이기종 GPU 클러스터에서 하이퍼 파라미터 최적화 가속을 위한 컨테이너 기반의 선점형 스케줄링 프레임워크 연구

유용혁, 박성용, 김영재
서강대학교 컴퓨터공학과
{yonghyuk, parksy, youkim}@sogang.ac.kr

Container-based Preemptive Scheduling Framework for Hyperparameter Optimization Acceleration in Heterogeneous GPU Clusters

Yonghyuk Yoo, Sungyong Park, Youngjae Kim
Dept. of Computer Science and Engineering, Sogang University

요 약

딥러닝의 인기가 날로 증가함에 따라, GPU 성능도 빠르게 발전하고 있다. GPU의 빠른 발전으로 자연스럽게 클러스터에는 다양한 GPU 종류가 존재하게 되었다. 선행 연구 Hermes는 클러스터 환경에서 딥러닝 작업의 수렴 속도를 우선으로 하는 스케줄링을 통해 하이퍼 파라미터 과정을 가속하였다. 하지만, GPU 이기종 환경을 고려하지 않아 하이퍼 파라미터 최적화 과정의 효율성이 떨어지는 문제가 있다. 본 논문은 위의 문제를 해결하기 위해 GPU 이기종 클러스터 환경에서의 하이퍼 파라미터 최적화 과정 가속을 위해 GPU 성능 차이를 고려한 컨테이너 기반의 선점형 스케줄링 프레임워크 *Hermes-v2*를 제안한다. *Hermes-v2*는 모니터링을 기반으로 딥러닝 모델이 GPU에 따른 성능 차이를 측정할 수 있다. *Hermes-v2*는 측정 결과를 바탕으로 성능 차이가 가장 큰 작업을 재배치하여 하이퍼 파라미터 최적화 과정을 가속한다. 하이퍼 파라미터 최적화 과정 가속을 평가하기 위해 우리는 Kubernetes 환경에서 개발한 *Hermes-v2*를 FIFO와 RR 스케줄링 기법 및 Hermes와 비교하여 실험을 진행했다. 실험 결과, *Hermes-v2*는 하이퍼 파라미터 최적화 과정을 선행 연구 Hermes와 비교해 최대 30% 단축했으며, 작업 종료 시간도 최대 32% 단축했다.

1. 서 론

딥러닝은 컴퓨터 비전, 자연어처리와 같은 다양한 분야에서 널리 사용되고 있다. 딥러닝 모델의 정확도는 하이퍼 파라미터라고 부르는 변수에 크게 영향을 받는다. 사용자는 다양한 하이퍼 파라미터 조합을 테스트하여 가장 높은 정확도를 갖는 파라미터 조합을 찾는다. 이때, 사용자는 각 하이퍼 파라미터 조합들의 학습이 얼마나 수렴되었는지 또는 정확도를 피드백 받아서 최적의 정도를 판단한다. 이러한 과정을 하이퍼 파라미터 최적화 과정이라고 부른다.

선행 연구 *Hermes* [1]는 단일 노드에 동종의 복수의 GPU가 있는 환경에서 하이퍼 파라미터 최적화 과정을 가속화 하는 연구로 딥러닝 작업의 수렴 속도를 우선으로 하는 GPU 선점형 스케줄링 기법과 GPU간 작업 균등을 위한 GPU 작업 재배치 기법을 제안했다. *Hermes*는 수렴 속도를 우선으로 하는 스케줄링 정책을 사용하여 최적의 하이퍼 파라미터 조합을 찾는 과정을 단축했다. 또한, 단일 노드에서 동종의 GPU를 사용하더라도 GPU가 연결된 PCIe 레인의 수의 차이 등의 원인으로 인해 학습 속도 차이가 존재해서 특정 GPU에서 작업이 빠르게 종료될 수 있다는 점에 착안하여 노드 내의 GPU 간의 작업 개수가 균등하게 유지하도록 GPU 간의 작업 재분배를 사용하였다.

하지만, 단일 노드의 GPU 간의 작업 균등만을 고려하는 것은 다양한 GPU가 존재하는 이기종 GPU 클라우드 환경에서의 효율성 증가 한계가 있다. 예를 들어, 표 1는 두 개의 딥러닝 모델(MobileNet, ResNet56)에 대해 세 개의 GPU(RTX 2070S, RTX 2080S, TITAN V)의 처리량, 그리고 2070S 대비 각 GPU의 이미지 처리량 비율을 보여준다. MobileNet은 두 GPU에서의 성능 차이가 거의 없지만 ResNet56의 경우 상당한 차이가 나 ResNet56은 TITAN V, MobileNet은 2070S에서 돌리는 것이 효율적이다. 또한, 단일 노드에서의 작업 균등은 클러스터 관점에서의 작업 균등이 이루어지지 않는다. 예를 들어, 위의 예시와 같이 2개의 GPU를 가지고 각 GPU에 5개의 ResNet56 딥러닝 작업들이 실행 중이라고 하자. TITAN V가 약 18% 더 빠르게 처리하여 노드 1의 최소 2개의 작업들이 먼저 종료되어 GPU가 쉬거나 클러스터 관점에서 작업 불균등이 발생한다. 이런 경우 최소 1개의 노드 2의 작업을 노드 1의 GPU에 재배치하여 실행하는 것이 효율적이다.

표 1: 딥러닝 모델 별 GPU 종류에 따른 이미지 처리량(2070S에 대한 정규화된 처리량).

Model	2070S	2080S (%)	TITAN V (%)
MobileNet	1924.8	99.7	102.2
ResNet56	696.2	107.2	117.8

본 연구에서 우리는 Hermes와 다르게 멀티 노드와 다른 GPU 종류가 존재하는 GPU 이기종 환경을 목적으로 한다. 이러한 환경에서는 효율성을 위해 해당 작업의 GPU 종류 간의 성능 차이를 고려하여 클러스터 내 작업을 더 효율적인 GPU에 재배치를 해야 한다. 따라서 우리는 GPU 이기종 클러스터 환경을 고려하고 성능 차이에 따른 작업 재배치를 지원하는 컨테이너 기반의 하이퍼 파라미터 최적화 과정 가속화를 위한 클러스터 메니저 프레임워크, *Hermes-v2*를 제안한다.

우리는 GPU 이기종 환경으로 세 개의 다른 GPU 종류가 존재하는 Kubernetes 환경에서 CNN 모델들과 CIFAR-10 데이터 세트를 TensorFlow를 이용하여 실험하였다. 그 결과 *Hermes2*는 Hermes보다 하이퍼 파라미터 최적화 과정을 최대 30%를 단축했으며, 작업 종료 시간을 32% 단축했다.

2. 관련 연구

AWS, Google Cloud와 같은 클라우드 또는 클러스터 환경에서는 하이퍼 파라미터 최적화 작업들을 주로 Kubernetes 같은 클러스터 매니저 위에 컨테이너 형태로 실행한다. 딥러닝 작업의 효율은 주로 GPU를 연산 가속기를 사용하기 때문에 하이퍼 파라미터 최적화 과정은 클러스터 매니저(Kubernetes)의 GPU 자원 관리 및 스케줄링에 크게 영향을 받는다.

기존 딥러닝을 위한 GPU 클러스터 매니저 연구인 Tiresias [2], Optimus [3]는 작업 종료 시간을 줄이는 것을 목적으로 한다. 작업 종료 시간의 단축의 경우 딥러닝 학습의 수렴 정도와 피드백 속도를 고려하지 않고 작업의 남은 시간이 가장 짧은 것을 우선하므로 하이퍼 파라미터 최적화 가속에는 적합하지 않다. Themis [4]는 작업 종료 시간의 관점에서 균등과 작업 종료 시간을 줄이는 것을 목적으로 하여 마찬가지로 수렴 정도와 피드백 속도를 고려하지 않

"본 연구는 과학기술정보통신부 및 정보통신기획평가원의 대학ICT연구센터육성지원사업의 연구결과로 수행되었음" (IITP-2020-2016-0-00465)

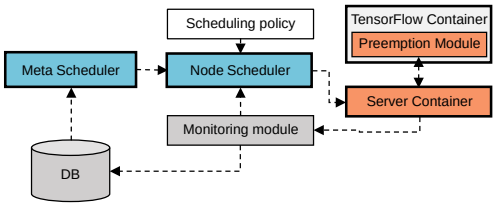


그림 1: Overall architecture

는다. Gandiva [5]는 라운드-로빈 스케줄링과 마이그레이션을 통해 병렬적으로 딥러닝 작업들을 실행하여 딥러닝 작업들의 피드백 시간을 단축했다. 하지만, 라운드 로빈 스케줄링으로는 작업들의 수렴 속도 분포가 다양한 경우 피드백 시간 단축의 한계가 있으며 딥러닝 모델의 GPU 종류에 따른 성능 차이를 고려하지 않아 GPU 이기종 환경에서는 효율성이 떨어진다.

딥러닝을 위한 GPU 이기종 환경을 고려한 연구한 AlloX [6], $\text{Gandiva}_{\text{fair}}$ [7]가 존재한다. AlloX는 CPU, GPU, 다른 가속기를 포함한 이기종 환경을 고려한 연구로 평균 작업 종료 시간을 줄이는 것이 목적으로 하이퍼 파라미터 최적화 가속에는 적합하지 않다. $\text{Gandiva}_{\text{fair}}$는 우리 연구와 마찬가지로 딥러닝 모델의 GPU 종류에 따른 성능 차이를 고려한다. 성능 차이를 기반으로 2차 가격 경매(Second-Price Auction)를 사용하여 스케줄링한다. 하지만, 위의 연구는 max-min fairness가 목적으로 딥러닝 학습의 수렴 정도와 피드백 속도를 고려하지 않고 스케줄링하여 하이퍼 파라미터 최적화 가속에는 적합하지 않다.

우리는 GPU 이기종 환경과 동시에 딥러닝 학습의 수렴 정도를 이용해 최적의 하이퍼 파라미터를 가진 딥러닝 작업을 우선으로 하여 하이퍼 파라미터 최적화 과정을 가속한다.

3. 설계 및 구현

3.1 전반적인 구조

*Hermes-v2*는 그림 1와 같이 파란색 영역의 두 개의 스케줄러(메타 스케줄러와 노드 스케줄러), 주황색 영역의 딥러닝 작업 내부의 서버 컨테이너와 선점 모듈, 회색 영역의 모니터링 모듈과 DB로 구성되어 있다. 노드 스케줄러는 각 물리적 노드에 존재하며, 메타 스케줄러는 클러스터 내 하나의 물리적 노드에서 존재하여 노드 스케줄러와 같은 노드에 존재할 수 있다.

메타 스케줄러는 *Hermes-v2*에서 가장 중요한 스케줄러로 클러스터의 작업 배치를 담당한다. 클러스터의 전반적인 노드의 작업들의 처리량 같은 정보들을 관리해 이를 바탕으로 GPU 종류에 따른 작업 성능 차이를 이용한 효율적인 작업 배치를 한다. 노드 스케줄러는 모니터링 모듈을 통해서 작업들의 정보를 전달받아 Hermes의 정책에 따라 노드 내의 작업들을 스케줄링한다.

딥러닝 작업 내부에는 서버 컨테이너와 선점 모듈가 있다. 서버 컨테이너는 노드 스케줄러로부터 명령을 받아 선점 모듈로 명령을 전달하고 선점 모듈로부터 딥러닝 작업의 정보들을 전달받아 모니터링 모듈에 전달한다. 선점 모듈은 서버 컨테이너로부터 시작/중지 명령어를 받아 GPU 자원을 해제 또는 얻는다. 또한, 모델 정보, 배치 크기, 정확도 같은 딥러닝 작업과 관련된 정보들을 수집하여 서버 컨테이너에 전달한다.

모니터링 모듈은 GPU 사용량을 추적하고 주기적으로 딥러닝 작업 내부의 서버로부터 정보들을 받아 DB에 저장한다. DB는 모든 작업들의 정보를 저장하여 메타 스케줄러의 배치에 사용된다.

3.2 스케줄링 흐름

먼저, 메타 스케줄러에서 배치 정책에 따라 대기 큐 (arriving queue)에 있는 작업 하나와 실행할 노드를 선택해 해당 노드에 작업을 할당한다. 그다음 노드 스케줄러는 매 스케줄링 주기 (현재 30초)마다 스케줄링 정책에 따라 대기 큐에 있는 작업 중 하나를 선택해 실행한다. 현재 스케줄링 정책은 선행 연구 [1]의 수렴 속도를 우선으로 한 정책을 사용한다.

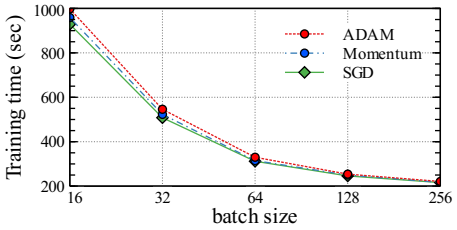


그림 2: 같은 GPU에서의 ResNet56 모델의 각 Optimizer별 배치 사이즈에 따른 학습 시간 변화

재배치 과정은 새로운 작업이 오거나 기존의 작업이 끝난 경우 메타 스케줄러에서 진행한다. 메타 스케줄러는 재배치 정책에 따라 재배치할 작업과 노드를 선택하여 해당 노드 스케줄러에 전달한다.

3.3 GPU 종류에 따른 성능 차이를 이용한 배치 정책

사전 지식 없이 딥러닝 모델의 GPU 종류 간에 따른 성능 차이를 이용하기 위해서는 딥러닝 작업을 실제로 각 GPU에서 실행을 해봐야 한다. 일반적으로 하이퍼 파라미터 최적화 과정시 같은 모델을 사용한다. 그림 2와 같이 동일 GPU 내 같은 모델인 경우 배치 사이즈에 따른 GPU에 따른 성능 차이 경향성이 비슷하므로 우리는 이것을 이용하여 GPU 간의 성능 차이를 설정했다.

먼저, 작업 배치 시 작업 개수 균등이 우선으로 모든 GPU에 작업들이 배치되고 실행된다. 메타 스케줄러는 모니터링 모듈을 통해 각 작업들의 처리량과 같은 지표들을 수집하여 GPU마다 성능을 기록한다. 그 이후의 작업들은 GPU별 성능을 등식

$$\mathrm{P}_G(j) = \mathrm{T}(j_t) \cdot \mathrm{B}(j)/\mathrm{B}(j_t) \tag{1}$$

로 측정을 한다. 여기서 j_t는 가장 먼저 GPU G에 배치된 작업으로 $\mathrm{T}(j_t)$은 작업 j_t의 이미지 처리량으로, j_t를 기준으로 작업 j의 GPU G에서의 성능을 계산한다. 위의 기준으로 GPU G_a에 배치된 j_a와 G_b에 배치된 j_b의 작업의 성능을

$$(P_{G_b}(j_a) + P_{G_a}(j_b)) - (P_{G_a}(j_a) + P_{G_b}(j_b)) \tag{2}$$

로 비교하여 등식 2이 양수인 경우 교환하는 것이 효율적이므로 서로 노드를 교환한다.

재배치 과정은 새로운 작업 또는 종료된 작업이 발생하여 GPU 간의 작업 개수 불균형이 발생한 경우 작업의 개수가 가장 적은 GPU의 작업을 선택한다. 해당 작업과 다른 작업들과 비교하여 등식 2이 양수인 작업이 존재하는 경우 이 중 가장 큰 값을 선택한다.

4. 실험 결과

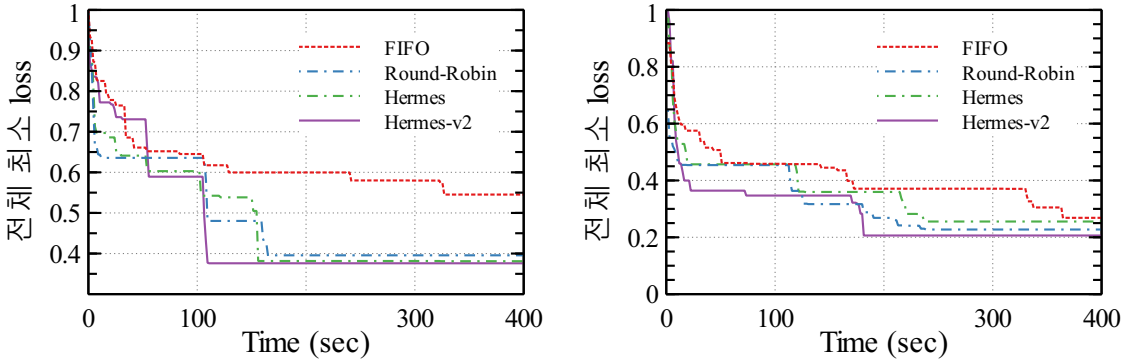
4.1 실험 환경

클러스터 환경	Kubernetes 1.15.x, Docker 18.09
GPU	(TITAN V, RTX 2070S, 2080S) 중 하나
CPU	AMD Ryzen 9 3900X
RAM	64GB
SSD	Samsum 970 EVO

표 2: 실험 노드 환경 설정

우리는 GPU 이기종 환경에서의 하이퍼 파라미터 최적화 가속의 정도를 평가하기 위해 Kubernetes master 노드 하나와 표 2의 서로 다른 하나의 GPU를 가진 3개의 노드가 존재하는 클러스터 환경에서 실험을 진행했다.

딥러닝 작업은 TensorFlow를 사용하여 모델을 구성하였으며 CIFAR-10 데이터 세트를 이용했다. 딥러닝 모델은 GPU 이기종 환경에서의 GPU 종류에 따른 성능 차이가 크지 않은 MobileNet과 성능 차이가 큰 ResNet56 모델을 사용하였다. 하이퍼 파라미터 조합은 epoch은 고정하고, optimizer, 배치 사이즈, learning rate에



(a) MobileNet (b) ResNet56

그림 3: 시간에 따른 전체 최소 loss 값 비교.

변화를 주어 총 12개의 작업을 구성하였다.

우리는 *Hermes-v2*의 하이퍼 파라미터 최적화 가속의 정도를 비교하기 위해 FIFO, 라운드-로빈, 그리고 Hermes와 비교하여 실험을 진행했다. *FIFO*는 Kubernetes의 기본 스케줄러이며 선점 기능은 사용하지 않았고, 라운드-로빈은 Gandiva [5]의 방법으로 GPU 시분할 공유 기법을 사용하여 실험을 진행했다. 선행 연구인 Hermes [1]는 본 논문과 동일하게 딥러닝 작업의 수렴 속도를 고려하는 정책을 사용했다. 그리고 현재 실험 환경은 노드에 하나의 GPU만 존재하여 Hermes는 정적 배치만을 사용한다. 스케줄링 주기는 FIFO를 제외하고 모두 30초로 설정하였다.

4.2 실험 결과

하이퍼 파라미터 최적화 과정 가속 비교: 우리는 *Hermes-v2*의 성능을 평가하기 위해 두 가지 딥러닝 모델에 대해 스케줄러별 시간에 따른 모든 작업들 중 가장 작은 loss를 측정했다. 그림 3는 각 딥러닝 모델의 전체 최소 loss 값을 보여주는 그래프이다. 왼쪽은 학습 시간이 더 짧고 GPU 종류 간 성능 차이가 크지 않은 MobileNet, 오른쪽은 학습 시간이 비교적 길며 성능 차이도 비교적 큰 ResNet56 모델의 실험 결과이다. MobileNet의 경우 Hermes의 102초보다 약 30초가량 빠르게 도달하여 약 30% 정도 시간을 단축했으며 다른 스케줄러 방법보다도 모두 더 빠르게 도달하였다. ResNet56의 경우 Hermes보다 50초가량 빠르게 도달하여 약 30% 정도의 시간을 단축했고 마찬가지로 다른 스케줄러보다 더 빠르게 도달한 것을 확인할 수 있다. 이는, *Hermes-v2*는 처리량뿐만 아니라 배치 사이즈까지 고려하여 같은 모델이더라도 배치 사이즈가 큰 작업을 더 좋은 GPU에 배치하기 때문이다. 예를 들어, MobileNet의 경우 2080S와 TITAN V는 3.5% 정도의 처리량 차이가 난다. 이 때, 배치 사이즈가 16에 비해 128 배치 사이즈를 가진 작업의 경우 8배 차이가 발생한다. 따라서, *Hermes-v2*는 이런 경우 더 큰 배치 사이즈를 가진 작업을 재배치를 하므로 더 빠르게 찾을 수 있다.

또한, 우리는 GPU 종류 간 성능 차이를 이용한 재배치의 효율성을 보기 위하여 Hermes와 평균 작업 종료 시간을 비교하였다. 먼저, 성능 차이가 크지 않은 MobileNet의 경우 Hermes는 평균 441초가 걸렸지만 *Hermes-v2*는 328초로 대략 25% 정도 단축시켰다. ResNet56의 경우 Hermes는 평균 922초 걸리고 *Hermes-v2*는 626초가 걸려 32% 정도 단축했다. 이 결과를 통해, GPU 종류 간 성능 차이를 이용한 재배치 방법이 효율적이라는 것을 알 수 있다.

GPU 종류 간 성능 측정 오버헤드 분석: GPU 종류 간 성능 측정의 오버헤드를 분석하기 위해 우리는 GPU 종류 간 성능 측정 및 계산에 필요한 과정을 분석하였다. 먼저, 작업의 모델 정보와 처리량과 같은 정보를 얻기 위한 모니터링 과정이 존재한다. 모니터링 과정은 딥러닝 작업의 내부 서버와 TensorFlow 작업과 통신하는 과정과 내부 서버와 노드 스케줄러의 수집 모듈과 통신하는 비용이 있다. 해당 과정들은 모두 같은 노드에서 통신하는 비용으로 각 과정은 1ms 내외의 시간이 소요된다. 또한, 5초를 주기로 정보를 수신하여 5초마다 1ms 내외의 시간이 소요되어 통신으로 인해 0.02%의 오버헤드가 발생한다.

그리고 전체 스케줄러에서 성능을 계산하고 재배치하는 부분이 있다. 성능 계산 과정은 재배치할 작업과 다른 작업들과 비교하여 $O(N)$의 시간밖에 소요되지 않는다. 또한, 재배치하는 과정은 아직 실행되지 않은 작업들이기 때문에 교환하여 GPU만 변경하면 되기 때문에 오버헤드가 거의 없다.

5. 결론

본 논문은 GPU 이기종 환경에서의 하이퍼 파라미터 가속을 위해 딥러닝 모델의 GPU 종류에 따른 성능 차이를 이용한 스케줄러인 *Hermes-v2*를 제안한다. *Hermes-v2*는 딥러닝 모델이 GPU에 따른 성능 차이 경향성이 비슷하다는 것을 이용하여 성능 차이를 측정한다. *Hermes-v2*는 위의 측정을 기반으로 성능 차이가 가장 큰 작업을 재배치하는 방식의 스케줄러 프레임워크를 개발했다. 실험 결과, *Hermes-v2*는 적은 오버헤드를 가지며 GPU 성능 차이에 따라 모델 성능 차이를 고려해 작업을 재배치하여 하이퍼 파라미터 최적화 과속을 30% 정도 단축했으며 선행 연구 Hermes와 비교해 작업 종료 시간도 최대 32%를 단축했다.

참고 문헌

[1] 손재원, "클라우드 환경에서 딥러닝 하이퍼 파라미터 최적화 가속을 위한 GPU 스케줄링 프레임워크," 석사학위논문, 서강대학교, 2020.

[2] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A GPU Cluster Manager for Distributed Deep Learning," in *Proc. 16th USENIX Symp. on Networked Systems Design and Implementation (NSDI '19)*, (Boston, MA), pp. 485–500, Feb. 2019.

[3] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *Proc. 13th EuroSys Conference (EuroSys '18)*, (Porto, Portugal), pp. 1–14, 2018.

[4] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and Efficient GPU Cluster Scheduling," in *Proc. 17th USENIX Symp. on Networked Systems Design and Implementation (NSDI '20)*, (Santa Clara, CA), pp. 289–304, Feb. 2020.

[5] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective Cluster Scheduling for Deep Learning," in *Proc. 13th USENIX Symp. on Operating Systems Design and Implementation (OSDI '18)*, (Carlsbad, CA), pp. 595–610, Oct. 2018.

[6] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "Allox: compute allocation in hybrid clusters," in *Proceedings of the Fifteenth European Conference on Computer Systems*, pp. 1–16, 2020.

[7] S. Chaudhary, R. Ramjee, M. Sivathanu, N. Kwatra, and S. Viswanatha, "Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning," in *Proceedings of the Fifteenth European Conference on Computer Systems*, pp. 1–16, 2020.