Journal of KIIT. Vol. 21, No. 10, pp. 1-9, Oct. 31, 2023. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2023.21.10.1

분산 딥러닝 환경에서 Straggler 문제해결을 위한 동적 워커 분류 기법

김형준*[1], 유헌창*[2], 김성석**

Dynamic Worker Classification Scheme for Addressing Straggler Problem in Distributed Deep Learning Environments

HyungJun Kim*[1], Heonchang Yu*[2], and Sungsuk Kim**

※ 이 논문은 정부(과학기술정보통신부, 교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2019M3E7A1113102)

요 약

Straggler로 인한 속도 저하는 작업 시간 단축을 주요 목적으로 하는 분산 딥러닝에 치명적인 영향을 미친다. 이를 Straggler 문제라고 한다. Straggler는 발생 시점과 속도 저하의 정도 모두 임의적이므로 심각한 경우 분산 딥러닝의 도입 자체를 무의미하게 만들 수 있다. 본 논문에서는 워커 속도 저하에 강건한 분산 딥러닝 학습을 위해 필수적인 동적 Straggler 탐지 기법을 제안한다. 제안 기법은 워커 성능 프로파일링 정보를 통해 결정되는 임계치를 기반으로 작동한다. 실험 결과를 통해 오분류 없이 신속하게 Straggler를 탐지함을 보인다.

Abstract

The detrimental impact of stragglers on distributed deep learning, which aims to optimize computational efficiency, is well-known. The unpredictable occurrence and varying severity of stragglers give rise to what is known as the straggler problem. In severe cases, the very essence of distributed deep learning becomes futile due to these issues. This paper introduces a dynamic technique for detecting stragglers, a crucial aspect for ensuring the resilience of distributed deep learning against worker slowdowns. The proposed method utilizes a threshold determined through worker performance profiling information. Experimental results validate the efficacy of the proposed method in swiftly identifying stragglers without any misclassification.

Keywords
distributed deep learning, straggler problem, straggler detection, worker classification

* 고려대학교 컴퓨터학과
- ORCID[1]: https://orcid.org/0009-0003-5078-4405
- ORCID[2]: https://orcid.org/0000-0003-2216-595X
** 서경대학교 소프트웨어학과 교수(교신저자)
- ORCID: http://orcid.org/0000-0002-7596-0757

· Received: Jul. 03, 2023, Revised: Aug. 28, 2023, Accepted: Aug. 31, 2023
· Corresponding Author: Sungsuk Kim
Dept. of Seokyeong University Software Engineering
Tel.: +82-2-940-7756, Email: sskim03@skuniv.ac.kr

Ⅰ. 서 론

딥러닝은 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 기술이 필요하다.

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 단축시킨다. 데이터 병렬화는 동종(Homogeneous) 환경에서 빠른 처리 속도를 보인다. 하지만 Straggler 발생 시 전

체 작업 속도가 가장 느린 워커를 기준으로 맞춰지게 된다[1]. 이를 Straggler 문제라고 하며, 각 워커의 계산 결과를 합쳐서 매개변수 업데이트를 진행하는 동기화의 특성에 의해 발생한다[2]. 심각한 속도 저하를 보이는 워커는 분산 딥러닝의 도입 자체를 무의미하게 만든다는 점에서 Straggler 문제는 분산 딥러닝 분야에서 해결해야 할 주요 과제로 여겨진다.

Straggler 문제해결의 핵심은 정상 속도인 워커와 Straggler의 동기화 작업을 최적화된 주기로 수행하는 것이다. 이를 위해서는 어떤 워커가 Straggler인지 분류해내는 작업이 필수적이다. 하지만 다음의 두 가지 이유로 인해 적절한 Straggler 탐지 방법을 제시하는 데 어려움이 따른다. 첫째로, 워커가 보이는 속도 저하의 정도와 발생 시점은 임의적이다. 둘째로, 정상 워커의 처리 속도 범위가 시스템 상황에 따라 변화하므로 정적인 값을 통해 Straggler 여부를 판단할 수 없다. 본 논문에서는 워커 성능 프로파일링 기반의 임계치를 이용한 동적 Straggler 탐지 기법을 제안하여 워커의 속도 저하에 강건한 분산 딥러닝 학습이 이루어질 수 있게 한다. 이를 통해 학습이 진행되는 시점의 시스템 상황을 기준으로 한 임계치를 적응적으로 찾고 Straggler를 신속하게 탐지한다.

논문의 구성은 다음과 같다. 2장에서는 분산 딥러닝에서 Straggler 문제를 해결하기 위해 제시된 기존 연구의 내용과 한계를 분석한다. 3장에서는 연구 동기를 밝히고, 4장에서는 제안하는 Straggler 탐지 기법 및 구현에 대해 설명한다. 이어서 5장에서는 제안 기법의 Straggler 탐지 성능을 검증하는 실험 및 평가가 이루어지고, 6장에서 결론을 제시하며 논문을 마무리한다.

Ⅱ. 관련 연구

앞서 설명한 것과 같이 Straggler 문제는 분산 딥러닝 분야의 대표적인 문제이다. 특히, Straggler는 임의의 시점에 불특정한 워커에서 발생하기 때문에 Straggler의 존재 여부를 미리 파악하는 것은 불가능하다. 분산 딥러닝 분야에서는 Straggler 문제로 인한 성능을 저하를 완화하기 위해 다양한 기법들이 제안되었다.

2.1 비동기 탈중앙 기법

비동기 방식의 분산 딥러닝은 전체 워커의 일부를 서브그룹으로 구성하여 동기화를 수행하기 때문에 서브그룹에 Straggler가 포함되지 않는 경우 Straggler의 영향을 받지 않는다. 또한, 동기식 방법에 비해 Straggler로 인한 영향을 상대적으로 적게 받는다. 하지만 기울기 계산 시작 시점과 가중치 업데이트 시점의 모델이 달라지는 경우 수학적으로 부정확한 값이 사용되는 문제가 발생하며, 이는 낮은 수렴율로 이어진다.

[1]은 워커가 그래디언트 계산을 완료하면 다른 워커 하나를 임의로 선택해 모델 평균화 작업을 수행하는 AD-PSGD 알고리즘을 제안하였다. AD-PSGD는 SGD와 같이 $O(1/\sqrt{k})$로 수렴하며, 올-리듀스 방식에 비해 Straggler에 대해 강건함을 실험을 통해 증명하였지만 동종 환경에서 올-리듀스 방식에 비해 3배 이상 느리다는 문제가 있다[2]. 분산 학습 기법인 Prague는 AD-PSGD를 동기화 그룹 크기가 2인 특수한 케이스로 보고 일반화하였으며, [3] 또한 유사한 접근법을 취하였다. Prague는 그룹 생성 시 워커 중복을 피하는 스케줄링 기법을 통해 성능 개선하였으며, 동종 환경에서도 올-리듀스에 준하는 성능을 달성한다. 그러나 실험에서 16개의 GPU 중 1개 워커의 5배 속도 저하만으로 60% 이상의 성능 저하가 발생하며, Straggler 문제에

여전히 취약함을 보였다. [4]에서 제안한 기법은 AD-PSGD와 마찬가지로 가십(Gossip) 기반 방법에 속하며, 그래디언트 압축을 통해 통신 비용을 감소시킨다. 하지만 Straggler 문제의 다양한 발생 원인 중 일부만을 다룰 수 있다는 한계를 갖는다.

2.2 그래디언트 코딩 기법

그래디언트 코딩은 중복 계산을 통해 Straggler의 계산 완료 시점까지 대기하지 않고도 정확한 기울기 값을 구할 수 있도록 하는 방법이다[5]. 이 방법은 Straggler의 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 Straggler 개수의 상한이 존재한다는 한계를 갖는다[6]. 이러한 문제들을 해결하기 위해 기존 연구에서는 다음과 같은 근사 그래디언트 코딩 방법이 제시되었다.

[7]은 재귀 다항식 구성에 기반한 코딩으로 전체 작업 시간을 단축하였으나 정해진 개수 이상의 Straggler를 감당할 수 없다는 문제가 존재한다. CodedReduce는 통신을 위해 마스터 노드와 L개 레이어의 워커들로 구성된 논리적 트리 토폴로지를 활용하여 링 올-리듀스의 효율적인 통신과 Straggler에 대한 강건함을 동시에 달성한다[8]. 하지만 CodedReduce가 최적의 성능을 발휘하기 위해 학습 시작 전에 확보해야 하는 Straggler의 개수 정보를 미리 파악하는 것은 불가능하다. 또한 그래디언트 코딩 및 그 근사 방법론들 모두 중복 계산에 의존하기 때문에 Straggler가 존재하지 않는 동종 환경에서의 작업 속도는 링 올-리듀스보다 느리다는 한계를 갖는다.

2.3 부분적 올-리듀스 기법

부분적 올-리듀스 기법은 Straggler의 동기화 참여가 전체 작업 지연의 원인이라는 점에 주목하여 Straggler 탐지 후 해당 워커를 다른 워커들과의 동기화 과정에서 일시적 또는 영구적으로 배제시키는 방식이다[9][10]. 이러한 방식은 Straggler가 존재할 때 일반적인 동기식 방법론에 비해 총 작업 시간을 크게 단축시킨다는 장점이 있으나, Straggler를 동기화 작업에서 배제하므로 할당된 데이터를 학습에 사용하지 못할 수 있으며 이는 수렴률 저하의 요인이 된다.

DPAR의 경우, 임계치 및 카운터를 활용한 Straggler 탐지를 수행한 뒤 Straggler를 제외한 상태에서 링 올-리듀스를 통해 모델을 업데이트한다[10]. 본 논문의 제안 기법 또한 임계치 및 카운터를 활용하며 DPAR의 Straggler 탐지 방법과 유사한 접근법을 취하고 있다. 하지만 DPAR은 사용자로부터 정적 임계치 값을 입력받아 사용하며, 적절한 임계치를 구하기 위해서는 시행착오가 불가피하게 요구된다는 점에서 사용자에게 부담을 전가하는 방식이다. 한편 제안 기법은 워커들 간의 통신을 통해 임계치를 동적으로 찾으며, 에포크마다 이 값을 시스템 상황에 맞게 자율적으로 갱신하는 방식으로, 사용자 측면의 개입을 필요로 하지 않는다.

본 논문의 제안 기법은 사용자의 개입 없이 모델 학습 도중 무작위적으로 발생하는 Straggler를 신속하게 탐지해냄으로써 Straggler에 적응적으로 대응하는 방식의 학습이 이루어질 수 있도록 한다. 또한 정상 워커와 Straggler의 동기화를 피하기 위해 확률에 의존하지 않고 중복 계산을 요구하지도 않는다는 점에서 위의 방법들에 비해 상대적으로 이점을 갖는다. 워커의 상태 정보를 이용하여 정상 워커와 Straggler와의 동기화 주기를 제어할 수 있기 때문이다. 제안 기법의 자세한 동작에 대해서는 3장에서 설명한다.

III. 연구 동기

이 장에서는 그림 1의 실험을 통해 Straggler 문제가 분산 학습에 미치는 영향의 심각성을 다룬다. 실험은 텐센트 클라우드 환경에서 수행하였으며 GN7.20XLARGE320 인스턴스 1개를 사용하였다. GPU는 NVIDIA T4를 총 4개 사용하였다. 분산 딥러닝 학습 프레임워크인 Horovod[11]로 ResNet-18[12] 모델을 총 90 에포크 동안 학습시키는 작업을 수행하였다. 데이터셋은 CIFAR-10을 사용하였다.

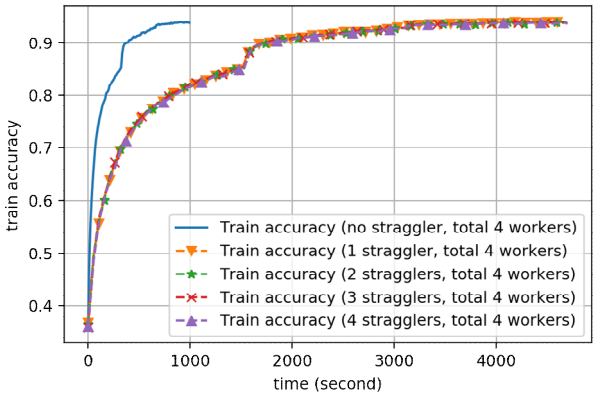


그림 1. Straggler 개수별 학습 정확도 추이
Fig. 1. Train accuracy trend by number of stragglers

Straggler는 Python의 time 라이브러리에서 sleep 함수를 사용하여 구현하였다. 속도 저하의 정도는 정상 워커의 5배 수준으로 설정하였다.

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

이 결과는 Straggler가 단 1개만 존재하더라도 전체 작업 시간이 큰 폭으로 증가함을 보여준다. 이처럼 Straggler가 1개만 존재하더라도 모든 워커가 Straggler인 경우와 같은 수준의 심각한 속도 저하를 초래하므로 모든 Straggler를 신속하고 정확하게 탐지 및 처리하는 기법이 반드시 필요함을 알 수 있다.

Ⅳ. 동적 워커 분류 기법

4.1 Straggler 탐지

워커 속도 저하에 강건한 분산 딥러닝을 위해서는 어떤 워커가 Straggler인지 탐지해내는 작업이 필수적으로 이루어져야 한다. 하지만 예측 기반 방법론은 Straggler 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 Straggler의 정확한 탐지를 더 어렵게 만든다.

본 논문은 가변적인 시스템 상황에 맞추어 신속하게 대응할 수 있는 Straggler 탐지 기법을 제안한다. 제안 기법을 통한 Straggler 탐지는 두 단계로 진행된다. 첫 번째 단계에서는 임계치를 정한다. 임계치 설정의 구체적인 과정은 다음과 같다. 먼저, 워커의 성능을 프로파일링하는 작업이 수행된다. 성능 지표는 Python의 time 함수를 이용해서 측정한 계산 시간이며, 측정은 이터레이션마다 이루어진다.

Algorithm 1: Dynamic straggler detection with counter

Input:: A set of workers $W=\{w_i|1 \le i \le n\}$
for worker $w_i \in W$ do
  counter $c_i = 0$
end for
while not reached convergence do
  if $c_i >$ limit and w_i is a straggler then
    continue
  end if
  Record local time t_i
  Compute gradients over selected samples
  Average gradients computed by $w_i \in W$
  Record local time t_j
  if $t_j - t_i >$ threshold then
    c_i+=1
  else if $c_i > 0$ and $t_j - t_i <$ threshold then
    c_i-=1
  end if
  if c_i==limit then
    classify w_i as a straggler
  end if
end while

여기서 가장 빠른 워커의 기록은 현재 시스템 환경에서 정상 워커의 성능을 나타낸다. 본 논문에서는 한 에포크 동안의 정상 워커 성능을 파악하기 위해 에포크 초반 n 이터레이션 동안의 최단 기록들을 모아 그 평균값을 구한다. 임계치는 이 값에 상수 k를 곱하여 정한다.

n, k 값은 실험에서 일반적으로 좋은 성능을 보인 n=5, k=2를 기본값으로 한다. 하지만 다음의 트레이드오프 관계를 고려하여 사용자가 요구사항에 맞게 설정할 수도 있다. n값이 클수록 Straggler 탐지까지 소요되는 시간이 증가하지만, 간헐적 속도 저하를 보이는 정상 워커를 Straggler로 오분류하는 경우를 줄일 수 있다. k값의 경우, Straggler 분류 기준이 되는 속도 저하의 정도를 나타낸다. 따라서 k값을 크게 설정한다면 높은 신뢰도의 탐지 결과를 얻지만, 그보다 경미한 수준의 워커 속도 저하가 전체 작업 시간을 증가시키는 것을 허용한다.

두 번째 단계에서는 카운터 기반 방식으로 Straggler를 탐지한다. 알고리즘 1에 이 탐지 과정이 의사코드로 제시되어 있다. 카운터는 워커마다 따로 관리되며 이터레이션마다 측정되는 워커의 계산 시간이 임계치를 초과할 때마다 워커 카운터의 값을 1만큼 증가시킨다. 이는 알고리즘 1의 8~16번째 줄의 내용에 해당한다. 만약 워커의 카운터 값이 일정 값에 도달하면 알고리즘 1의 17~18번째 줄에서와 같이 해당 워커를 Straggler로 분류한다. Straggler 분류 기준이 되는 카운터 값은 사용자가 직접 정하거나 에포크를 이루는 전체 이터레이션의 일정 비율로 정한다. 본 논문의 제안 기법은 이와 같이 사용자의 개입을 필요로 하지 않는 자율적인 방식으로 Straggler를 탐지해낸다.

한 가지 특기할 사항은 분산 딥러닝에서 워커들의 작업은 각각 별도의 프로세스라는 점이다. 이는 가장 빠른 워커 계산 시간을 구하기 위해 프로세스 간 통신이 이루어져야 함을 의미한다. 본 논문에서는 MPI(Message Passing Interface) 통신 그룹을 구성하여 워커들이 서로 값을 교환할 수 있는 메커니즘을 구현했다. 임계치 설정에 있어 워커들 간의 MPI 통신은 다음과 같이 이루어진다. 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 Straggler 탐지에 적용된다.

4.2 워커 상태 회복

분산 딥러닝에서는 탐지된 Straggler는 남은 학습 과정으로부터 제외시키는 것이 일반적이다. Straggler로 인한 작업 속도 저하를 최소화하기 위해서이다. 하지만 Straggler로 분류된 워커가 정상 상태로 회복할 수도 있다. 이 경우, 해당 워커를 작업에 다시 복귀시킴으로써 학습 시간을 단축시킬 수 있다.

Straggler로 분류된 워커가 정상 상태로 회복했음을 파악하기 위해서는 해당 워커의 작업 처리 속도를 지속적으로 모니터링해야 한다. 본 논문의 제안 기법에서는 이를 위해 새로운 에포크의 초반 n 이터레이션 동안 Straggler를 포함한 학습을 진행하고 워커마다 계산 소요 시간을 측정한다. 워커의 계산 소요시간이 임계치 이하인 경우, 카운터 값을 1만큼 감소시킨다. 따라서 Straggler로 분류되었던 워커가 정상 속도를 회복한다면 카운터 값을 Straggler 분류 기준값 미만으로 감소시킬 수 있고, 다시 모델 학습 작업에 복귀할 수 있다.

아직 회복하지 못한 Straggler의 동기화 참여는 전체 작업의 속도를 저하시킨다. 하지만, GPU는 고가의 강력한 계산 자원이므로 제한된 횟수의 이터레이션 동안 위험을 감수하고 Straggler로 분류된 워커의 회복 가능성을 확인한다.

4.3 정상 워커 오분류 최소화

본 연구의 제안 기법은 워커가 최근 기록한 속도 저하의 빈도를 기준으로 Straggler를 탐지함으로써 정상 워커가 Straggler로 오분류되는 경우를 최소화하고자 한다. 이를 구현하기 위한 핵심 아이디어는 총 두 가지이다.

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 Straggler로 분류하는 1종 오류를 범할 위험이 크다는 점이다. 계산 속도에 영향을 미치는 요소는 다양하므로 정상 워커 또한 간헐적인 속도 저하를 보인다. 따라서 최악의 경우만을 누적 집계한다면 작업의 후반부에 대다수의 워커를 Straggler로 분류하게 될 위험이 있다. 또한 이 위험은 작업의 규모 및 소요 시간에 비례하여 증가한다. 따라서, 본 연구는 워커가 기록한 시간이 임계치 미만인 경우 카운터 값을 차감한다. 이는 워커의 총 속도 저하 횟수가 아닌 최근 기록한 속도 저하의 빈도를 기준으로 Straggler 탐지가 이루어질 수 있도록 한다.

두 번째는 카운터 값의 범위 제한이다. 제안 기법에서 카운터 값은 0 이상 Straggler 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 Straggler의 발생 또는 정상 워커로의 회복 판단이 이루어지기까지 많은 시간이 소요된다. Straggler의 분류 지연은 전체 작업 시간 증가로, 정상 워커의 재합류 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 있도록 한다.

V. 실험 및 평가

5.1 실험 환경

실험은 그림 1에서와 같이 텐센트 클라우드 환경에서 GN7.20XLARGE320 인스턴스 1개를 사용하여 수행하였다. GPU는 NVIDIA T4를 4개 사용하였다.

데이터셋은 CIFAR-10으로 ResNet-18 모델을 총 10 에포크 동안 학습시키는 작업을 수행하였으며, 분산 딥러닝 학습 프레임워크인 Horovod를 사용하였다. Straggler는 Python의 time 라이브러리에서 sleep 함수를 사용하여 구현하였다.

실험에 사용된 매개변수는 다음과 같이 설정하였다. 먼저, Nesterov 모멘텀 [13]을 적용하였으며 $m = 9$를 사용하였다. 가중치 감쇠를 위해서는 $\lambda = 0.00005$를 사용하였다. 학습률을 위해서는 레퍼런스 학습률 [14]을 사용하였고, 웜-업 기법을 학습 초반 5번의 에포크 동안 적용하였다. 속도 저하의 정도는 정상 워커의 3배 수준으로 설정하였다. 임계치 설정을 위해 곱하는 상수값으로는 2를 사용하였으며, 각 에포크마다 초반 5번 이터레이션 동안의 모니터링을 통해 워커 퍼포먼스 프로파일링 작업을 수행하였다.

5.2 실험 결과

실험은 제안하는 워커 분류 기법이 Straggler의 발생과 Straggler의 정상 워커로의 회복을 모두 신속하고 정확하게 탐지하는지 확인하기 위해 다음과 같이 설계되었다. 먼저, 에포크의 전체 이터레이션 횟수 중 절반만큼은 임의의 워커 1개를 Straggler로 지정하여 3배만큼 속도를 저하시킨다. 이때 에포크 시작 시간과 Straggler 탐지가 이뤄진 시간을 측정함으로써 Straggler 탐지까지 소요된 시간을 파악할 수 있다. 다음으로, 에포크의 전체 이터레이션 횟수 중 나머지 절반만큼은 Straggler로 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 Straggler의 정상 상태 회복을 감지하는 데 소요된 시간을 알 수 있다. 마지막으로, 지정된 Straggler가 에포크의 전반부에서 학습 도중 정상 상태를 회복한 것으로 분류되는 경우, 그리고 정상 워커가 Straggler가 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.

그림 2와 그림 3은 에포크 1에서 임계치가 설정된 시점부터 Straggler 탐지가 이뤄진 시점까지 Straggler로 지정된 워커의 카운터 값의 변화 추이를 나타낸다.

붉은 색으로 표시된 구간은 Straggler 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 Straggler 탐지를 완료하였다. 전체 학습이 10 에포크 동안 진행되는 과정에서 Straggler 탐지 동안 10번 이루어졌으며, 임계치 설정 시점부터 Straggler 탐지까지 소요된 시간은 배치 사이즈 32, 64인 경우 각각 평균 1.43초, 1.44초를 기록하였다.

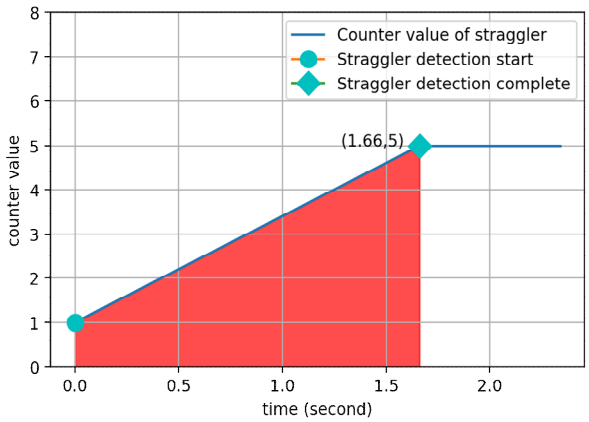


그림 2. 워커별 배치 사이즈 32 실험에서 임계치 설정 이후 Straggler의 카운터 값 변화 추이
Fig. 2. Counter value trend of straggler after threshold setting in 32 batch size per worker experiment

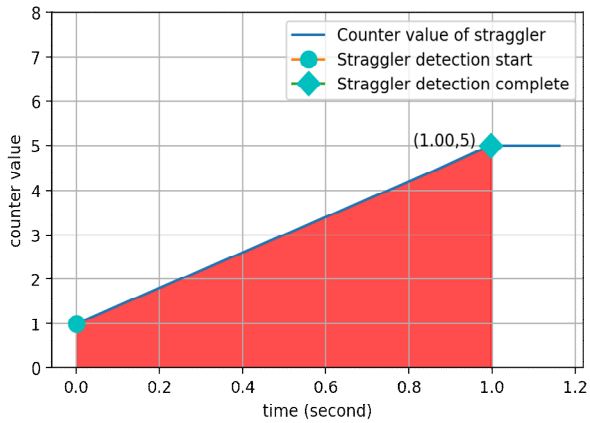


그림 3. 워커별 배치 사이즈 64 실험에서 임계치 설정 이후 Straggler의 카운터 값 변화 추이
Fig. 3. Counter value trend of straggler after threshold setting in 64 batch size per worker experiment

그림 4와 그림 5는 에포크 1에서 중반 이후 Straggler의 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. 그림 4와 그림 5에서 확인할 수 있듯이 배치 사이즈 32, 64인 경우 각각 0.21초, 0.30초만에 Straggler로 분류되었던 워커가 정상 속도를 회복하였음을 감지하였다.

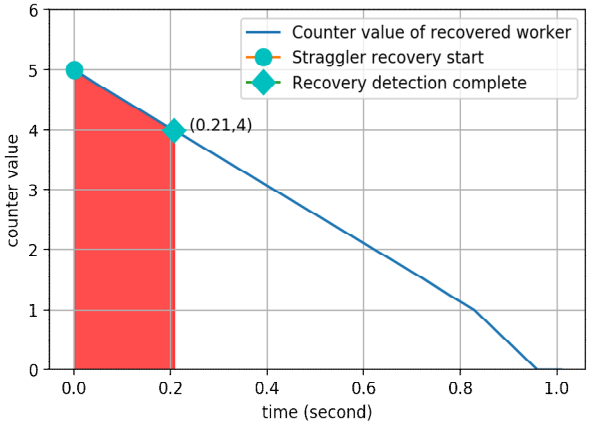


그림 4. 워커별 배치 사이즈 32 실험에서 속도 저하 해제 이후 정상 상태 회복 워커의 카운터 값 변화 추이
Fig. 4. Counter value trend of recovered worker after slowdown unset in 32 batch size per worker experiment

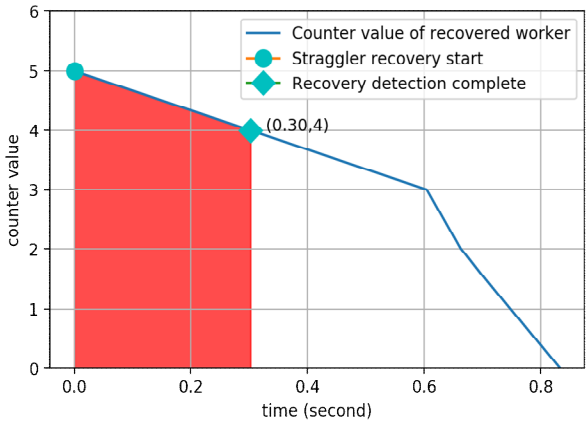


그림 5. 워커별 배치 사이즈 64 실험에서 속도 저하 해제 이후 정상 상태 회복 워커의 카운터 값 변화 추이
Fig. 5. Counter value trend of recovered worker after slowdown unset in 64 batch size per worker experiment

10 에포크 동안 Straggler 탐지에 실패하거나 정상 워커가 Straggler로 오분류된 경우, 그리고

Straggler로 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.

Ⅵ. 결론 및 향후 과제

본 논문은 동적으로 임계치를 설정하고 카운터 기반 방식으로 Straggler를 탐지하고 워커 회복 여부를 결정하는 워커 분류 기법을 제안하였다. 이는 분산 딥러닝의 주요 과제인 Straggler 문제 해결을 달성하는 데 필수적인 Straggler 탐지 작업을 신속하고 정확하게 수행할 수 있도록 한다. 제안 기법의 성능에 대한 검증은 5장의 실험 및 평가를 통해 이루어졌으며, 오분류 없이 Straggler 탐지 및 정상 속도 회복 감지 모두 높은 정확도로 수행함을 보였다.

향후 과제로는 두 가지가 있다. 첫째로, 본 논문의 제안된 기법에서 아직 수동으로 설정해야 하는 매개변수들의 설정을 자동화하는 과제이다. 임계치 설정 작업을 동적으로 수행하는 부분은 본 논문의 기여점이다. 그러나 임계치 설정 과정에서 사용되는 상수 k의 값이나 에포크 초반에 모니터링을 수행하는 이터레이션 횟수 등의 값은 수동으로 설정해야 하는 부분으로 남아 있다. 이러한 매개변수들의 최적값을 자동으로 찾을 수 있는 기법들을 연구함으로써 Straggler 탐지에 소요되는 시간을 추가적으로 단축하고 사용자가 이러한 값들을 찾는 데 투입하는 시간적 비용적 부담을 줄일 수 있을 것이다. 둘째로, 본 논문의 제안 기법과 같은 워커 분류 기법을 통합한 분산 딥러닝 학습 기법을 고안하는 과제이다. 2장에서 지적한 바와 같이, 기존의 비동기 탈중앙 기법 및 그래디언트 코딩 기법은 Straggler 탐지 및 워커 상태 회복 감지 메커니즘을 결여하고 있다. 제안 기법을 통해 Straggler 탐지 및 워커의 상태 회복을 감지할 수는 있지만, 실제 분산 딥러닝 분야에서 Straggler 문제를 해결하기 위해서는 이러한 워커 분류 기법을 통합시킨 분산 딥러닝 학습 기법이 제시되어야 한다. 향후 추가적 연구를 통해 이러한 과제들을 해결할 수 있기를 기대한다.

References

[1] 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.

[2] Q. Luo, J. He, Y. Zhuo, and X. Qian, "Prague: High-performance heterogeneity-aware asynchronous decentralized training", Proc. of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, New York, United States, pp. 401-416, Mar. 2020. https://doi.org/10.1145/3373376.3378499.

[3] 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. https://doi.org/10.1145/3448016.3452773.

[4] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication", International Conference on Machine Learning, PMLR, alifornia, USA, Vol. 97, pp. 3478-3487, Jun. 2019.

[5] 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.

[6] R. Bitar, M. Wootters, and S. E. Rouayheb, "Stochastic gradient coding for straggler mitigation in distributed learning", IEEE Journal on Selected Areas in Information Theory, Vol. 1, No. 1, pp. 277-291, May 2020. https://doi.org/10.1109/JSAIT.2020.2991361.

[7] M. Ye and E. Abbe, "Communication-computation efficient gradient coding", International Conference on Machine Learning, PMLR, Stockholm, Sweden,

Vol. 80, pp. 5610-5619, Jul. 2018.

[8] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S, Avestimehr, "Coded computation over heterogeneous clusters", IEEE Transactions on Information Theory, Vol. 65, No. 7, pp. 4227-4242, Jul. 2019. https://doi.org/10.1109/TIT.2019.2904055.

[9] 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.

[10] H. J. Kim, C. Song, H. M. Lee, and H. Yu, "Addressing Straggler Problem Through Dynamic Partial All-Reduce for Distributed Deep Learning in Heterogeneous GPU Clusters", 2023 IEEE International Conference on Consumer Electronics, Las Vegas, NV, USA, Jan. 2023. https://doi.org/10.1109/ICCE56470.2023.10043527.

[11] 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.

[12] 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.

[13] Y. Nesterov, "Introductory lectures on convex optimization: A basic course", Springer Science & Business Media, Vol. 87, 2003.

[14] 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.

.

저자소개

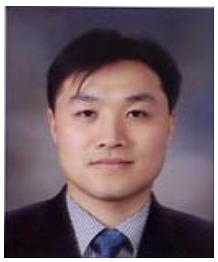
김 형 준 (HyungJun Kim)

2020년 8월 : 고려대학교 철학과(문학사)
2023년 8월 : 고려대학교 컴퓨터학과(공학석사)
2023년 9월 ~ 현재 : 고려대학교 컴퓨터학과(박사과정)
관심분야 : 분산 딥러닝, 분산시스템, 오토스케일링, GPU 클러스터 자원 관리

유 헌 창 (Heonchang Yu)

1989년 2월 고려대학교 컴퓨터학과(학사)
1991년 2월 : 고려대학교 컴퓨터학과(석사)
1994년 2월 : 고려대학교 컴퓨터학과(박사)
1998년 ~ 현재 : 고려대학교 컴퓨터학과 교수
관심분야 : 클라우드컴퓨팅, 가상화, 분산시스템

김 성 석 (Sungsuk Kim)



1997년 2월 : 고려대학교 컴퓨터과학과(이학사)
1999년 2월 : 고려대학교 컴퓨터과학과(이학석사)
2003년 2월 : 고려대학교 컴퓨터과학과(이학박사)
2003년 3월 ~ 현재 : 서경대학교 컴퓨터과학과 교수
관심분야 : 분산시스템, 빅데이터, 인공지능