

(19) 대한민국특허청(KR)
(12) 공개특허공보(A)(11) 공개번호 10-2025-0083990
(43) 공개일자 2025년06월10일

(51) 국제특허분류(Int. Cl.)
G06N 3/098 (2023.01) G06N 3/09 (2023.01)
H04L 67/10 (2022.01)
(52) CPC특허분류
G06N 3/098 (2023.01)
G06N 3/09 (2023.01)
(21) 출원번호 10-2024-0039731
(22) 출원일자 2024년03월22일
심사청구일자 2024년03월22일
(30) 우선권주장
1020230172602 2023년12월01일 대한민국(KR)

(71) 출원인
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암
동5가)
(72) 발명자
김형준
서울특별시 강북구 삼양로19길 113, 111동 604호
유현창
서울특별시 노원구 노원로 214, 6동 202호
(74) 대리인
김등용

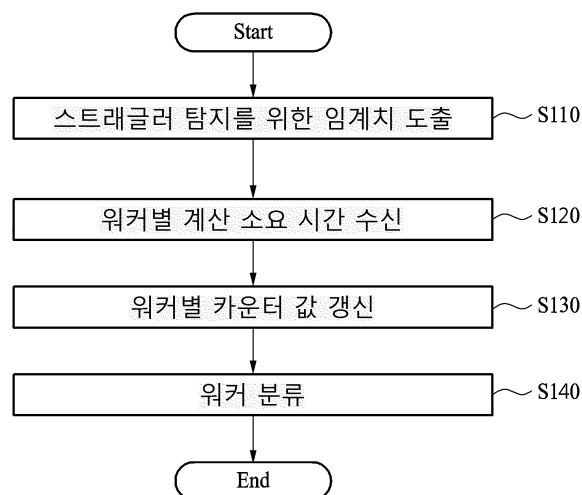
전체 청구항 수 : 총 7 항

(54) 발명의 명칭 분산 딥러닝 환경에서의 적응적 워커 분류 장치 및 방법

(57) 요약

분산 딥러닝(Distributed Deep Learning) 환경에서, 루트 노드(root node)에 의해 수행되는 워커 분류 방법이 개시된다. 상기 워커 분류 방법은 a) 스트래글러(stragger) 탐지를 위한 임계치가 도출되는 단계, b) 워커별 계산 소요 시간이 수신되는 단계, c) 워커별 카운터 값이 갱신되는 단계, 및 d) 스트래글러가 탐지되는 단계를 포함하고, 상기 a) 단계는 에포크의 초반 n (n 은 임의의 자연수) 이터레이션 동안 수행되고, 상기 b) 내지 d) 단계는 초반 n 이터레이션 이후의 이터레이션 동안 반복적으로 수행된다.

대표도 - 도6



(52) CPC특허분류

H04L 67/10 (2022.05)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711193387
과제번호	2022-0-01198-002
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	정보통신방송혁신인재양성
연구과제명	융합보안대학원(고려대학교)
과제수행기관명	고려대학교산학협력단
연구기간	2023.01.01 ~ 2023.12.31

이 발명을 지원한 국가연구개발사업

과제고유번호	1711179291
과제번호	00156439
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	정보통신방송혁신인재양성
연구과제명	ICT혁신인재4.0(고려대학교)
과제수행기관명	고려대학교산학협력단
연구기간	2023.01.01 ~ 2023.12.31

명세서

청구범위

청구항 1

분산 딥러닝(Distributed Deep Learning) 환경에서, 루트 노드(root node)에 의해 수행되는 워커 분류 방법에 있어서,

- a) 스트래글러(straggle) 탐지를 위한 임계치가 도출되는 단계;
- b) 워커별 계산 소요 시간이 수신되는 단계;
- c) 워커별 카운터 값이 갱신되는 단계; 및
- d) 스트래글러가 탐지되는 단계를 포함하고,

상기 a) 단계는 에폭의 초반 n (n 은 임의의 자연수) 이터레이션 동안 수행되고, 상기 b) 내지 d) 단계는 초반 n 이터레이션 이후의 이터레이션 동안 반복적으로 수행되는,

워커 분류 방법.

청구항 2

제1항에 있어서,

상기 a) 단계는,

워커들 각각으로부터 이터레이션을 수행하는 과정에서 소요되는 시간인 계산 시간을 수신하는 단계; 및

이터레이션 별로 가장 짧은 계산 시간에 대한 평균에 미리 정해진 상수를 곱하여 상기 임계치를 도출하는 단계를 포함하는,

워커 분류 방법.

청구항 3

제1항에 있어서,

상기 c) 단계는, 상기 임계치를 초과하는 계산 시간을 송신한 워커의 카운터 값을 증가시키는,

워커 분류 방법.

청구항 4

제1항에 있어서,

상기 c) 단계는, 상기 임계치 미만의 계산 시간을 송신한 워커의 카운터 값을 감소시키는,

워커 분류 방법.

청구항 5

제1항에 있어서,

상기 d) 단계는, 미리 정해진 기준 카운터 값과 같거나 큰 카운터 값을 갖는 워커를 스트래글러로 분류하는,

워커 분류 방법.

청구항 6

제5항에 있어서,

상기 d) 단계는, 스트래글러로 분류된 워커의 학습 참여를 배제하는,

위커 분류 방법.

청구항 7

제1항에 있어서,

상기 a) 단계가 제1 에포크 이후의 임의의 에포크에서 수행되는 경우, 상기 a) 단계는,

이전의 에포크 동안 스트래글러로 분류된 위커를 포함한 모든 위커로부터 계산 소요 시간을 수신하는 단계;

이전의 에포크에서 도출된 임계치를 이용하여, 위커별 카운터 값을 갱신하는 단계; 및

기준 카운터 값 미만의 카운터 값을 갖는, 스트래글러로 분류된 위커를 정상 위커로 분류하는 단계를 포함하는,

위커 분류 방법.

발명의 설명

기술 분야

[0001] 본 발명은 분산 딥러닝 환경에서의 위커 분류 장치에 관한 것으로, 특히 분산 딥러닝 환경에서 속도 저하를 유발하는 위커를 적응적으로 탐지하고 분류하는 장치 및 방법에 관한 것이다.

배경 기술

[0002] 딥러닝(Deep Learning)은 자율주행, 신약개발, 그리고 자연어 처리에 이르기까지 다양한 분야에서 활용되고 있다. 딥러닝 모델 크기와 학습 데이터의 양이 급속도로 증가하는 상황에서 적절한 시간과 비용의 투입으로 양질의 모델을 얻기 위해서는 멀티 노드 및 멀티 GPU를 활용하여 학습 작업의 효율적 처리를 뒷받침하는 분산 딥러닝(Distributed Deep Learning) 기술이 필요하다.

[0003] 데이터 병렬화(Data parallelism)는 분산 딥러닝의 주류를 이루고 있는 방식으로, 학습 로직 상 병렬화 가능한 부분의 작업을 나눠서 동시에 처리함으로써 작업 소요 시간을 대폭 단축시킬 수 있다. 데이터 병렬화는 동종(Homogeneous) 환경에서 빠른 처리 속도를 보인다. 하지만, 학습 속도의 저하를 유발하는 스트래글러(Straggler) 발생 시 전체 작업 속도가 가장 느린 위커를 기준으로 맞춰지게 된다. 이를 스트래글러 문제라고 하며, 이는 각 위커의 계산 결과를 합쳐서 매개변수 업데이트를 진행하는 동기화의 특성에 의해 발생한다. 심각한 속도 저하를 보이는 위커는 분산 딥러닝의 도입 자체를 무의미하게 만든다는 점에서 스트래글러 문제는 분산 딥러닝 분야에서 해결해야 할 주요 과제로 여겨진다.

[0004] 스트래글러 문제 해결의 핵심은 정상 속도인 위커와 스트래글러의 동기화 작업을 최적화된 주기로 수행하는 것이다. 이를 위해서는 어떤 위커가 스트래글러인지 분류해내는 작업이 필수적이다. 하지만, 다음의 두 가지 이유로 인해 적절한 스트래글러 탐지 방법을 제시하는 데 어려움이 따른다. 첫째로, 위커가 보이는 속도 저하의 정도와 발생 시점은 임의적이다. 둘째로, 정상 위커의 처리 속도 범위가 시스템 상황에 따라 변화하므로 정적인 값을 통해 스트래글러 여부를 판단할 수 없다.

[0005] 본 발명에서는 위커 성능 프로파일링 기반의 임계치를 이용한 동적 스트래글러 탐지 기법을 제안하여 위커의 속도 저하에 강건한 분산 딥러닝 학습이 이루어질 수 있게 한다. 이를 통해 학습이 진행되는 시점의 시스템 상황을 기준으로 한 임계치를 적응적으로 찾고 스트래글러를 신속하게 탐지할 수 있다.

선행기술문헌

특허문헌

[0006] (특허문헌 0001) 대한민국 공개특허 제2023-0099543호 (2023.07.04. 공개)
(특허문헌 0002) 대한민국 공개특허 제2020-0068562호 (2020.06.15. 공개)
(특허문헌 0003) 대한민국 등록특허 제2296319호 (2021.08.31. 공고)
(특허문헌 0004) 대한민국 공개특허 제2023-0055271호 (2023.04.25. 공개)

발명의 내용

해결하려는 과제

[0007] 본 발명이 이루고자 하는 기술적인 과제는 분산 딥러닝 환경에서 적응적 워커 분류 장치 및 방법을 제공하는 것이다.

과제의 해결 수단

[0008] 본 발명의 일 실시예에 따른, 분산 딥러닝(Distributed Deep Learning) 환경에서, 루트 노드(root node)에 의해 수행되는 워커 분류 방법은 a) 스트래글러(straggle) 탐지를 위한 임계치가 도출되는 단계, b) 워커별 계산 소요 시간이 수신되는 단계, c) 워커별 카운터 값이 갱신되는 단계, 및 d) 스트래글러가 탐지되는 단계를 포함하고, 상기 a) 단계는 에포크의 초반 n (n 은 임의의 자연수) 이터레이션 동안 수행되고, 상기 b) 내지 d) 단계는 초반 n 이터레이션 이후의 이터레이션 동안 반복적으로 수행된다.

발명의 효과

[0009] 본 발명의 실시예에 따른, 분산 딥러닝 환경에서 적응적 워커 분류 장치 및 방법에 의할 경우, 신속하고 정확하게 실시간으로 워커의 상태를 분류할 수 있다.

[0010] 또한, 이를 통해 동기화 과정에서 대기 시간을 단축할 수 있어, GPU 평균 이용률을 높이고 느린 워커(즉, 스트래글러)로 인한 작업 지연을 최소화할 수 있다.

도면의 간단한 설명

[0011] 본 발명의 상세한 설명에서 인용되는 도면을 보다 충분히 이해하기 위하여 각 도면의 상세한 설명이 제공된다.

도 1은 스트래글러 개수별 학습 정확도 추이를 나타내는 그래프를 도시한다.

도 2는 워커별 배치 사이즈 32 실험에서 임계치 설정 이후 스트래글러의 카운터 값 변화 추이를 나타내는 그래프를 도시한다.

도 3은 워커별 배치 사이즈 64 실험에서 임계치 설정 이후 스트래글러의 카운터 값 변화 추이를 나타내는 그래프를 도시한다.

도 4는 워커별 배치 사이즈 32 실험에서 속도 저하 해제 이후 정상 상태 회복 워커의 카운터 값 변화 추이를 나타내는 그래프를 도시한다.

도 5는 워커별 배치 사이즈 64 실험에서 속도 저하 해제 이후 정상 상태 회복 워커의 카운터 값 변화 추이를 나타내는 그래프를 도시한다.

도 6은 본 발명의 일 실시예에 따른, 분산 딥러닝 환경에서 적응적 워커 분류 방법을 설명하기 위한 흐름도이다.

발명을 실시하기 위한 구체적인 내용

[0012] 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.

[0013] 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.

[0014] 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.

- [0015] 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.
- [0016] 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.
- [0017] 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.
- [0018] 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.
- [0019] 스트래글러(Straggler) 문제는 분산 딥 러닝 분야의 대표적인 문제이다. 특히, 스트래글러는 임의의 시점에 불특정한 워커에서 발생하기 때문에 스트래글러의 존재 여부를 미리 파악하는 것은 불가능하다. 우선, 분산 딥러닝 분야에서는 스트래글러 문제로 인한 성능 저하를 완화하기 위해 기법들을 소개한다.
- [0020] 1. 비동기 탈중앙 기법
- [0021] 비동기 방식의 분산 딥러닝은 전체 워커의 일부를 서브그룹으로 구성하여 동기화를 수행하기 때문에 서브그룹에 스트래글러가 포함되지 않는 경우 스트래글러의 영향을 받지 않는다. 또한, 동기식 방법에 비해 스트래글러로 인한 영향을 상대적으로 적게 받는다. 하지만 기율기 계산 시작 시점과 가중치 업데이트 시점의 모델이 달라지는 경우 수학적으로 부정확한 값이 사용되는 문제가 발생하며, 이는 낮은 수렴율로 이어진다.
- [0022] Lian 등은 워커가 그래디언트 계산을 완료하면 다른 워커 하나를 임의로 선택해 모델 평균화 작업을 수행하는 AD-PSGD 알고리즘을 제안하였다(X. Lian, W.i Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent", International Conference on Machine Learning, PMLR, Stockholm, Sweden, Vol. 80, pp. 3043-3052, Jul. 2018.). AD-PSGD는 SGD와 같이 $O(1/\sqrt{k})$ 로 수렴하며, 올-리듀스 방식에 비해 스트래글러에 대해 강건함을 실험을 통해 증명하였지만 동종 환경에서 올-리듀스 방식에 비해 3배 이상 느리다는 문제가 있다.
- [0023] 2. 그래디언트 코딩 기법
- [0024] 그래디언트 코딩은 중복 계산을 통해 스트래글러의 계산 완료 시점까지 대기하지 않고도 정확한 기율기 값을 구할 수 있도록 하는 방법이다(R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning", International Conference on Machine Learning, PMLR, Sydney NSW Australia, Vol. 70, pp. 3368-3376, Aug. 2017.). 이 방법은 스트래글러의 무작위적 속도 저하에 직접적인 영향을 받지 않는다는 장점이 있다. 하지만, 더 많은 양의 계산 결과를 네트워크 상으로 전송하여 계산 및 통신 부하를 증가시킨다. 또한, 감당할 수 있는 스트래글러 개수의 상한이 존재한다는 한계를 갖는다.
- [0025] 3. 부분적 올-리듀스 기법
- [0026] 부분적 올-리듀스 기법은 스트래글러의 동기화 참여가 전체 작업 지연의 원인이라는 점에 주목하여 스트래글러 탐지 후 해당 워커를 다른 워커들과의 동기화 과정에서 일시적 또는 영구적으로 배제시키는 방식이다(X. Miao, et al., "Heterogeneity-aware distributed machine learning training via partial reduce", Proc. of the

2021 International Conference on Management of Data, New York, United States, pp. 2262-2270, Jun. 2021.). 이러한 방식은 스트래글러가 존재할 때 일반적인 동기식 방법론에 비해 총 작업 시간을 크게 단축시킨다는 장점이 있으나, 스트래글러를 동기화 작업에서 배제하므로 할당된 데이터를 학습에 사용하지 못할 수 있으며 이는 수렴률 저하의 요인이 된다.

[0027] 도 1은 스트래글러 개수별 학습 정확도 추이를 나타내는 그래프를 도시한다.

[0028] 도 1의 실험은 텐센트 클라우드 환경에서 수행하였으며 GN7.20XLARGE320 인스턴스 1개가 사용되었다. GPU는 NVIDIA T4를 총 4개 사용하였다. 분산 딥러닝 학습 프레임워크인 Horovod(A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow", arXiv preprint arXiv:1802.05799, Feb. 2018. <https://doi.org/10.48550/arXiv.1802.05799>.)로 ResNet-18(K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", Proc. of the IEEE conference on computer vision and pattern recognition, Las Vegas, NV, USA, pp. 770-778, Jun. 2016.) 모델을 총 90 에폭 동안 학습시키는 작업을 수행하였다. 데이터셋은 CIFAR-10을 사용하였다. 스트래글러는 Python의 time 라이브러리에서 sleep 함수를 사용하여 구현하였으며, 속도 저하의 정도는 정상 워커의 5배 수준으로 설정하였다.

[0029] 도 1의 실험에서 확인할 수 있는 중요한 사항은 2가지이다. 첫째, 전체 작업의 수행 시간이 스트래글러의 속도 저하 정도에 비례해서 증가한다는 점이다. 스트래글러가 존재하지 않을 때, 총 작업 수행 시간은 990.97초를 기록한 반면 스트래글러가 1개일 때 총 작업 수행 시간은 4650.11초를 기록하였다. 이는 4.69배 증가한 수치로, 스트래글러의 속도 저하 정도와 유사한 크기의 값이다. 둘째, 스트래글러의 개수가 1개이더라도 스트래글러의 개수가 더 많은 경우와 같은 수준으로 전체 작업 시간을 증가시킨다는 점이다. 스트래글러가 2개, 3개, 4개일 때 총 작업 수행 시간은 4693.09초, 4684.54초, 4698.86초를 기록하였다. 이는 스트래글러가 존재하지 않는 경우 대비 4.74, 4.73, 4.74배에 해당하는 수치이다.

[0030] 이 결과는 스트래글러가 단 1개만 존재하더라도 전체 작업 시간이 큰 폭으로 증가함을 보여준다. 이처럼 스트래글러가 1개만 존재하더라도 모든 워커가 스트래글러인 경우와 같은 수준의 심각한 속도 저하를 초래하므로 모든 스트래글러를 신속하고 정확하게 탐지 및 처리하는 기법이 반드시 필요함을 알 수 있다.

[0031] 이하에서는, 본 발명에서 제안하는 동적 워커 분류 기법을 소개한다.

[0032] 1. 스트래글러 탐지

[0033] 워커 속도 저하에 강건한 분산 딥러닝을 위해서는 어떤 워커가 스트래글러인지 탐지해 내는 작업이 필수적으로 이루어져야 한다. 하지만, 예측 기반 방법론은 스트래글러 탐지에 사용하기에 부적합하다. 기존의 통계 기반 또는 인공지능 예측 기법은 고차원 데이터의 복잡한 패턴을 파악하는 데 강점을 보인다. 그러나 분산 딥러닝 환경에서 워커의 작업 처리 속도가 저하되는 시점과 정도는 모두 임의적이다. 또한, 워커의 작업 처리 속도는 디바이스 스펙 뿐만 아니라 네트워크 혼잡, 발열 등 다양한 요인에 의해 수시로 변동한다. 이는 스트래글러의 정확한 탐지를 더 어렵게 만든다.

[0034] 본 발명은 가변적인 시스템 상황에 맞추어 신속하게 대응할 수 있는 스트래글러 탐지 기법을 제안한다. 제안 기법을 통한 스트래글러 탐지는 두 단계로 진행될 수 있다.

[0035] 첫 번째 단계에서는 임계치를 결정한다. 임계치 설정의 구체적인 과정은 다음과 같다. 먼저, 워커의 성능을 프로파일링하는 작업이 수행된다. 성능 지표는 Python의 time 함수를 이용해서 측정한 계산 시간이며, 측정은 이터레이션마다 이루어진다. 즉, 계산 시간은 단일의 이터레이션을 수행하는데 소요되는 시간을 의미할 수 있다. 여기서, 가장 빠른 워커의 기록은 현재 시스템 환경에서 정상 워커의 성능을 나타낸다. 본 발명에서는, 한 에폭 동안의 정상 워커 성능을 파악하기 위해 에폭 초반 n (n 은 임의의 자연수) 이터레이션 동안의 최단 기록들을 모아 그 평균값을 구한다. 임계치는 산출된 평균값에 상수 k (k 는 임의의 실수)를 곱한 값으로 결정될 수 있다.

[0036] n 과 k 의 값은 실험에서 일반적으로 좋은 성능을 보인 $n=5$, $k=2$ 를 기본값으로 한다. 하지만, 다음의 트레이드오프 관계를 고려하여 사용자가 요구사항에 맞게 설정할 수도 있다. 즉, 본 발명의 권리범위가 n 및/또는 k 의 값에 제한되는 것은 아니며, 실시예에 따라, n 및/또는 k 의 값은 상이하게 변경될 수 있다. n 값이 클수록 스트래글러 탐지까지 소요되는 시간이 증가하지만, 간헐적 속도 저하를 보이는 정상 워커를 스트래글러로 오분류하는 경우를 줄일 수 있다. k 값의 경우, 스트래글러 분류 기준이 되는 속도 저하의 정도를 나타낸다. 따라서 k 값을 크게 설정한다면 높은 신뢰도의 탐지 결과를 얻지만, 그보다 경미한 수준의 워커 속도 저하가 전체 작업 시간을

증가시키는 것을 허용한다.

[0037] 두 번째 단계에서는 카운터 기반 방식으로 스트래글러를 탐지한다. 알고리즘 1에 스트래글러 탐지 과정에 대한 의사코드로 제시되어 있다. 카운터는 워커마다 별도로 관리될 수 있으며, 이터레이션마다 측정되는 워커의 계산 시간이 임계치를 초과할 때마다 워커 카운터의 값을 1만큼 증가시킬 수 있다. 이는 알고리즘 1의 8~16번째 줄의 내용에 해당한다. 만약, 워커의 카운터 값이 일정 값에 도달하면 알고리즘 1의 17~18번째 줄에서와 같이 해당 워커를 스트래글러로 분류한다. 스트래글러 분류 기준이 되는 카운터 값은 사용자가 직접 정하거나 에포크를 이루는 전체 이터레이션의 일정 비율로 정할 수 있다. 본 발명의 제안 기법은 이와 같이 사용자의 개입을 필요로 하지 않는 자율적인 방식으로 스트래글러를 탐지할 수 있다.

[0038] 한 가지 특기할 사항은 분산 딥러닝에서 워커들의 작업은 각각 별도의 프로세스일 수 있다는 점이다. 이 경우, 가장 빠른 워커 계산 시간을 구하기 위해 프로세스 간 통신이 이루어져야 함을 의미한다. 본 발명에서는 MPI(Message Passing Interface) 통신 그룹을 구성하여 워커들이 서로 값을 교환할 수 있는 메커니즘을 구현할 수 있다. MPI(Walker, David W., and Jack J. Dongarra. "MPI: a standard message passing interface." Supercomputer 12 (1996): 56-68)는 표준 메시지 전달 인터페이스로, 프로세스 그룹에 속한 워커들 간의 점대점(point-to-point) 또는 집합적 통신 루틴을 지원한다.

[0039] 임계치 설정에 있어 워커들 간의 MPI 통신은 다음과 같이 이루어질 수 있다. 먼저, 각 워커가 자신의 계산 시간을 측정한 후 루트 워커에게 전송한다. 다음으로, 루트 워커가 전송받은 계산 시간 중 최솟값을 토대로 구한 임계치를 나머지 모든 워커들에게 브로드캐스트한다. 이 과정이 에포크마다 이루어짐으로써 시스템 여건 변화를 반영한 임계치가 모델 학습 전 과정에 걸쳐 스트래글러 탐지에 적용될 수 있다.

[0040] [알고리즘 1]

Algorithm 1: Dynamic straggler detection with counter

```

Input:: A set of workers  $W = \{w_i | 1 \leq i \leq n\}$ 
1: for worker  $w_i \in W$  do
2:   counter  $c_i = 0$ 
3: end for
4: while not reached convergence do
5:   if  $c_i > \text{limit}$  and  $w_i$  is a straggler then
6:     continue
7:   end if
8:   Record local time  $t_i$ 
9:   Compute gradients over selected samples
10:  Average gradients computed by  $w_i \in W$ 
11:  Record local time  $t_j$ 
12:  if  $t_j - t_i > \text{threshold}$  then
13:     $c_i += 1$ 
14:  else if  $c_i > 0$  and  $t_j - t_i < \text{threshold}$  then
15:     $c_i -= 1$ 
16:  end if
17:  if  $c_i == \text{limit}$  then
18:    classify  $w_i$  as a straggler
19:  end if
22: end while
  
```

[0041]

[0042] 2. 워커 상태 회복

[0043] 분산 딥러닝에서, 탐지된 스트래글러는 남은 학습 과정으로부터 제외시키는 것이 일반적이다. 스트래글러로 인한 작업 속도 저하를 최소화하기 위해서이다. 하지만, 스트래글러로 분류된 워커가 정상 상태로 회복할 수도 있

다. 이 경우, 해당 워커를 작업에 다시 복귀시킴으로써 학습 시간을 단축시킬 수 있다.

- [0044] 스트래글러로 분류된 워커가 정상 상태로 회복했음을 파악하기 위해서는 해당 워커의 작업 처리 속도를 지속적으로 모니터링하여야 한다. 본 발명의 제안 기법에서는, 이를 위해 새로운 에포크의 초반 n 이터레이션 동안 스트래글러를 포함하여 학습을 진행하고 워커마다 계산 소요 시간을 측정한다. 워커의 계산 소요시간이 임계치 이하인 경우, 카운터 값을 1만큼 감소시킬 수 있다. 따라서, 스트래글러로 분류되었던 워커가 정상 속도를 회복한다면 카운터 값을 스트래글러 분류 기준값 미만으로 감소시킬 수 있고, 다시 모델 학습 작업에 복귀할 수 있다.
- [0045] 아직 회복하지 못한 스트래글러의 동기화 참여는 전체 작업의 속도를 저하시킨다. 하지만, GPU는 고가의 강력한 계산 자원이므로 제한된 횟수의 이터레이션 동안 위험을 감수하고 스트래글러로 분류된 워커의 회복 가능성을 확인할 수 있다.
- [0046] 3. 정상 워커 오분류 최소화
- [0047] 본 발명의 제안 기법은 워커가 최근 기록한 속도 저하의 빈도를 기준으로 스트래글러를 탐지함으로써 정상 워커가 스트래글러로 오분류되는 경우를 최소화하고자 한다. 이를 구현하기 위한 핵심 아이디어는 총 두 가지이다.
- [0048] 첫째로, 카운터 값의 증감 조건을 모두 포함하는 메커니즘의 적용이다. 단순히 임계치를 초과한 경우를 누적 집계한다면 카운터 값은 단조 증가하게 된다. 이는 워커 성능의 최악의 경우에만 주목하는 방식이다. 이 방식의 문제점은 정상 워커를 스트래글러로 분류하는 1종 오류를 범할 위험이 크다는 점이다. 계산 속도에 영향을 미치는 요소는 다양하므로 정상 워커 또한 간헐적인 속도 저하를 보일 수 있다. 따라서, 최악의 경우만을 누적 집계한다면 작업의 후반부에 대다수의 워커를 스트래글러로 분류하게 될 위험이 있다. 또한, 이 위험은 작업의 규모 및 소요 시간에 비례하여 증가한다. 따라서, 본 발명은 워커가 기록한 시간이 임계치 미만인 경우 카운터 값을 차감할 수 있다. 이를 통해, 워커의 총 속도 저하 횟수가 아닌 최근 기록한 속도 저하의 빈도를 기준으로 스트래글러 탐지가 이루어질 수 있다.
- [0049] 두 번째는 카운터 값의 범위 제한이다. 제안 기법에서 카운터 값은 0 이상 스트래글러 탐지 기준 이하의 범위를 벗어날 수 없다. 이러한 제한을 두는 목적은 워커 상태 변화에 대한 신속한 대응이다. 범위 제한이 없다면 상태 지속 시간에 비례하여 카운터 값이 무제약적으로 증가 또는 감소한다. 이때 워커 상태가 변화한다면 임계치를 기준으로 스트래글러의 발생 또는 정상 워커로의 회복 판단이 이루어지기까지 많은 시간이 소요된다. 스트래글러의 분류 지연은 전체 작업 시간 증가로, 정상 워커의 제합류 지연은 계산 자원 낭비 및 잠재적 성능 향상의 미실현으로 이어진다. 따라서 제안 기법은 카운터 값의 범위 제한을 통해 카운터가 워커 상태 변화에 항시 기민하게 반응할 수 있다.
- [0050] 다만, 카운터 값의 상한은 스트래글러 탐지 기준에 미리 정해진 일정 수치(p , p 는 임의의 자연수)를 더한 값이 될 수도 있다. 이를 통하여, 대부분의 상태가 스트래글러인 워커가 마지막 단계에서만 정상 워커로 판단되어, 카운터 값이 스트래글러 탐지 기준 미만이 되는 것을 방지할 수 있다.
- [0051] 실험 및 평가
- [0052] 실험은 도 1에서와 같이 텐센트 클라우드 환경에서 GN7.20XLARGE320 인스턴스 1개를 사용하여 수행되었다. GPU는 NVIDIA T4를 4개 사용하였다. 데이터셋은 CIFAR-10으로 ResNet-18 모델을 총 10 에포크 동안 학습시키는 작업을 수행하였으며, 분산 딥러닝 학습 프레임워크인 Horovod를 사용하였다. 스트래글러는 Python의 time 라이브러리에서 sleep 함수를 사용하여 구현하였다.
- [0053] 실험에 사용된 매개변수는 다음과 같이 설정하였다. 먼저, Nesterov 모멘텀(Y. Nesterov, "Introductory lectures on convex optimization: A basic course", Springer Science & Business Media, Vol. 87, 2003.)을 적용하였으며 $m = 9$ 를 사용하였다. 가중치 감쇠를 위해서는 $\lambda = 0.00005$ 를 사용하였다. 학습률을 위해서는 레퍼런스 학습률(P. Goyal, et al., "Accurate, large minibatch sgd: Training imagenet in 1 hour", arXiv preprint arXiv:1706.02677, Jun. 2017. <https://doi.org/10.48550/arXiv.1706.02677>.)을 사용하였고, 웹-업 기법을 학습 초반 5번의 에포크 동안 적용하였다. 속도 저하의 정도는 정상 워커의 3배 수준으로 설정하였다. 임계치 설정을 위해 곱하는 상수값으로는 2(즉, $k = 2$)를 사용하였으며, 각 에포크마다 초반 5번(즉, $n = 5$) 이터레이션 동안의 모니터링을 통해 워커 퍼포먼스 프로파일링 작업을 수행하였다.
- [0054] 실험은 제안하는 워커 분류 기법이 스트래글러의 발생과 스트래글러의 정상 워커로의 회복을 모두 신속하고 정확하게 탐지하는지 확인하기 위해 다음과 같이 설계되었다. 먼저, 에포크의 전체 이터레이션 횟수 중 절반만큼은 임의의 워커 1개를 스트래글러로 지정하여 3배만큼 속도를 저하시킨다. 이때, 에포크 시작 시간과 스트래글

러 탐지가 이뤄진 시간을 측정함으로써 스트래글러 탐지까지 소요된 시간을 파악할 수 있다. 다음으로, 에포크의 전체 이터레이션 횟수 중 나머지 절반만큼은 스트래글러로 지정된 워커에 적용되었던 속도 저하를 해제한다. 여기서, 속도 저하를 해제시킨 첫 이터레이션의 시작 시간과 첫 카운터 감소가 이루어진 시간을 각각 기록할 수 있으므로 그 차를 통해 스트래글러의 정상 상태 회복을 감지하는 데 소요된 시간을 알 수 있다. 마지막으로, 지정된 스트래글러가 에포크의 전반부에서 학습 도중 정상 상태를 회복한 것으로 분류되는 경우, 그리고 정상 워커가 스트래글러로 분류되는 경우를 합하여 오분류가 이루어진 경우를 파악할 수 있다. 오분류의 발생 횟수 및 빈도를 통해 제안하는 워커 분류 기법의 분류 정확도를 평가할 수 있다. 실험은 제안 기법이 배치 사이즈에 무관하게 잘 동작하는지 검증하기 위해 워커별 배치 사이즈가 32인 경우와 64인 경우 두 차례에 걸쳐 진행되었다.

[0055] 도 2와 도 3은 에포크 1에서 임계치가 설정된 시점부터 스트래글러 탐지가 이뤄진 시점까지 스트래글러로 지정된 워커의 카운터 값의 변화 추이를 나타내는 그래프를 도시한다.

[0056] 붉은 색으로 표시된 구간은 스트래글러 발생 시점부터 탐지 이전까지를 나타낸다. 임계치 설정과 동시에 카운터 값 조정이 이루어지므로 카운터 값이 1인 상태에서 시작함을 확인할 수 있다. 배치 사이즈 32, 64인 경우 각각 1.66초, 1.00초가 경과된 시점에, 이터레이션 횟수 기준으로는 각각 10 이터레이션, 6 이터레이션만에 스트래글러 탐지를 완료하였다. 전체 학습이 10 에포크 동안 진행되는 과정에서 스트래글러 탐지 동안 10번 이루어졌으며, 임계치 설정 시점부터 스트래글러 탐지까지 소요된 시간은 배치 사이즈 32, 64인 경우 각각 평균 1.43초, 1.44초를 기록하였다.

[0057] 도 4와 도 5는 에포크 1에서 중반 이후 스트래글러의 속도 저하가 해제된 시점부터 카운터 값 감소가 이루어진 시점까지 해당 워커의 카운터 값의 변화 추이를 나타낸다. 붉은 색으로 표시된 영역은 워커 회복 시작 시점부터 상태 회복이 확인된 시점까지를 나타낸다. 도 4와 도 5에서 확인할 수 있듯이 배치 사이즈 32, 64인 경우 각각 0.21초, 0.30초만에 스트래글러로 분류되었던 워커가 정상 속도를 회복하였음을 감지하였다.

[0058] 10 에포크 동안 스트래글러 탐지에 실패하거나 정상 워커가 스트래글러로 오분류된 경우, 그리고 스트래글러로 분류된 워커가 속도 저하가 적용된 전반부에서 도중에 정상 워커로 오분류된 경우 모두 발생하지 않았다. 마찬가지로, 정상 속도 회복 감지 역시 전체 학습이 10 에포크 동안 진행되는 과정에서 모두 분류에 성공하였다.

[0059] 도 6은 본 발명의 일 실시예에 따른 워커 분류 방법을 설명하기 위한 흐름도이다.

[0060] 도 6에 도시된 워커 분류 방법은 적어도 프로세서(processor) 및/또는 메모리(memory)를 포함하는 컴퓨팅 장치에 의해 수행될 수 있다. 따라서, 워커 분류 장치로 명명될 수도 있는 컴퓨팅 장치는 PC(Personal Computer), 서버(server) 등을 포함한다. 실시예에 따라, 컴퓨팅 장치는 클라우드 환경에서 가상화된 가상 머신이나 가상 머신에 의해 제공되는 컨테이너 등으로 구현될 수도 있다.

[0061] 또한, 컴퓨팅 장치에 의해 구현되는 노드들(루트 노드(root node) 및/또는 워커 노드들(worker nodes))은 개별적으로 분리된 하드웨어로 구현되거나, 또는 노드들 중 적어도 일부는 하나의 하드웨어로 구현될 수도 있다(즉, 논리적으로 구분될 수도 있다). 따라서, 노드들 중 적어도 둘은 유무선 통신망을 통하여 통신하거나, 프로세스 간 통신(예컨대, MPI) 프로토콜을 통해 통신할 수 있다. 또한, 노드들 중 루트 노드는 워커 분류 방법을 수행하는 워커 분류 장치를 의미할 수 있다. 이하에서, 워커 분류 방법을 설명함에 있어 앞선 기재와 중복되는 내용에 관하여는 그 구체적인 기재를 생략하기로 한다.

[0062] 우선, 스트래글러 탐지를 위한 임계치가 도출된다(S110). 임계치는 워커 노드를 스트래글러(또는 스트래글러 워커)로 분류하는 기준으로써, 각 워커별 카운터 값을 갱신하는 기준이 될 수 있다.

[0063] 임계치는 각 에포크의 초반 n 이터레이션 동안의 계산 소요 시간을 기초로 결정된다(또는 산출된다). 이를 위해, 루트 노드는 초반 n 이터레이션에 대한 계산 소요 시간을 워커 노드들 각각으로부터 수신할 수 있다. 이때, n 이터레이션에 대한 계산 소요 시간은 1회에 걸쳐 또는 복수회(예컨대, n 회)에 걸쳐 수신될 수 있다. 실시예에 따르면, 계산 소요 시간은 각 워커 노드로부터 수신된 시간 정보(예컨대, 타임 스탬프)인 현재 이터레이션의 시작 시간(또는 이전 이터레이션의 종료 시간)과 현재 이터레이션의 종료 시간을 기초로 연산될 수도 있다. 이하의 각 단계에서도 동일하게 해석될 수 있다.

[0064] 임계치의 도출을 위해, 각 이터레이션 동안의 최단 기록들에 대한 평균을 산출할 수 있다. 산출된 평균에 미리 정해진 상수(k)를 곱함으로써 임계치가 도출된다. 도출된 임계치는 워커들 각각으로 전송될 수 있다.

[0065] S110 단계가, 최초의 에포크가 아닌 에포크, 즉 제2 에포크나 그 이후의 에포크에서 수행되는 경우, 스트래글러로 분류된 워커 노드도 학습에 참여할 수 있다. 또한, S110 단계 동안 워커 노드들 각각에 대한 카운터 값이 갱

신되는 것도 가능하다. 이때 카운터 값의 갱신의 기준인 임계치는 이전 에포크에서 결정된 임계치가 이용될 수 있다. 실시예에 따르면, 카운터 값의 갱신은 스트래글러 워커만을 대상으로 수행되는 것도 가능하다. 이를 통해, 스트래글러로 분류된 워커의 카운터 값이 기준치 이하로 감소될 수 있고, 이러한 경우에 스트래글러로 분류된 워커는 다시 학습 과정에 참여할 수 있다.

[0066] 초기 n회의 이터레이션이 경과한 후, 워커별 계산 소요 시간이 수신된다(S120). 계산 소요 시간의 수신은 소정의 유무선 통신 프로토콜이나 MPI 프로토콜에 의해 수행될 수 있다.

[0067] 워커별 카운터 값이 갱신된다(S130). 이때, 루트 노드는 카운터 값을 각 워커별로 구분하여 관리할 수 있다. 카운터 값은 워커의 계산 소요 시간과 임계치의 비교를 통해 갱신될 수 있다. 예컨대, 계산 소요 시간이 임계치보다 크다면(또는 크거나 같다면) 해당 워커 노드의 카운터 값이 1 증가될 수 있다. 또한, 계산 소요 시간이 임계치보다 작다면(또는 작거나 같다면) 해당 워커 노드의 카운터 값이 1 감소될 수도 있다.

[0068] 워커 분류 동작이 수행된다(S140). 즉, 기준 카운터 값과 동일한 카운터 값을 갖는 워커 노드가 속도 저하를 유발하는 워커 노드, 즉 스트래글러로 분류(또는 판단)될 수 있다. 스트래글러로 분류된 워커 노드는 이후의 학습 과정에서 배제될 수 있다.

[0069] 상술한 S120 내지 S140 단계는 나머지 이터레이션 동안 반복적으로 수행될 수 있다. 다시 말하면, S120 내지 S140 단계는 초기 임계치 도출 과정 이후에 매 이터레이션 마다 수행되는 것으로 이해될 수 있다. 실시예에 따르면, 반복적인 동작은 스트래글러로 분류된 워커 노드가 학습 과정에서 제외된 상태에서 수행될 수도 있다.

[0070] 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, 컨트롤러, 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 컨트롤러를 포함할 수 있다. 또한, 병렬 프로세서(Parallel Processor)와 같은, 다른 처리 구성(Processing Configuration)도 가능하다.

[0071] 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.

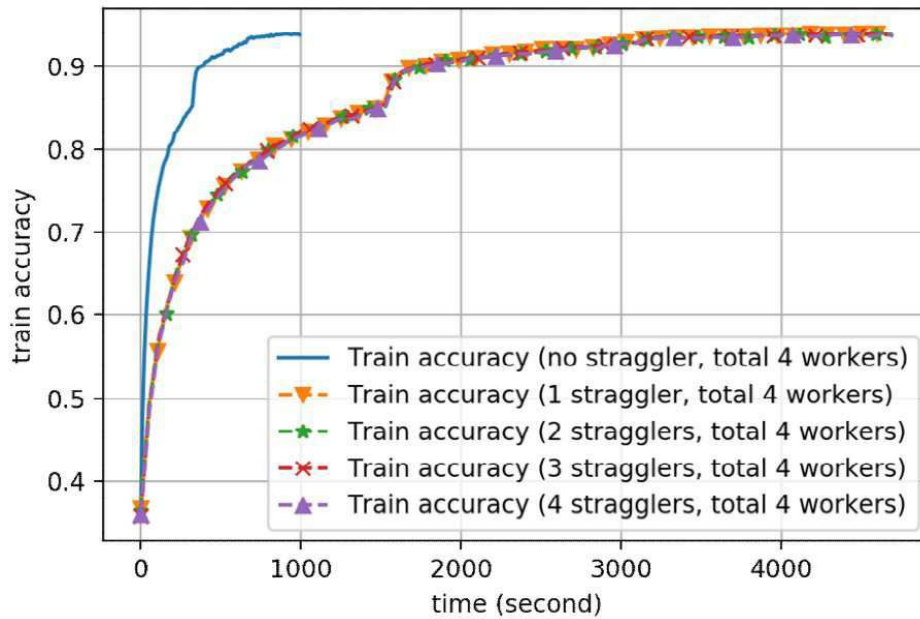
[0072] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(Magnetic 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.

[0073]

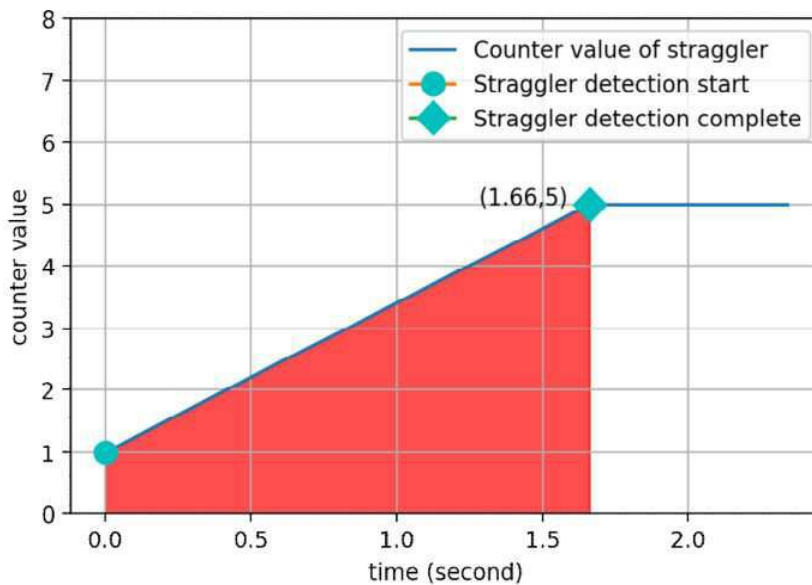
본 발명은 도면에 도시된 실시예를 참고로 설명되었으나 이는 예시적인 것에 불과하며, 본 기술 분야의 통상의 지식을 가진 자라면 이로부터 다양한 변형 및 균등한 타 실시예가 가능하다는 점을 이해할 것이다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성 요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성 요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다. 따라서, 본 발명의 진정한 기술적 보호 범위는 첨부된 등록청구범위의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

도면

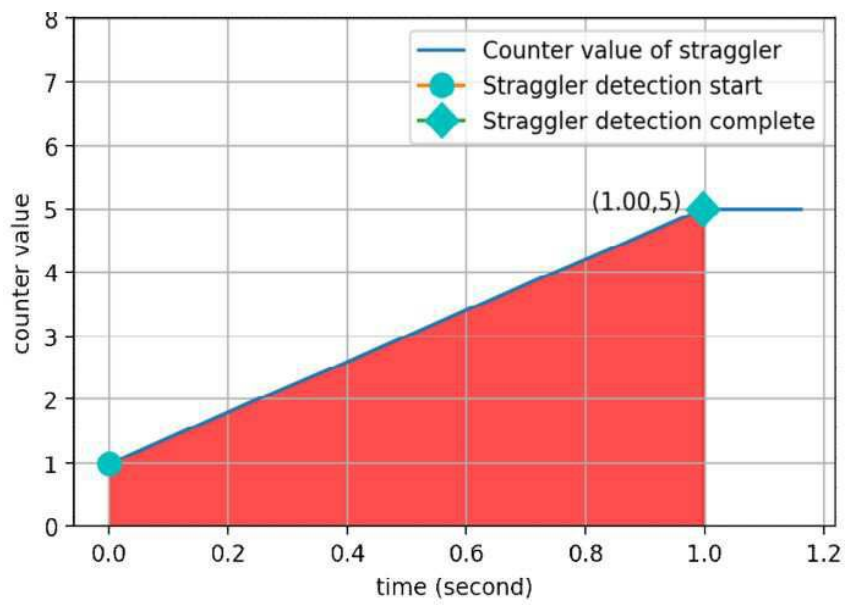
도면1



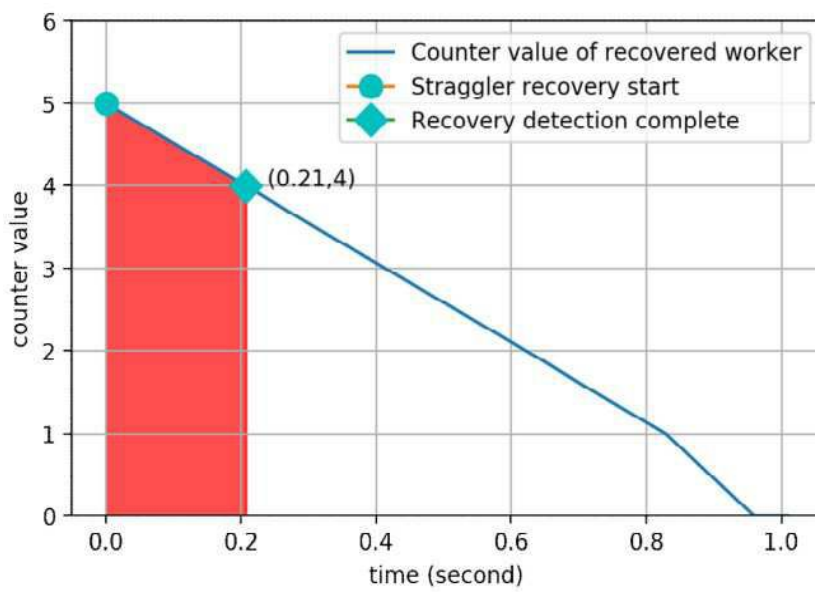
도면2



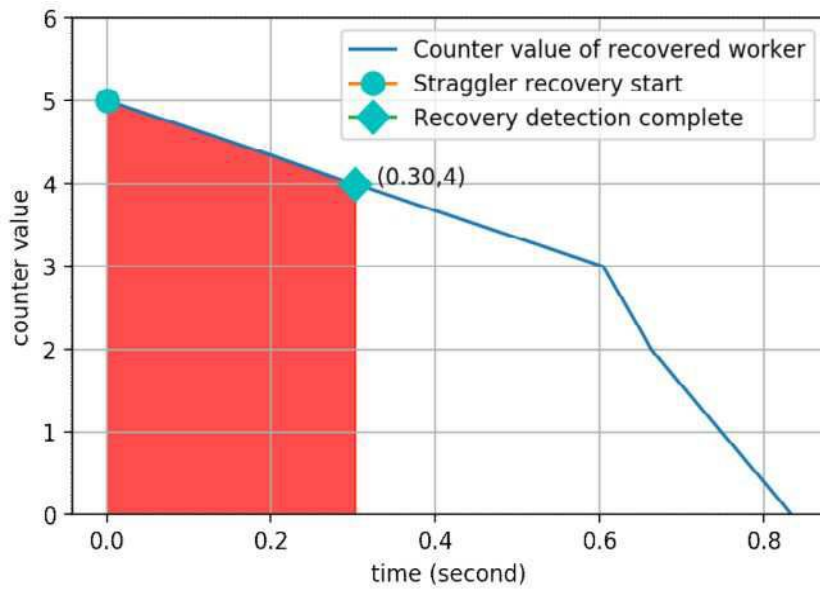
도면3



도면4



도면5



도면6

