드에 대한 LRU(Least Recently Used) 기반의 큐를 유지하는 것이다. 시스템은 주기적으로 큐의 꼬리에 있는 요소의 고정된 크기를 확인하여 기존 객체를 제거할 수 있다.

[0076] 본 발명에서는, 공유 모델 매니저가 공유 모델의 액세스 로그를 저장하는 LRU 큐를 유지하고, 공유 메모리 사용량이 임계값 Θ를 초과할 때 큐의 꼬리에서 고정된 개수의 로그를 확인한다. 공유 메모리 영역의 사용량이 Θ를 초과하면, 매니저는 최근에 가장 사용되지 않은 모델을 희생자로 간주하여 공유 메모리 영역에서 제거할 수 있다. 그러나, 희생 모델과 관련된 일부 진행 중인 컨테이너(in-progress containers)가 있을 수 있다. 여전히 일부 컨테이너에서 사용되는 모델을 제거하면, 손상된 가중치 값과 같은 얘기치 못한 실패가 발생할 수 있다. 이러한 문제를 피하기 위해, 매니저는, 모든 컨테이너가 종료될 때까지, 희생자를 후보(candidate)로 표시한 후, 모든 컨테이너가 종료되면 후보를 삭제할 수 있다. 알고리즘 2는 이러한 전체 과정을 의사코드(pseudo-code)로 기술한다.

[0077] [알고리즘 2]

Algorithm 2 Shared-model eviction algorithm

Input: shm_{used}, $shm_{capacity}$
shm threshold θ
LRU queue Q
reference counters list R
initialize *candidate* models list C := {}
while $shm_{used}/shm_{capacity} \geq \theta$ **do**
  move $Q.tail$ to C
  for C_i in C **do**
    if $R[C_i] == 0$ **then**
      release shared memory region of C_i
      delete C_i
    end if
  end for
end while

[0078]

[0079] 도 6은 본 발명의 일 실시예에 따른 DNN 모델의 인메모리 공유 방법을 설명하기 위한 흐름도이다.

[0080] DNN 모델의 인메모리 공유 방법은 적어포 프로세서(Processor) 및/또는 메모리(Memory)를 포함하는 컴퓨팅 장치에 의해 수행될 수 있다. 다시 말하면, DNN 모델의 인메모리 공유 방법을 이루는 단계들 중 적어도 일부는 컴퓨

팅 장치에 포함되는 프로세서의 동작으로 이해될 수 있다. 또한, 컴퓨팅 장치는 PC(Personal Computer), 서버(server), 래탑 컴퓨터, 태블릿 PC 등을 포함한다. 실시예에 따라, 컴퓨팅 장치는 물리적으로 구별되는 복수의 장치들로 구현될 수 있다. 구체적인 예로, 인메모리 공유 방법은 컨테이너 환경을 제공하는 쿠버네티스 시스템에서 수행될 수 있다. 쿠버네티스 시스템은 쿠버네티스 클러스터를 구성하는 쿠버네티스 장치와 복수의 워커 노드들을 포함할 수 있다. 이하에서, DNN 모델의 인메모리 공유 방법을 설명함에 있어, 앞선 기재와 중복되는 내용은 그 구체적인 기재를 생략하기로 한다.

[0081] 우선, 사용자 단말로부터 소정의 추론 모델(DNN 모델을 의미할 수 있음)을 대상으로 하는 추론 요청이 수신된다(S110). 추론 요청은 쿠버네티스 장치에 의해 수신될 수 있다. 추론 요청은 추론 모델의 입력 및/또는 추론 모델에 대한 정보(추론 모델을 식별하기 위한 정보 또는 추론 모델을 선택하기 위한 추론 동작에 대한 정보)를 포함할 수 있다.

[0082] 추론 요청을 위한 컨테이너를 배치할 워커 노드가 선택된다(S120). 예컨대, 쿠버네티스 장치는 복수의 워커 노드들 중에서 수신된 추론 요청을 처리하기 위한 컨테이너를 설치할 워커 노드를 선택할 수 있다. 이를 위해, 쿠버네티스 장치는 각 워커 노드의 공유 메모리에 공유된(또는 저장된) 공유 모델에 대한 메트릭(metric, 척도)에 대한 요청을 각 워커 노드로 송신하고, 이에 대한 응답으로 각 워커 노드로부터 공유 모델에 대한 메트릭을 수신할 수 있다. 수신된 메트릭은 공유 모델에 대한 정보(모델을 식별하기 위한 정보, 공유 메모리의 용량, 공유 메모리의 사용량) 등을 포함할 수 있다.

[0083] 또한, 쿠버네티스 장치는 선택의 기준이 되는 각 워커 노드의 스코어를 산출할 수 있다. 스코어는 알고리즘 1을 이용하여 산출될 수 있다. 이때, 가장 높은 스코어를 갖는 워커 노드가 컨테이너를 배치할 워커 노드로 결정될 수 있다.

[0084] 선택된 워커 노드에 컨테이너가 배치된다(S130).

[0085] 선택된 워커 노드에 공유 모델이 이미 존재하는 경우, 워커 노드는 공유 모델의 메타데이터를 컨테이너에 로드함으로써, 추론 동작을 수행할 준비를 완료한다. 이에 따라, 추론 요청에 대한 추론 동작이 수행된다. 이를 위해, 추론을 위한 입력 데이터는 쿠버네티스 장치로부터 워커 노드로 전송되고, 추론의 결과는 다시 쿠버네티스 장치로 전송된다. 쿠버네티스 장치는 추론 결과를 사용자 단말로 전송할 수 있다.

[0086] 선택된 워커 노드에 공유 모델이 존재하지 않는 경우, 워커 노드는 공유 모델을을 공유 메모리에 저장함으로써, 추론 동작을 수행할 준비를 완료한다. 이에 따라, 추론 요청에 대한 추론 동작이 수행된다. 이를 위해, 추론을 위한 입력 데이터는 쿠버네티스 장치로부터 워커 노드로 전송되고, 추론의 결과는 다시 쿠버네티스 장치로 전송될 수 있다. 쿠버네티스 장치는 추론 결과를 사용자 단말로 전송할 수 있다.

[0087] 워커 노드에 의한 공유 메모리 모니터링 동작이 수행된다(S140). 즉, 워커 노드는 공유 메모리의 사용량 등을 주기적으로 또는 비주기적으로 모니터링할 수 있다. 또한, 워커 노드는 공유 메모리에 저장된 공유 모델의 사용에 대한 LRU 기반의 큐를 유지할 수 있다.

[0088] 모니터링 결과에 따라, 공유 메모리에 저장된 적어도 하나의 공유 모델이 삭제될 수 있다(S150). 워커 노드는 공유 메모리의 사용량이 임계치를 초과하는 경우, 가장 마지막에 이용된 공유 모델을 삭제함으로써, 메모리 효율성을 제고할 수 있다.

[0089] 이상에서 설명된 장치는 하드웨어 구성 요소, 소프트웨어 구성 요소, 및/또는 하드웨어 구성 요소 및 소프트웨어 구성 요소의 집합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성 요소는, 예를 들어, 프로세서, 콘트롤러, ALU(Arithmetic Logic Unit), 디지털 신호 프로세서(Digital Signal Processor), 마이크로컴퓨터, FPA(Field Programmable array), PLU(Programmable Logic Unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(Operation System, OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 애플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술 분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(Processing Element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 콘트롤러를 포함할 수 있다. 또한, 병렬 프로세서(Parallel Processor)와 같은, 다른 처리 구성(Processing Configuration)도 가능하다.

[0090] 소프트웨어는 컴퓨터 프로그램(Computer Program), 코드(Code), 명령(Instruction), 또는 이들 중 하나 이상의

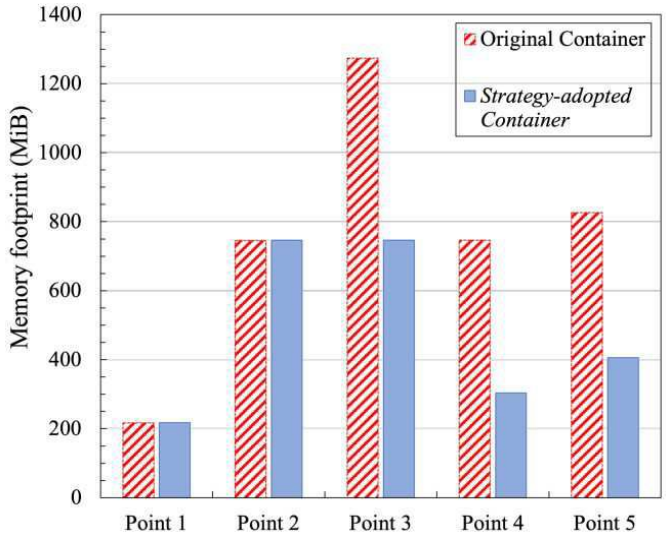
조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로(Collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성 요소(Component), 물리적 장치, 가상 장치(Virtual Equipment), 컴퓨터 저장 매체 또는 장치, 또는 전송되는 신호 파(Signal Wave)에 영구적으로, 또는 일시적으로 구체화(Embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.

[0091] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(Magnetic Media), CD-ROM, DVD와 같은 광기록 매체(Optical Media), 플롭티컬 디스크(Floptical Disk)와 같은 자기-광 매체(Magneto-optical Media), 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 상기된 하드웨어 장치는 실시예의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

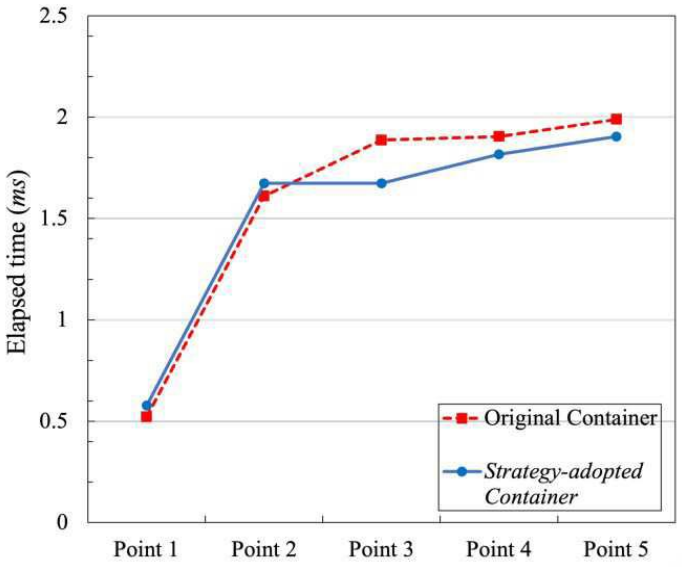
[0092] 본 발명은 도면에 도시된 실시예를 참고로 설명되었으나 이는 예시적인 것에 불과하며, 본 기술 분야의 통상의 지식을 가진 자라면 이로부터 다양한 변형 및 균등한 타 실시예가 가능하다는 점을 이해할 것이다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성 요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다. 따라서, 본 발명의 진정한 기술적 보호 범위는 첨부된 등록청구범위의 기술적 사상에 의해 정해져야 할 것이다.

도면

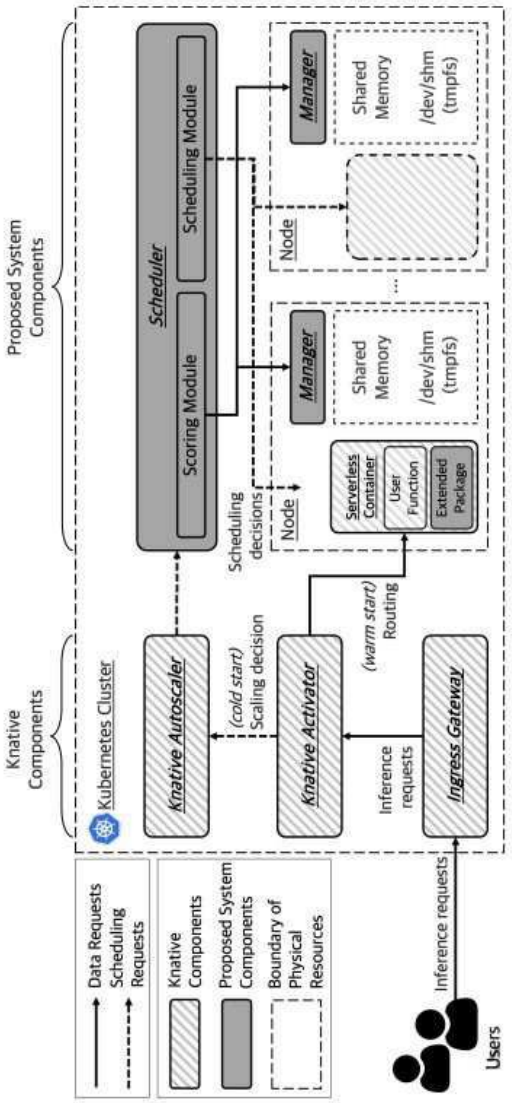
도면1



도면2


2.5
2
1.5
1
0.5
0
Elapsed time (ms)
Point 1
Point 2
Point 3
Point 4
Point 5
Original Container
Strategy-adopted Container

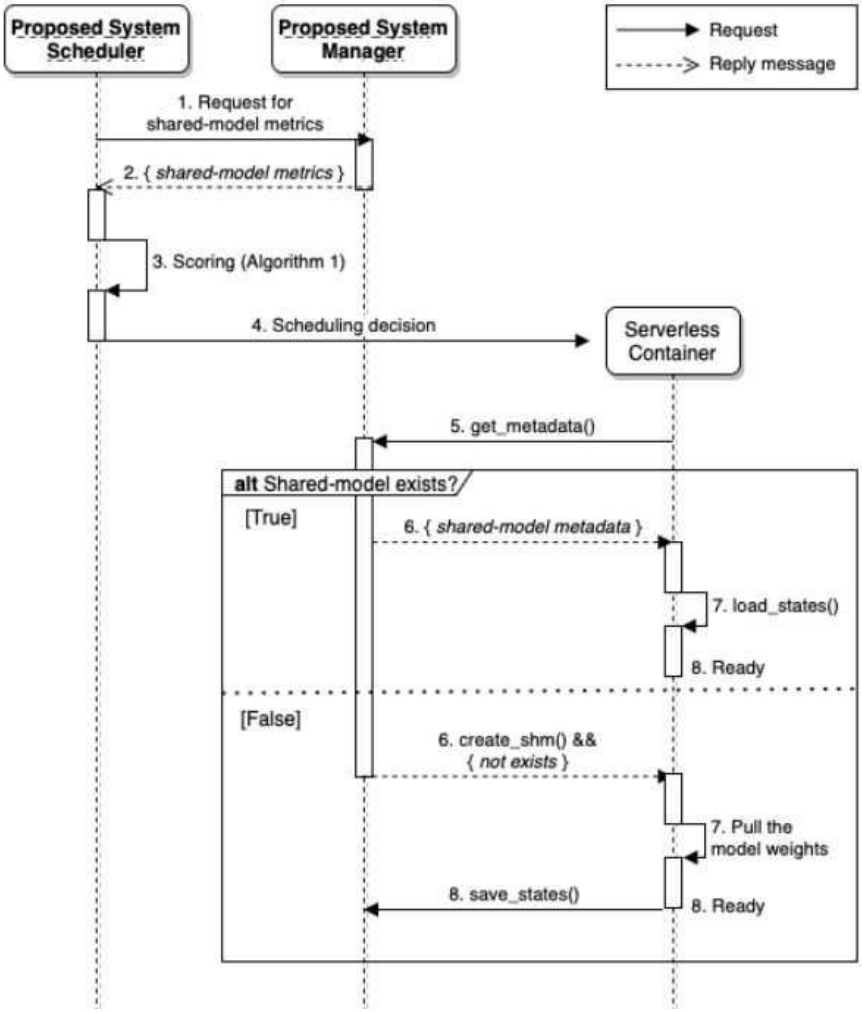
도면3


Proposed System Components
Knative Components
Kubernetes Cluster
Scheduler
Scheduling Module
Scoring Module
Knative Autoscaler
Knative Activator
Ingress Gateway
Manager
Shared Memory
/dev/shm (tmpfs)
Node
Serverless Container
User Function
Extended Package
Scheduling decisions
(cold start) Scaling decision
(warm start) Routing
Inference requests
Users
Data Requests
Scheduling Requests
Knative Components
Proposed System Components
Boundary of Physical Resources

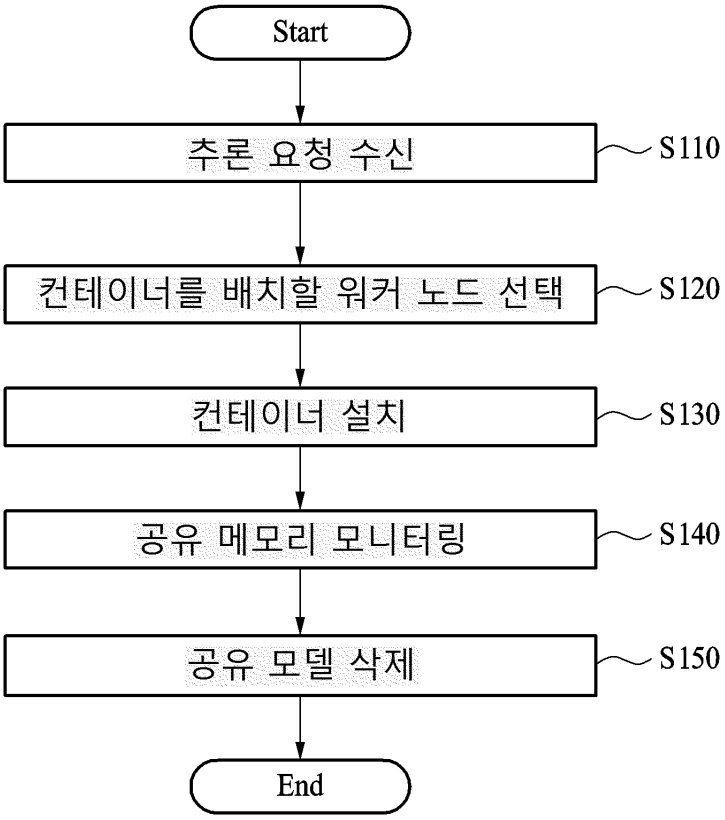
도면4


```
{
    "shm_sum": 553438412,
    "shm_disk": {
      "used": 553438412,
      "free": 16216351540,
      "all": 16769789952
    },
    "model_count": 1,
    "models": [{
        "model_name": "vgg16-based",
        "tag_name": "v0.0.1",
        "shmname": "shm_c01dca5b",
        "shmsize": 553438412
    }]
}
```


도면5



도면6


Start
추론 요청 수신
S110
컨테이너를 배치할 워커 노드 선택
S120
컨테이너 설치
S130
공유 메모리 모니터링
S140
공유 모델 삭제
S150
End