석 사 학 위 논 문

GPU 가상화 환경에서 강화 학습을 이용한 작업 배치 기법

A Job Placement Using Reinforcement Learning
in GPU Virtualization Environment

숙명여자대학교 대학원

컴퓨터과학과 컴퓨터과학전공

오 지 선

석 사 학 위 논 문

GPU 가상화 환경에서 강화 학습을 이용한 작업 배치 기법

A Job Placement Using Reinforcement Learning
in GPU Virtualization Environment

숙명여자대학교 대학원

컴퓨터과학과 컴퓨터과학전공

오 지 선

석 사 학 위 논 문

GPU 가상화 환경에서 강화 학습을 이용한 작업 배치 기법

A Job Placement Using Reinforcement Learning in GPU Virtualization Environment

지도교수 김 윤 희

이 논문을 공학 석사 학위 논문으로 제출함

2019년 12월

숙명여자대학교 대학원

컴퓨터과학과 컴퓨터과학전공

오 지 선

오지선의 공학 석사 학위 청구 논문을 인준함

GPU 가상화 환경에서 강화 학습을 이용한 작업 배치 기법

A Job Placement Using Reinforcement Learning
in GPU Virtualization Environment

2019년 12월

심사위원장 (인)

위 원 (인)

위 원 (인)

숙명여자대학교 대학원

목 차

표 목 차

그 림 목 차

국 문 요 약

GPU는 생물, 화학 등 계산 과학 분야 및 이미지, 영상 등 머신 러닝 분야에서 고속 처리를 위하여 널리 사용된다. 최근 데이터 센터와 클라우드 업체에서는 GPU를 채택하여 컴퓨팅 자원으로 제공하고 있다. 대부분의 클라우드 제공자는 사용자에게 전용 엑세스 방식으로 GPU 자원을 할당하기 때문에 점유한 GPU 자원을 모두 사용하지 않을 수 있다. 이를 해결하기 위해 여러 사용자에게 GPU 자원을 공유하도록 할당할 수 있으나 상호 간의 간섭으로 인한 개별 작업의 성능 저하가 발생할 수 있다. 뿐만 아니라 클라우드 제공자가 다양한 자원 상에서 다양한 응용의 특성을 고려하면서 성능을 예측하거나 휴리스틱하게 제어하는 것은 어렵다. 그러므로 상호 작업 간 간섭을 최소화하고 자원 활용도를 높이는 지능적 작업 배치 기법이 필요하다.

본 연구에서는 응용의 자원 사용 이력을 정의하고 이를 입력으로 하는 강화 학습 기법 기반 작업 배치 기법을 제안한다. 자원 사용 이력 학습은 강화 학습의 하나인 DQN 모델을 사용한다. 학습을 통해 현재 자원의 상태를 초과하지 않으며 실행 시 전체 성능에 영향을 적게 줄 공동 배치될 여러 작업을 예측하여 제공한다. 이를 통해 다양한 실행 특성을 가지는 응용들의 성능 저하를 방지하며, 자원을 공유하여 실행함으로써 자원 활용도를 높인다. 다양한 자원 사용 특성을 가지는 워크로드를 제안하는 학습 기법과 다른 기법들을 사용해 분석하여 본 연구의 우수성을 보인다. 실험을 통해 제안하는 기법이 전체 수행 시간 단축과 효율적인 자원 사용이 가능하며, 실행한 작업 각각의 성능 역시 보존할 수 있음을 보인다.

주제어: GPU, 응용 작업 이력, DQN 학습, 간섭 예측, 다중 작업 배치

I. 서 론

1. 연구 배경

GPU(Graphics Processing Unit)는 수천 개의 프로세싱 코어로 구성되어 높은 병렬 처리 연산을 수행한다. 높은 컴퓨팅 가속 처리의 장점으로 GPGPU(General-Purpose GPU)의 기능으로써 머신 러닝(ML: Machine Learning), 고성능 컴퓨팅(HPC: High Performance Computing) 응용 등 다양한 분야에서 폭 넓게 사용되고 있다. 이러한 이유로 클라우드 및 서버 인프라에서는 사용자의 다양한 응용 프로그램의 실행을 위해 GPU 서버를 제공하고 있다. Amazon EC2 [1], Nimbix [2], Microsoft Azure [3], Alibaba [4] 와 같은 여러 대규모 클라우드 회사는 GPU 서비스를 제공한다. 이러한 서비스 제공자 (e.g., Kubernetes [5], Mesos [6])는 GPU 리소스가 요청되면 사용자에게 주문형 엑세스 대신 전용 엑세스 권한을 제공한다. 이러한 전용 엑세스 권한은 대기 시간이 긴 응용 프로그램인 경우 전체 GPU 리소스 접근을 차단하여 전체 시스템 처리량을 줄일 수 있다 [7]. 이러한 결과로 GPU를 충분히 활용하지 못하여 유휴 컴퓨팅 리소스를 초래할 수 있다.

컴퓨팅 인프라의 목표는 높은 리소스 사용률 달성과 운영 비용 절감에 있다. 현존하는 대다수의 컴퓨팅 인프라에서는 이러한 목표를 위해 CPU, 메모리 등의 자원을 효율적으로 공유하도록 제공한다 [46]. GPU는 물리적/가상 클러스터 시스템에서 제공하는 다양한 자원 중

높은 비용과 에너지 사용량을 가진다 [43]. 따라서 GPU 자원 역시 운영에 있어서 경제적 이점을 위해 GPU를 공유해야 한다. 최소 수의 GPU 인스턴스를 확보하고, 여러 시스템에서 GPU를 공유하여 사용함으로써 운영비용을 절감할 수 있다 [44], [45]. 또한 높은 GPU 가격은 여러 사용자가 GPU 자원을 공유하면 보다 저렴하게 제공받을 수 있다.

GPU를 공유하여 여러 응용 프로그램을 공동으로 실행하는 것은 GPU 자원 활용도를 높일 수 있다. 최근 GPU 공유를 위해 NVIDIA는 실행 중인 여러 응용의 커널을 하나의 프로세스로 실행하는 MPS(Multi-Process Service) [8]를 제공한다. 그러나 이 기술은 응용의 커널 패턴을 알아야 성능 향상에 도움이 될 수 있다. GPU 자원에서의 응용 공동 실행을 위한 스케줄링 기존 연구가 존재한다. 모니터링을 기반으로 하여 응용을 공동 배치하는 방법 [9], [10], [11], 또는 응용의 GPU 사용 프로파일링(Profiling) 정보를 사용하여 가중치에 따라 배치하는 방법 [12]이 있다.

그러나 서버에 공동 배치된 응용 프로그램은 자원 활용도를 높일 수는 있으나 전체 수행에 성능 저하가 발생할 수 있다. 여러 작업들이 실행 될 때 공유되는 자원들로 인해 경쟁이 발생하기 때문이다. 뿐만 아니라 충분히 제공 받은 자원 외에도 서로 간섭을 발생시킬 수 있으며 성능을 예측할 수 없다. 응용 프로그램은 스케줄러에서 일반적으로 고려하는 자원(CPU, GPU, 메모리) 외에 캐시, GPU 코어(core), I/O 와 같은 기본 자원도 공유하여 사용하기 때문이다 [13], [14]. 그러나 실제 응용 프로그램은 다양한 자원 사용 특성이 존재하고, 복잡한 환경과 상호 작용이 존재하기 때문에 작업의 자원 사용량 만을 통해 간섭과 성능을 예측하는 것은 어렵다.

본 연구에서는 다양한 응용 프로그램을 대상으로 하여 경쟁이 발생하는 자원 사용 이력을 사용하여 강화 학습을 적용한 작업 배치 기법을 제안한다. 여러 응용의 실행 특성을 설명하고, 응용을 공동 실행했을 경우에 보이는 성능 저하를 통해 간섭을 예측하는 것이 어려움을 보인다. 따라서 클러스터의 환경과 응용의 자원 사용 이력을 정의하여, 이를 통해 강화 학습을 적용한 작업 배치 기법을 제안한다. 제안하는 배치 기법을 통해 다른 배치 기법과의 성능을 비교하고 작업의 성능 저하를 비교한다. 강화 학습의 트레이닝 오버헤드 분석을 통하여 추후 관리에 있어서 강화 학습 기반 작업 배치 기법이 효율적인 성능을 도출할 수 있음을 보인다. 마지막으로 GPU 공유 시 가격을 가정하고 계산하여 공유 시 비용 절감의 장점을 보인다.

2. 연구의 목표 및 내용

본 논문은 GPU 클러스터 환경에서 HPC 및 ML 응용의 작업 이력을 기반으로 강화 학습을 적용한 데이터 배치 기법을 제안한다. 자원 경쟁이 발생하는 자원에 따른 작업의 이력을 정의하여 이 이력을 입력으로 강화 학습을 적용한 데이터 배치로 수행 시간 및 자원 활용도를 높인다. 또한 공동 배치 시 각 작업 별 수행 시간과 트레이닝 오버헤드를 분석함으로써 각 작업의 성능 및 전체 성능 저하가 작음을 보인다. 본 논문의 주요 연구 내용은 다음과 같다.

첫째, 여러 응용의 실행 특성과 응용 공동 실행 시 발생하는 간섭과 자원과의 관계를 보인다.

둘째, 응용의 자원 사용 이력을 정의한다.

셋째, 자원 사용 이력을 입력으로 하는 강화 학습 기반 작업 배치 기법을 제안한다.

넷째, 다른 비교 기법과 제안하는 알고리즘에 대해 실험을 통해 성능을 검증한다.

3. 논문의 구성

본 논문의 구성은 다음과 같다. 2장에서는 자원 공유 기법과 강화 학습을 적용한 데이터 배치 기법에 대한 연구에 대해 정리하고 본 논문과 비교 분석한다. 3장에서는 응용의 특성 및 자원 공유 시 성능과의 관계를 보인다. 4장에서는 작업 이력을 사용한 강화 학습 기반 작업 배치 기법에 대해 설명한다. 5장에서는 제안한 데이터 배치 기법을 바탕으로 실험을 진행하고 결과를 분석한다. 마지막으로 6장에서는 결론을 맺는다.

II. 관련 연구

본 장에서는 GPU 클러스터 환경에서의 자원 공유를 통한 자원 관리 기법에 대해 설명한다. 또한, 다양한 기계 학습을 활용한 자원 스케줄링 기법에 대한 연구를 소개하고, 마지막으로 본 연구와 관련 연구들을 비교 분석한다.

1. GPU 클러스터 자원 관리 기법

GPU는 GPGPU 역할로서의 발전으로 기존의 데이터 센터에서 채택하여 사용하고 있다. 또한 클라우드 제공자 역시 여러 종류의 GPU 자원을 사용자에게 제공하고 있다. 하나의 노드에 하나의 사용자만 배정하는 기존의 클라우드 제공자의 서비스 방식은 상대적으로 비용이 높은 GPU 노드의 자원 낭비를 야기시킬 수 있다. 따라서 GPU 자원을 효율적으로 사용하기 위해서는 실행하는 응용의 특성을 파악하여 GPU 자원을 공유하여 관리하는 기법이 필요하다. 이를 위해 클라우드 및 클러스터 환경에서의 GPU 자원을 공유하여 관리하는 기법에 대한 연구가 진행되어왔다.

Cheol-Ho Hong [12]은 클라우드 환경에서 HPC 응용 프로그램의 GPU 사용 강도에 따른 시스템 전체 가중치 공정 공유 시스템을 제안한다. GPU 커널 실행 요청이 짧고 반복적인 경우에 발생하는 메모리 매핑 I/O 등의 오버헤드가 발생한다. 이를 위해 가상 머신(VM:

Virtual Machine)들 간 공유 메모리 영역을 사용하여 락(Lock) 없는 큐를 관리하여 해결하였다. 그러나 VM들 간의 자원 사용에 대한 공정성 관리는 HPC 응용과 같이 커널의 수와 길이가 다양한 경우에는 처리하지 못한다. Chia-Chen Chang [15]은 GPU 자원 프로비저닝(Provisioning)을 통해 쿠버네티스(Kubernetes) 기반 모니터링 및 서버 확장 플랫폼을 제안한다. 자원의 동적 프로비저닝은 모니터링, 분석, 계획, 실행 4단계로 구성하여 진행된다. 모니터링에서 수집된 수행 중인 서버의 상태를 통해 분석하며, 분석 내용을 바탕으로 프로비저닝 스케줄링을 실행한다. 그러나 [15] 연구에서 GPU와 메모리 값만을 사용해 자원 프로비저닝을 하기 때문에 다양한 자원 사용 값을 가지는 HPC 응용의 경우에는 적합하지 않다. Khaled M. Diab [16] 는 클러스터 환경에서 다른 사용자들의 GPU 자원을 공유하여 작업을 동시에 실행 가능하게 하는 시스템을 제안한다. Openstack 기반 VM을 사용하여 각 자원의 정보, 실행 응용의 정보를 수집하고 이를 기반으로 API를 가로채 두 개의 GPU 커널이 실행 가능하도록 하였다. 응용의 정보는 수행 시간, GPU 메모리를 수집하였으며 실험에서는 계산 집약적인 특성을 가지는 머신 러닝 응용 만을 대상으로 스케줄링 하였다. 그러나 HPC 응용의 경우 I/O 집약적인 특성을 가질 수 있으므로 공유되는 자원, 수집되는 자원의 정보가 추가적으로 필요하다. [17], [18] 연구에서는 GPU 자원을 공유할 수 있는 쿠버네티스 기반의 플랫폼인 GaiaGPU를 소개한다. GPU 공유는 GPU 자원을 일정한 크기로 분류하고 실행하는 컨테이너(Container)에게 할당하는 방식으로 제공한다. 처음에는 충분한 크기의 GPU를 할당하고 뒤에 들어오는 작업이 존재할 경우 비용 트리를 통해 적합한 GPU를 찾는다. 그 후 기존에 존재하던 작업의 자원의 양을 조절하여 들어오는 작업에 자원을 할당한다. 그러나 이는 자원 할당이 실제 자원의 상태, 실행 중인

작업의 최소 자원 요구량 만을 확인하기 때문에 작업이 실행 중 자원의 양보다 초과해서 요구할 경우는 고려하지 못한다.

2. 머신 러닝 기반 시스템 관리 기법

로드 밸런싱, 트래픽 라우팅, 패치 파일의 목적 분류 [19] 등의 다양한 영역에서 시스템 운영을 위한 머신 러닝 연구가 활발히 진행되고 있다. 환경, 응용의 정보, 실행 응용에 따른 환경 변화 등 과거 또는 현재의 다양한 데이터들을 입력으로 받아 응용들 간의 간섭, 작업 배치 및 자원 관리 등의 결과를 학습을 통해 얻을 수 있다. 이러한 이유로 클러스터 운영에 있어서의 머신 러닝의 적용에 대한 관심도가 높아지고 있으며 [20], 작업 및 자원 관리를 위한 연구가 활발하다 [21], [22], [23], [24], [25].

Fabiana Rossi [21]는 머신 러닝 중 강화 학습을 통해 오토-스케일링(Auto-scaling)이 가능한 자원 관리 기법을 제안한다. 강화 학습 중 Dynamic-Q를 사용하여 다음 작업 컨테이너의 최적의 수평적 스케일링(Horizontal scaling), 수직적 스케일링(Vertical scaling)을 결정한다. 런타임 비용을 최소화하는 전략에 목표를 두고 응용의 CPU 점유율 이력을 기반으로 학습하였다. Hongzi Mao [22]는 DQN(Deep Q-Networks) 강화 학습을 통해 자원 관리 기술을 제안하였다. 수행 시간에 따라 CPU, 메모리를 일정한 슬롯(slot) 단위로 나누어 입력 값으로 사용하고 학습시킨다. 이를 통해 다음 작업에 대한 자원 제공이 보상 값으로 나오며 이를 환경에 반영한다. 그러나 위의 연구들은 CPU

자원 이력을 기반으로 학습을 진행하였기 때문에 GPU 자원에서 실행했을 경우에 추가되는 자원(GPU 메모리, PCIe 등)에 대한 고려가 필요하다.

Yixin Bao [23]은 GPU 클러스터 환경에서 심층 강화 학습(DRL: Deep Reinforcement Learning)을 사용한 작업 배치 프레임워크를 제안한다. 학습 모델은 사용자, CPU, GPU 자원 총 3개의 입력을 학습시키며 이를 바탕으로 서버에 동시에 실행할 작업을 배치시킨다. 보상은 평균 작업 완료 시간 최소화를 목표로 한다. [23] 연구에서는 서로 다른 유형의 머신 러닝 응용이 존재할 때 낮은 수준의 간섭으로 작업이 배치된다고 하였다. 하지만 응용의 CPU, GPU의 이력만으로 간섭 영향을 줄일 수 있기에는 부족하며 간섭이 발생할 수 있는 다른 자원들에 대한 입력이 더 필요하다. Yash Ukidave [24]는 GPU에서 실행되는 응용의 이력 값들을 통해 머신 러닝 모델 기반 응용 간의 간섭 인지 스케줄러를 제안한다. 커널들의 길이를 다르게 하여 서로 간섭에 영향을 주는 이력들을 파악하였으나 실제 응용에서는 적용이 되지 않음을 증명하였다. 이에 실제 응용 간에 간섭이 발생할 수 있는 이력들을 정의하여 회귀 모델인 랜덤 포레스트(RF: Random Forest), 선형 회귀(LR: Linear Regression)을 통해 학습시켰다. 그러나 논문에서 정의한 이력들은 커널의 길이에 초점을 두어 학습시켰기 때문에 일정한 커널 수행 패턴을 가지는 머신 러닝 응용의 경우만 적합하다. [25] 연구는 GPU 기반 클러스터 및 클라우드 서버에서 응용 공동 실행을 위한 간섭 인식 스케줄러를 제안한다. 온라인으로 간단한 이력 만을 프로파일링하여 비어있는 SVD(Single Value Decomposition)의 테이블 값을 협업 필터링(CF: Collaborative Filtering)을 사용하여 예측한다. 그러나 협업 필터링의 경우 잘못된

예측 값으로 수렴할 수 있다. 또한 일정한 특징을 가지는 응용의 경우에는 적합하나 예측이 불가능한 응용의 경우에는 이 방법이 적합하지 않을 수 있다.

3. 관련 연구와 본 연구와의 비교

표 1은 다양한 머신 러닝을 사용한 작업 배치 기법에 대한 연구와 본 연구를 비교한 것이다. 사용한 머신 러닝 기법과 대상 자원 및 대상 응용을 비교하였다. 또한 연구에서 여러 작업의 GPU 자원의 다중 응용 실행 가능 여부와 간섭에 관련 있는 자원 이력을 학습에 고려하였는지 여부를 비교하였다. 연구 1은 일정한 자원 사용량을 가지는 가상 작업, 연구 2, 3, 4는 머신 러닝 응용을 대상으로 한다. 본 연구와 유사한 연구 2의 경우 다중 응용 실행은 가능하나 다중 실행 시 여러 자원에서 발생하는 간섭은 고려하지 않았다. 또한 본 연구와 유사한 연구 4는 간섭이 발생하는 자원의 항목은 정의하였으나 커널의 길이에 초점을 두어 정의하였으며 두 응용 쌍에서 발생하는 간섭만 고려하였다.

<표 1> 관련 연구의 논문과 본 연구와의 비교

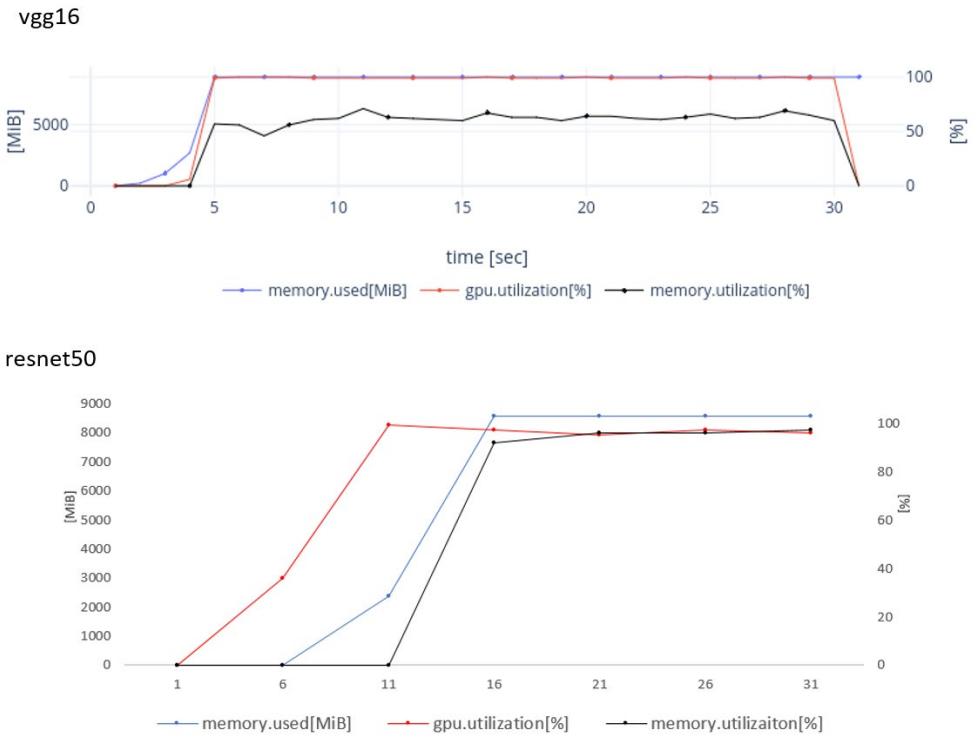
	연구1 [22]	연구2 [23]	연구3 [25]	연구4 [24]	본 연구
머신 러닝 기법	DQN	DRL	CF	RF, LR	DQN
대상 자원	CPU	GPU	GPU	GPU	GPU
대상 응용	가상 작업	머신 러닝 응용	머신 러닝 응용	머신 러닝 응용	머신 러닝 응용 & HPC 응용
다중 응용 실행 여부	X	O (다중 응용)	O (3개 응용)	O (2개 응용)	O (다중 응용)
간섭 관련 세부 자원 고려 여부	X	X	X	O	O

III. GPU 응용 특성과 자원 공유 시 성능 관계

본 장에서는 HPC 및 ML 응용의 자원 사용 특성을 설명하고, GPU 클러스터 환경에서 응용이 공동 실행 될 때의 성능 저하와 공유되는 자원과의 관계를 설명한다. 이를 통해 성능 예측을 위한 작업 배치를 위한 강화 학습의 필요성 보인다. 3.1절 에서는 HPC 및 ML 응용의 자원 사용 특성을 설명하고, 3.2절에서는 응용 공동 실행 시 성능과 공유 자원과의 관계를 보인다.

1. HPC 및 ML 응용의 자원 사용 특성

GPU 클러스터 환경에서 HPC 및 ML 응용 실행 시 다양한 자원 사용 특성을 보인다. 두 분야의 응용의 경우 방대한 데이터 처리를 위해 GPU를 사용하나 서로 다른 특성을 가진다. ML 응용의 경우 데이터 셋의 크기, 레이어의 수, batch 사이즈에 따라 수행 시간의 차이는 있으나 실행 동안 GPU 자원을 일정하게 사용한다 [26]. 그림 1은 Tensorflow 벤치마크 [27]의 ML 응용 vgg16, resnet50 실행 시 사용하는 GPU와 GPU 메모리 사용 패턴을 보여준다. 실행 처음 데이터 셋을 GPU 메모리에 올려 두는 시간을 제외하고 학습 기간 동안 GPU와 GPU 메모리 사용량이 거의 일정하며 종료될 때까지 유지되는 패턴을 가진다.



[그림 1] vgg16, resnet50 ML 응용의 자원 사용 패턴 예

HPC 응용은 과학 응용의 해석을 위해 종속적으로 실행해야하는 과정이 존재한다. 응용의 각기 다른 전처리 과정, 해석 과정에 따라 수행되기 때문에 HPC 응용은 다양한 GPU 자원 사용 패턴을 가진다. 그림 2는 HPC 응용 중 LAMMPS [28], QMCPACK [29]의 자원 사용 패턴을 보여준다. LAMMPS 응용의 경우, 수행되는 동안 평균 약 40%의 GPU 사용을 보인다. GPU 메모리는 약 363MB~8,499MB까지 점차적으로 증가하며 GPU 메모리 사용량이 가장 클 때 GPU를

100%까지 사용한다. QMCPACK 응용은 GPU 사용은 총 3단계의 계단식으로 증가한다. GPU 메모리의 경우는 응용 수행 중반까지는 사용하지 않으며, 중반 이후(약 95초)로 약 5135MB까지 급격하게 증가하는 모습을 보인다.

[그림 2] LAMMPS, QMCPACK HPC 응용의 자원 사용 패턴 예

따라서 ML 응용은 학습 및 분석하는 동안 일정한 자원 사용 패턴을 가지고 HPC 응용은 실행 단계에 따라 다양한 자원 사용 패턴을 가지기 때문에 공동 실행할 때 성능에 영향을 미치는 자원과 관계에 대한 분석을 진행한다.

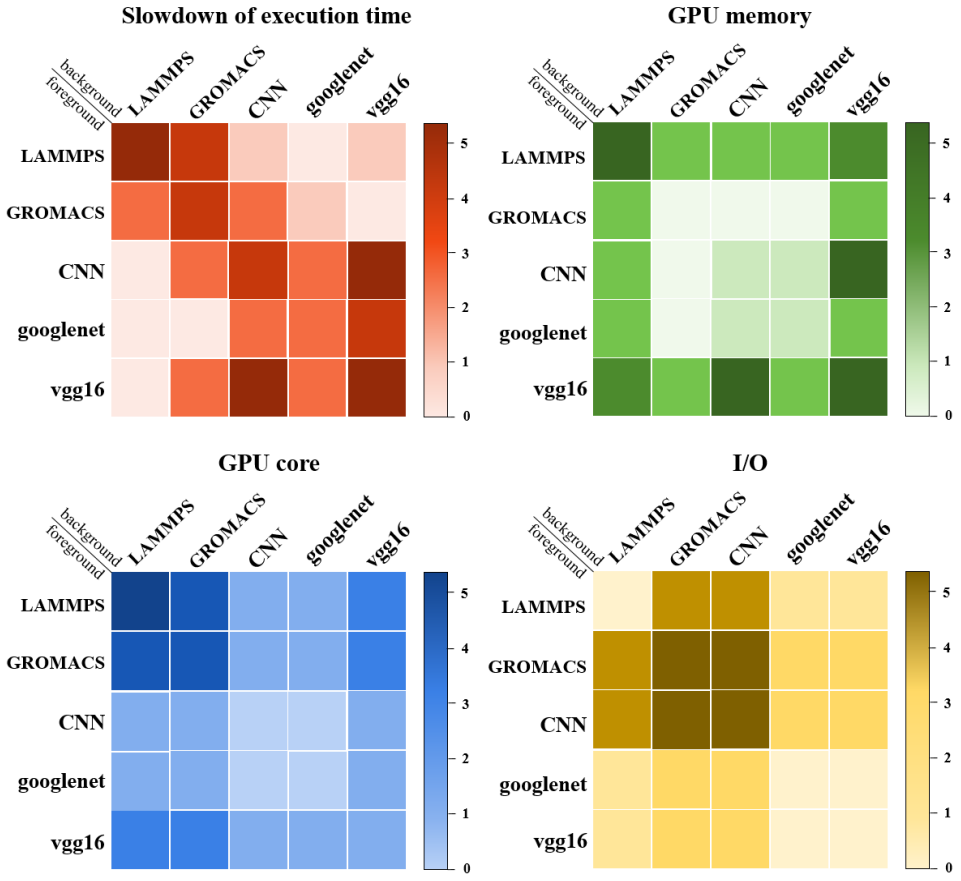
2. 응용 공동 실행 시 성능과 공유 자원과의 관계

3.1에서 살펴본 바와 같이 GPU에서 실행되는 HPC, ML 응용은 여러 자원 사용 특성을 가진다. GPU 클러스터 환경에서 자원 활용을 위해 여러 응용을 실행시킬 때 응용의 각기 다른 자원 사용 패턴으로 인하여 성능을 예측하기 어려워진다. 뿐만 아니라 응용이 충분히 자원을 제공받더라도 서로 간섭을 발생시킬 수 있다 [30]. 스케줄러에서 고려하는 자원(CPU, GPU, 메모리) 외에도 GPU 코어, I/O와 같은 자원에서도 간섭이 발생하기 때문이다 [7].

본 연구에서는 응용을 공동 배치하여 실행 시 성능과 공유되는 자원의 사용을 비교한다. 비교를 통해 성능이 공유되는 자원에 의해 영향을 받을 수는 있으나 정확한 성능 예측을 하는 데에 충분하지 않음을 보인다. 비교를 위해 유사한 자원 사용 특징을 보이는 Tensorflow 벤치마크의 ML 응용 3개(CNN, vgg16, googlenet)와 다양한 자원 사용 이력을 가지는 NGC(NVIDIA GPU Cloud) [31]의 HPC 응용 2개(LAMMPS, GROMACS)를 사용하였다.

그림 3은 2개의 응용을 배치하여 공동 실행했을 때 수행 시간과 GPU 메모리, 코어, I/O 사용량을 나타낸다. 예를 들면 수행 시간에서는 짙은 빨간색 일수록 수행 시간이 늘어난 정도, GPU 메모리에서는 짙은 초록색 일수록 사용 정도가 높음을 의미하며 5에 해당된다. 응용 (LAMMPS, LAMMPS), (vgg16, CNN), (CNN, vgg16), (vgg16, vgg16)의 경우 메모리 부족이 발생하기 때문에 수행 시간에 영향을 주는 것을 확인할 수 있다. 그러나 이 경우를 제외하면 GPU 메모리, 코어, I/O의 사용량을 통해 응용 공동 실행의 성능을 유추하는 것은

어렵다. 예를 들어 (googlenet, vgg16)의 경우 수행 시간의 성능 저하 정도는 4이나 GPU 메모리는 3, GPU 코어는 2, I/O는 1의 사용 정도를 가진다. 이는 여러 응용이 실행 될 때 공유되는 각 자원의 사용이 복합적인 상호 작용이 발생하기 때문에 성능에 영향을 주는 것을 의미한다. 각 자원의 사용량을 통해 기존의 휴리스틱(Heuristic)한 방법으로 성능을 예측하는 스케줄링은 NP-Hard 문제이며, 응용의 특성과 자원의 상태를 고려하며 성능을 예측하는 것이 필요하다.



<그림 3> 응용 공동 배치 실행 시 성능과 자원 사용량과의 관계 비교

IV. GPU 응용의 다중 작업을 위한 강화 학습 기반 작업 배치 기법

본 장에서는 GPU 클러스터 환경에서 GPU 응용의 다중 작업을 위한 강화 학습 기반 작업 배치 기법에 대해 소개한다. 4.1절에서는 학습을 위한 응용의 각 자원 별 세부 작업 이력 항목을 정의한다. 4.2절에서는 강화 학습 기반 작업 이력 학습 모델에 대해 설명하고 4.3절에서는 자원 공유 작업 배치 서비스 모델에 대해 설명한다.

1. GPU 응용의 자원 사용 이력 항목

응용을 공동 배치하여 실행시킬 때 성능 저하가 발생할 뿐만 아니라 복잡한 컴퓨팅 시스템 환경에 맞춰 실행을 관리하는 것은 어렵다. 환경을 관찰하고 이에 맞춰 최적의 정책을 결정해 나가는 강화 학습은 이러한 클라우드 및 클러스터 컴퓨팅 시스템에 적용하기 적합하다 [32], [33].

본 연구는 강화 학습 사용을 위해 수집할 응용의 자원 사용 이력 항목을 정의한다. 표 2는 GPU 클러스터 환경에서 GPU 응용에 대한 작업 이력의 항목을 나타내며 이는 크게 3가지 범주로 구분된다. 첫 번째, GPU 응용의 사전 정보는 사용자가 응용을 수행하기 위해 시스템에 제출하고자 하는 정보이며 응용 이름, 응용 타입, input 파일

크기와 input 파라미터를 필요로 한다. 두 번째, GPU 응용의 프로파일링 정보는 응용 실행 시 온라인으로 수집되는 정보를 나타낸다. 응용이 실행 될 때 사용하는 자원과 수행 시간을 포함한다. 수집된 자원은 여러 GPU 응용이 실행 될 때 자원 경쟁이 발생하여 성능 저하를 유발시킬 수 있는 항목이다. 자원은 GPU, GPU 메모리, GPU 코어, PCIe 사용 프로파일링 정보를 포함한다. 각 항목에 대한 사용량은 일정한 시간 동안 주기적으로 모니터링하여 수집한다. 세 번째, 클러스터 환경 정보는 작업을 수행하는데 사용된 각 자원의 정보이다. 시간 마다 수집된 GPU 응용 프로파일링 정보와 환경 정보를 통해 다중 작업 배치 시 자원에 과도하게 배치하는 것을 방지하기 위해 사용된다.

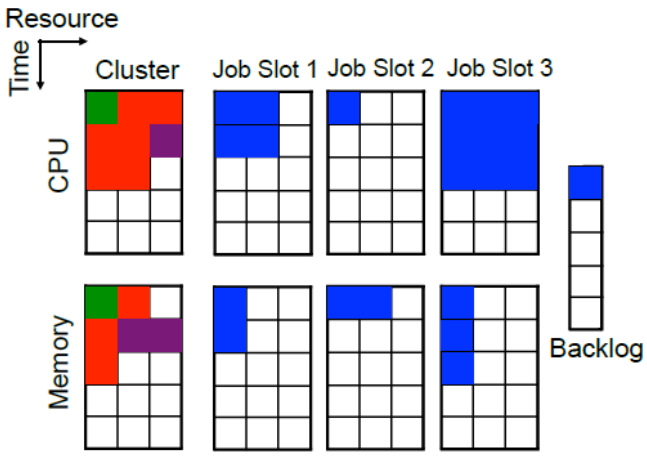
<표 2> GPU 응용 자원 사용 이력 항목

범주	속성
GPU 응용 사전 정보	응용 이름
	응용 타입
	Input 파일 크기
	Input 파라미터
GPU 응용 프로파일링 정보	GPU 활용도
	GPU 메모리 사용량
	GPU 코어 사용량
	PCIe 처리량
	수행 시간
클러스터 환경 정보	노드 이름
	GPU 카드
	GPU 구조
	GPU 메모리
	GPU 코어 수
	PCIe 대역폭

2. 강화 학습 DQN 모델

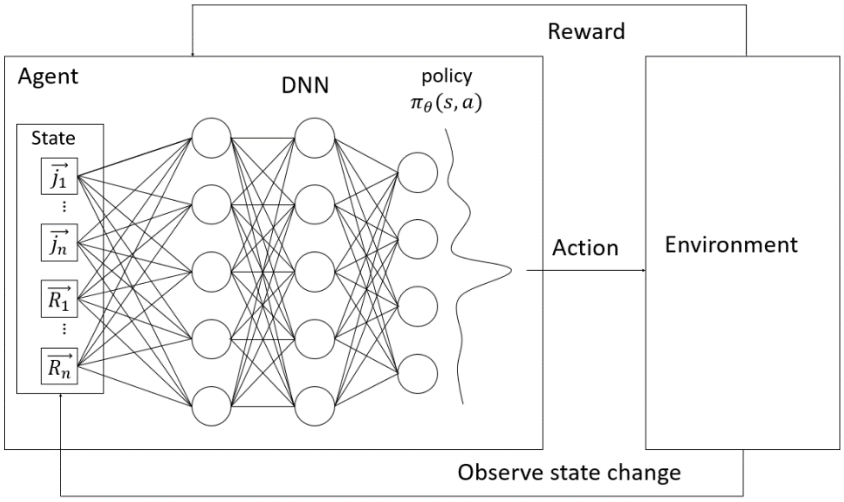
강화 학습은 시스템과 직접적인 상호 작용을 통해 더 나은 결정을 내리는 전략을 학습하는 것으로 목표로 한다[34]. 강화 학습 전략은 어떠한 상황에서 무엇을 행동 하는지를 반복적으로 학습하기 때문에 사전 지식 없이 적용할 수 있다는 장점을 가지고 있다. 그러나 기존의 강화 학습 기법은 수많은 입력 데이터를 테이블 형식으로 저장하고 문제를 푸는 것에 한계가 있다. DQN은 AlphaGo를 개발한 DeepMind 팀에서 처음 소개한 기법 [35]으로 기존 강화 학습의 문제점을 개선하기 위해 DNN(Deep Neural Network)를 함수 근사기(Function Approximator)로 사용한다.

본 연구의 학습 모델은 기존 DQN 모델 기반 작업 배치 모델 deepRM [36]를 수정하였다. 기존의 deepRM은 1)가상 작업을 생성하여, 2)CPU, 메모리 자원을 대상으로 가상 작업은 일정한 자원 사용량 만을 가진다(그림 4). 또한 3)학습 결과는 단일 작업 배치 수행만을 고려하였다. 따라서 본 연구에서는 실제 응용 작업을 대상으로 다양한 자원 사용 특성을 반영한 DQN 모델을 통해 다중 작업 배치 기법을 제안한다. 1)실제 GPU HPC, ML 응용을 대상으로, 2)응용의 변화하는 GPU, GPU 메모리, GPU 코어, PCIe 자원 사용 이력을 학습시킨다. 3)실제 응용의 가상 작업 슬롯을 생성하여 다중 작업 배치 기법을 고려하였다. 4)더 작은 분포 편차를 위해 정책 파라미터를 수정하였다.



<그림 4> 기존 DQN 모델 기반 작업 배치 기법[36]

그림 5는 에이전트(Agent)가 환경(Environment)과 상호 작용하며 학습과 보상, 관찰을 반복하는 전반적인 DQN 구조를 나타낸다. 단계 t에서 에이전트는 현재 상태 s_t 를 관찰하고 행동(Action) a 를 수행하도록 요청 받는다. 행동 후 상태가 s_{t+1}로 전환되고 에이전트는 보상 r_t를 받는다. 이 때 상태 전환 및 보상은 환경 상태와 에이전트가 수행한 행동에만 영향을 받기 때문에 마르코프(Markov) 속성을 갖는다. 에이전트는 학습 과정을 반복하며 예상 누적 할인 보상(Cumulative Discounted Reward)을 최대화 하는 것을 목표로 한다. 할인 누적 보상은 $J(\theta) = \max \mathrm{E}[\Sigma_{t=0}^{\infty}\gamma^t r_t]$ 으로 표현되고 할인 요인(Discount Factor)인 γ은 0과 1 사이의 값이다. 여기서 행동은 에이전트의 정책을 기반으로 선택 되는데 정책은 $\pi:\ \pi(s,a) \rightarrow [0,1];\ \pi(s,a)$는 상태 s에서 행동 a가 실행될 확률로 나타낸다. 그러나 위에서도 언급했듯이 수 많은 {state, action} 값을 테이블로 저장하는 것이 어렵기 때문에 함수 근사기 DNN를 사용하여 정책 파라미터인 θ를 구한다. 따라서 정책은 $\pi_\theta(s,a)$로 나타낼 수 있다.



<그림 5> 제안하는 DNN 기반 DQN 구조

따라서 DQN의 최종 목표는 정책 π_θ에 따라 상태 s에서 행동 a를 수행할 때 예상 되는 할인 누적 보상 $Q^{\pi_\theta}(s,\mathrm{a})$을 최대화 하는 것이다(식 1).

$$\nabla_\theta J(\theta) = E_{\pi\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta log\pi_\theta(s,\mathrm{a}) Q^{\pi_\theta}(s,\mathrm{a})\right] \qquad (1)$$

이 때 강화 학습은 Q 값 분포의 편차를 줄이기 위해 경사 하강법(Gradient descent) [37]를 사용하여 정책 파라미터 θ를 업데이트한다. 기존의 경사 하강법은 잘못된 값으로 편향 될 수 있는 Q 대신 정책에 따라 얻은 경험적 할인 누적 보상 V만을 사용한다. 본 연구에서는 더 낮은 분포를 가지는 정책 파라미터를 위해 예측 할인

누적 보상 Q^{π_θ}에서 정책에 따라 얻은 경험적 할인 누적 보상 V^{π_θ}를 뺌으로써 경사 하강 값을 예측한다(식 2).

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta log\pi_\theta(s_t, a_t)(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \qquad (2)$$

파라미터를 수정한 전반적인 DQN 모델 트레이닝 알고리즘은 그림 6과 같다.

Algorithm 1 DqnGPU Model Training

```
Initialize Q(s,a) and Model(s,a), ∀s ∈ S, ∀a ∈ A(s)
for each iteration:
  for each iteration:
    run jobs' episode i = 1, ..., N:
      Select a_i = max_a Q(s_i, a)
      Execute action a_i in emulator and observe reward r_i
      and image x_{i+1}
      Set s_{i+1} = s_i, a_i, x_{i+1} and update reward r_i, policy q_i
    Compute returns: v_t^i = Σ_{s=t}^{Li} γ^{s-t} r_s^i
    For t = i to L:
      For i = 1 to N:
        Δθ ← Δθ + α∇_θ log π_θ(s_t^i, a_t^i)(q_t^i - v_t^i)
      end
    end
  end
  θ ← θ + Δθ
end
```

<그림 6> DQN 모델 트레이닝 알고리즘

공동 작업 배치를 위해 DQN 모델 학습의 입력으로 4.1절의 표 2에서 정의한 응용의 자원 사용 이력을 사용한다. 그림 5에서 볼 수 있듯이 모델의 입력 상태 $s = (j, R)$ 는 작업 벡터 j와 클러스터 자원

벡터 R를 포함한다. 클러스터의 자원 환경 이력 항목을 통해 클러스터 자원은 4개의 튜플(tuple) 세트 식 3과 같이 나타낼 수 있으며 클러스터 각 자원의 사용 가능한 자원 양을 의미한다. 작업 j는 각 작업 당 일정한 timestep T 마다 수집된 자원의 프로파일링 이력 정보이다. j_i는 해당 응용 APP의 timestep T_j마다 변화하는 자원 사용량의 정보를 가지고 있으며 식 4, 5와 같이 나타낼 수 있다.

$$R \in \{R_{gpu}, R_{gpumem}, R_{gpucore,} R_{pcie}\} \qquad (3)$$

$$j_i \in \{APP_{id}, T_j, \vec{r_j}\} \qquad (4)$$

$$\vec{r_j} = (r_{j.gpu}, r_{j.gpumem}, r_{j.gpucore}, r_{j.pcie}) \qquad (5)$$

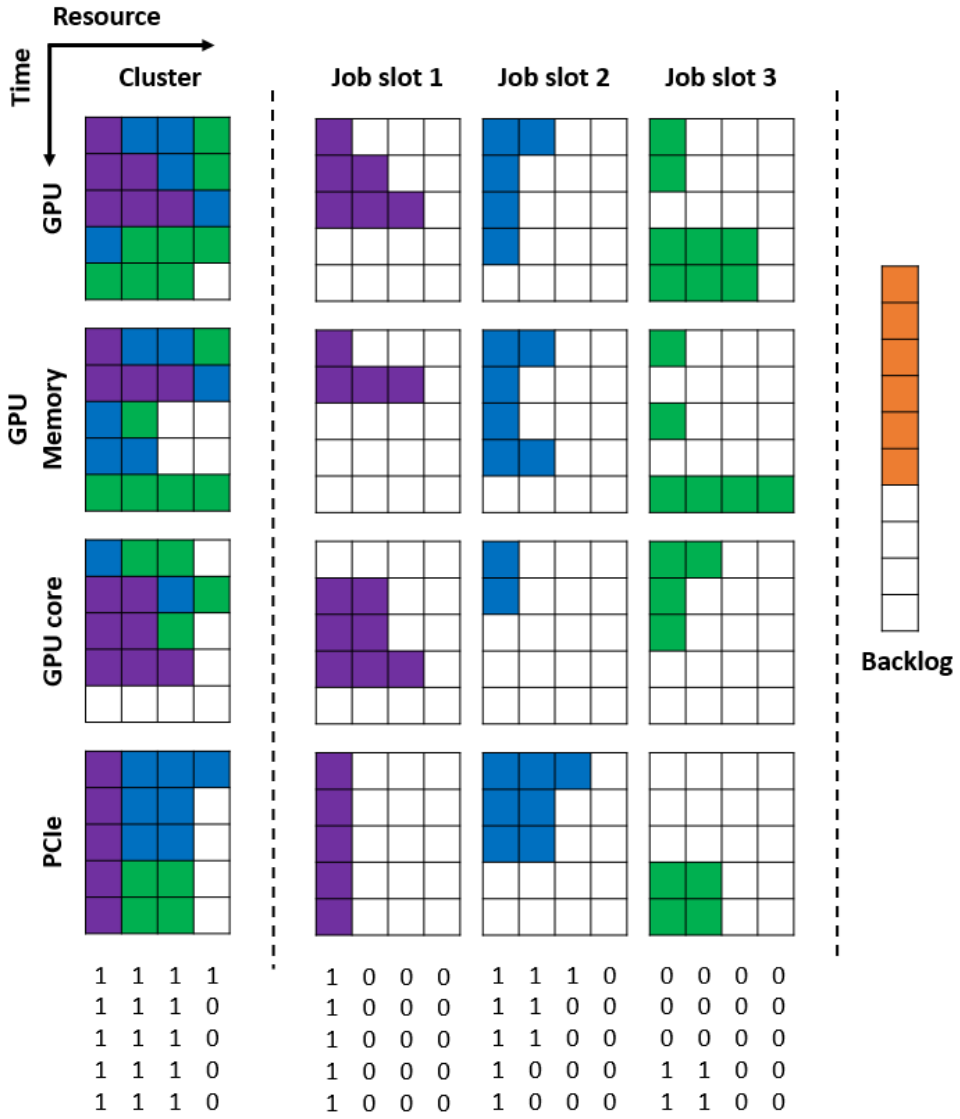
상태 공간은 클러스터 환경의 현재 할당된 자원과 예약 대기 중인 작업의 자원 프로파일링 이력을 별개의 이미지로 나타낸다. 그림 7은 시스템의 상태 공간의 예이다. 클러스터 이미지는 각 자원에 대한 이미지 4개로 구성되어 있으며 시간 T마다 예정된 작업들에 대한 자원 별 할당을 보여준다. 작업 이미지는 온라인으로 수집된 응용의 프로파일링 이력을 기반으로 T 단계마다 사용하는 자원 양을 보여준다. 예를 들어, 그림 7의 보라색 작업은 다음 5개의 시간 간격 당 {0, (1, 1, 0, 1)}, {1, (2, 3, 2, 1)}, {2, (3, 0, 2, 1)}, {3, (0, 0, 3, 1)}, {4, (0, 0, 0, 1)}의 자원 요구 사항을 나타낸다. 이를 기반으로 작업의 자원 별 이미지를 0과 1로 구성된 테이블로 구성한다. 작업 이미지는 T단계에 사용되는 자원이 있는 경우 자원의 사용량만큼 1, 나머지는 0으로 채운 테이블로 표현된다. 이렇게 구성된 공동으로 배치할 여러 작업 이미지들을 기반으로 클러스터 이미지를 구성하며 이 때 $\sum_j r_{j.gpu} \cdot T_j \leq R_{gpu} \cdot \mathrm{T}$, $\sum_j r_{j.gpumem} \cdot T_j \leq R_{gpumem} \cdot \mathrm{T}$, $\sum_j r_{j.gpucore} \cdot T_j \leq R_{gpucore} \cdot \mathrm{T}$, $\sum_j r_{j.pcie} \cdot T_j \leq R_{pcie} \cdot \mathrm{T}$를 만족해야 한다. 따라서 그림 7의 예와 같이

PCIe 자원을 대상으로 클러스터 이미지와 3개의 작업 이미지 테이블을 표현할 수 있다.

신경망의 입력은 고정된 상태로 표현되는 것이 바람직하기 때문에 작업 이미지 슬롯의 수 M 만큼을 입력으로 사용한다. 그림 7에서처럼 3개의 작업 이미지 슬롯이 존재할 때 3개의 작업이 학습의 입력으로 사용될 수 있다. 학습을 통한 결과는 다음 행동 a 를 결정하는데 사용된다. 행동 공간 $a = \{0, 1, \dots, M\}$로 표현된다. $a = 1$인 경우는 첫 번째 작업 슬롯에 할당된 작업을 실행하라는 것을 의미한다. $a = 0$ 은 행동 보류이며 현재 T 단계에서 클러스터 이미지에 작업 이미지가 들어갈 공간이 없다는 것을 나타낸다. 따라서 T 단계씩 진행되면서 적합한 작업 a 가 결정, 실행되며 자원 클러스터 이미지가 업데이트 됨으로써 반복한다. M개 이후의 작업에 대한 정보는 상태 공간의 백 로그 구성 요소에 포함된다. 빈 작업 슬롯이 존재할 경우 백 로그의 작업으로 채운다. 그러나 위의 방법은 결과 a 에 따라 하나의 작업만 실행 가능하기 때문에 작업을 공동으로 배치하여 실행할 수 없다.

본 연구에서는 다중 작업 배치를 위해 신경망의 입력을 수정하여 학습하였다. 작업 이미지 슬롯 M 개에 따라 가상 작업 이미지 슬롯을 생성한다. 예를 들어 상태 공간에 작업 이미지 슬롯이 3개가 존재하고 각 슬롯 당 a, b, c의 작업이 존재한다고 가정한다. 가상 작업 이미지 슬롯은 a, b, c, ab, ac, bc, abc 총 7개로 구성되고 구성된 작업의 자원 사용량에 따라 테이블이 생성된다. 이 가상 작업 이미지 슬롯을 입력으로 하여 학습된다. 학습의 결과로 행동 $a =$ ab인 경우에 작업 a와 b를 공동 배치하여 실행한다. 따라서 공동 배치를 위한 신경망 학습의 입력은 작업 이미지 슬롯 M개의 조합인 $\sum_{x=1}^{M} {}_{M}C_{x} = 2^{M} - 1$개로 구성된 가상 작업 이미지 슬롯이 사용된다. 학습의 결과로 응용 M개 조합 중

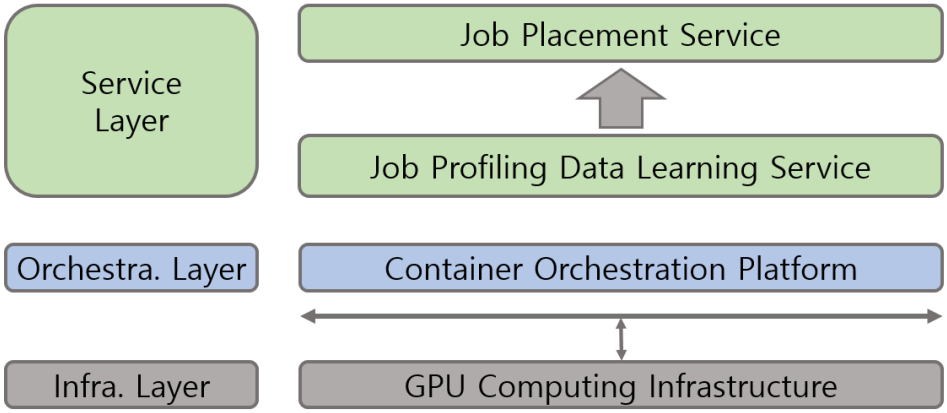
한 경우가 나오며 이를 통해 공동 배치할 작업이 결정되고 실행된다.



<그림 7> 제안 하는 기법의 4개의 자원과 대기 작업 슬롯에 대한 상태 공간 예

3. 강화 학습 모델 기반 작업 배치 기법

그림 8은 본 연구에서 제안하는 학습 기반 작업 배치 서비스 모델의 구조도를 보여준다. 본 논문에서는 인프라는 GPU 컴퓨팅 컨테이너를 대상으로 하였으며, 오케스트레이션 플랫폼은 쿠버네티스를 사용하였다. 제안하는 서비스 모델은 4.2 장에서 제안하는 작업 이력 학습 모델과 학습 결과를 반영한 작업 배치 서비스를 제공한다.



<그림 8> 학습 기반 작업 배치 서비스 모델

작업 배치 서비스는 앞서 추론된 응용의 특성과 시스템 환경에 맞춰 나온 행동 값 즉, 공동 배치 될 작업들을 실제 배치하여 실행하고 시스템과 작업의 상태를 모니터링한다. 이에 따라 작업의 실패, 종료 등을 파악하고 다음 작업들을 자원에 배치하는 서비스를 제공한다.

본 연구에서는 사용자가 수행하고자 하는 GPU 응용의 프로파일링 정보가 존재한다고 가정한다. 또한 수행하고자 하는 응용의 라이브러리 및 컴파일러가 포함된 컨테이너 이미지가 이미지 저장소인 레지스트리에 존재한다고 가정한다.

Algorithm 2 Job Placement Algorithm

1: Submit Application App_i
2: Find $Profiling\ \ Data_i, ClusterStatus$
3: Set $IuputData_i = \{Profiling\ \ Data_i, ClusterStatus\}$
4: Order of $jobs_j$ and $|jobs_j| \leftarrow DqnGPU(IuputData_i)$
5: For each $jobs_j$ to $jobs_{j...n}$ do
6: Assign $jobs_j$ to GPU
7: MonitoringSystem()
8: end

<그림 9> 학습 결과 기반 작업 배치 알고리즘

그림 9는 강화 학습 모델을 기반으로 사용자가 제출한 응용의 프로파일링 정보 이력과 클러스터 자원의 상태를 반영하면서 공동 실행할 작업을 배치하는 전반적인 흐름을 나타내는 알고리즘이다. 사용자가 실행하고자 하는 응용을 하는 응용을 제출한다(line 1). 제출한 응용의 프로파일링 데이터($Profiling\ \ Data$)와 시스템의 상태 정보($ClusterStatus$)를 확인하고(line 2) 학습 모델의 입력으로 사용할 입력 데이터를 설정한다(line 3).

$$Profiling\ \ Data = \text{식 } 4\text{의 } j_i \in \{APP_{id}, T_j, \vec{r_j}\}$$

$$ClusterStatus = \text{식 } 3\text{의 } \mathrm{R} \in \{R_{gpu}, R_{gpumem,} R_{gpucore}, R_{pcie}\}$$

이 후 강화 학습을 적용하여 전체 워크로드 큐의 작업의 순서와 그 순서에 공동 배치될 작업들을 결정한다(line 4). 추론된 작업들의 순서에 따라 작업들을 GPU에 공동 배치하고 실행되며(line 6) 작업의 상태를 모니터링 한다(line 7). 이는 전체 작업을 배치할 때까지 반복하여 수행한다(line 5−8).

V. 실험 및 결과

본 장에서는 제안한 기법의 성능을 평가하기 위해 실험을 진행하고 그 결과를 분석했다. 5.1절에서는 실험 대상 응용 및 비교 기법을 간략하게 설명하고, 5.2 절에서는 실험 환경을 소개한다. 마지막으로 진행한 실험에 대해 설명하고 결과를 분석한다.

1. 대상 응용 및 비교 기법

제안하는 작업 배치 기법의 성능을 검증하기 위해 다음의 HPC 응용 및 ML 벤치마크 응용을 사용했다. HPC 응용은 NGC에서 제공하는 응용 4개와 Tensorflow 벤치마크 응용 7개를 실험의 대상 응용으로 한다. 실험에 사용한 응용의 이름은 표 3과 같다.

<표 3> 실험 대상 응용

HPC	ML	
LAMMPS	googlenet	resnet50
GROMACS[38]	alexnet	renet101
QMCPACK	vgg11	resnet152
HOOMD[39]	vgg16	

실험을 위해 위 응용의 자원 사용 이력에 따라 작업 부하 별 워크로드 5가지를 생성하였다. GPU 활용도 평균 40%를 기준으로 GPU_heavy, GPU_light 워크로드와 GPU 메모리 활용도 50%를 기준으로 GPU 메모리_heavy, GPU 메모리_light 워크로드, 모든 응용이 무작위로 섞여 있는 random 워크로드 총 5가지 워크로드를 구성하였다. 각 워크로드는 기준에 속하는 응용을 총 30개로 랜덤하게 생성한다. 총 수행 시간, GPU 활용도 평균, GPU 메모리 활용도 평균을 측정했다.

본 연구와의 비교를 위해 4가지의 작업 배치 기법을 사용한다. 단일 작업 배치 기법인 SJF와, 다중 작업 배치 기법인 Tetris[40], DeepRM Max, DeepRM Mean을 사용하였다. DeepRM Max와 Mean은 강화 학습을 적용한 배치 기법이다. 각 작업 배치 기법에 대한 설명은 아래와 같다.

- SJF: 작업의 수행 시간이 증가하는 순서로 작업을 배치
- Tetris: 응용의 작업 사용량과 자원 가용성에 따라 packing 방식으로 배치
- DqnGPU: 본 연구에서 제안하는 배치 기법
- DeepRM Max: 작업의 자원 사용량 최대 값을 입력으로 하여 강화 학습을 적용하여 배치
- DeepRM Mean: 작업의 자원 사용량 평균 값을 입력으로 하여 강화 학습을 적용하여 배치

2. 실험 환경

실험을 위해 GPU 기반 컨테이너 클러스터 환경을 구축하였다. 클러스터 환경은 컨테이너 오케스트레이션 플랫폼인 쿠버네티스를 사용하였다. 쿠버네티스는 CaaS(Container as a Service) 형태의 클러스터 컴퓨팅 오픈 소스 프로젝트로 컨테이너의 배치, 스케일링, 운영을 자동화하기 위한 서비스를 제공한다. Amazon, Alibaba, Samsung SDS, baidu, Huawei, IBM 등과 같은 대규모 기업들에서 클라우드 운영에 있어서 채택하여 사용하고 있다.

노드의 여러 작업 배치를 위해 쿠버네티스의 기본 장치 플러그인 k8s를 수정하였다. 기존 플러그인은 쿠버네티스 환경에서의 서버 즉, 다중 작업 실행을 위한 여러 서버가 생성되지 않는다. 또한 노드의 단일 카드 내에 여러 작업의 공동 실행이 지원되지 않는다. 따라서 본 연구의 실험을 위해 다중 서버 생성 및 다중 작업 실행이 가능하도록 플러그인을 수정하였다. 오픈 소스인 Alibaba fake GPU [41]를 참조하여 수정하였다. 또한 작업들은 NVIDIA docker 컨테이너 [42]를 사용하여 실행하였다. 실험에 사용된 컴퓨팅 노드의 사양은 아래 표 4와 같다.

<표 4> 쿠버네티스 클러스터 환경 정보

	Node Details	
	CPU(master)	GPU(node)
Architecture	Intel® Core™ i7-5820K	Nvidia GeForce Titan Xp D5x

	Node Details	
	CPU(master)	GPU(node)
Core Clock	3.30GHz	1.58GHz
Num of Cores	6	3840
Mem. Size	32GB	12GB
Threading API	-	Nvidia CUDA 10.0
PCIe bandwidth	-	32GB/s
OS	Ubuntu 16.04.6 LTS	Ubuntu 16.04.6 LTS

3. 실험 및 결과 분석

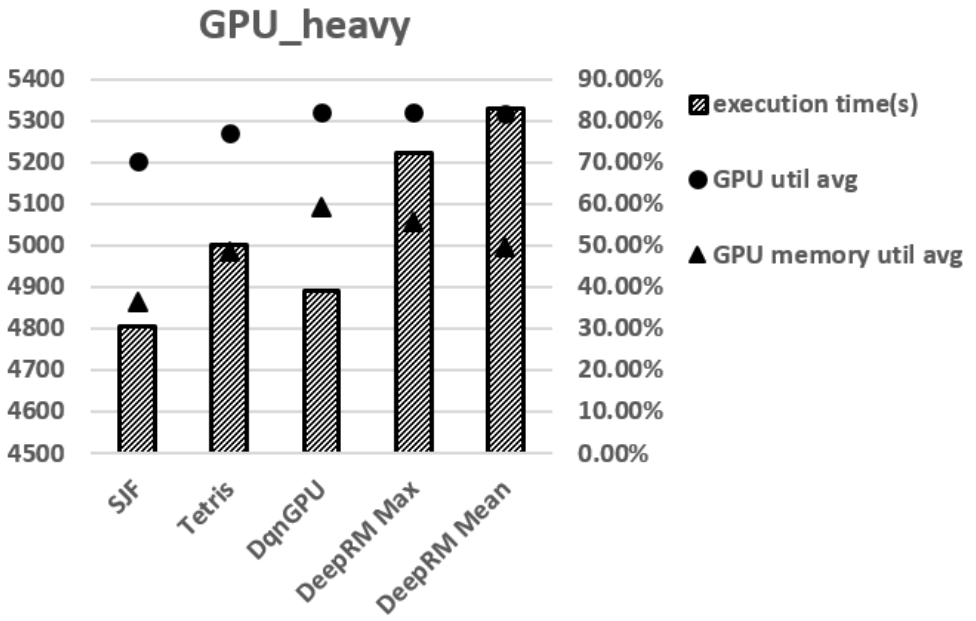
제안하는 작업 배치 기법의 성능을 위하여 작업 부하 별 성능 분석과 작업의 속도 저하, 트레이닝 오버헤드를 비교 분석한다. 또한 가격 비교를 통해 공동 배치의 장점을 보인다.

1) 작업 부하 별 수행 결과 분석

실험에서는 제안한 학습 모델을 통해 추론된 작업 배치 결과를 이용하여 수행한 결과를 다른 배치 기법과 비교하여 보여준다.

GPU_heavy 워크로드는 평균 40% 이상의 GPU 활용도를 보이는 vgg11, vgg16, resnet50, resnet101, resnet152, QMCPACK, HOOMD, LAMMPS 응용으로 구성되었다. 그림 10은 GPU_heavy 워크로드

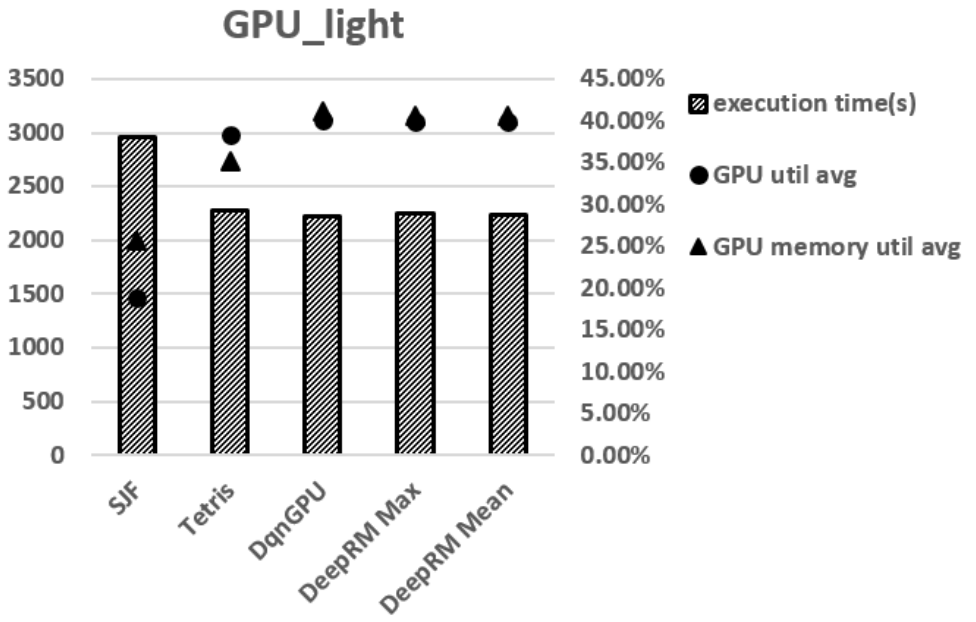
실행의 결과이다. SJF 방식으로 배치한 경우, 4807초 가장 짧은 수행 시간을 갖는다. 그러나 GPU 활용도 평균은 70.50%, GPU 메모리 평균에서는 36.44%로 가장 낮다. 다중 작업이 실행되는 기법 중 본 연구에서 제안하는 기법이 4891초로 가장 높은 성능을 보이며, GPU 활용도 82.09%, 59.13%로 높은 자원 활용을 보인다. DeepRM Mean의 경우에는 수행 시간이 5530초로 가장 길며 이는 GPU 메모리가 최대로 쓸 때 메모리 문제(OOM: Out Of Memory)가 발생하여 실패한 응용을 다시 실행하였기 때문이다.



<그림 10> GPU_heavy 워크로드의 실행 결과

GPU_light 워크로드는 평균 40% 미만의 GPU 활용도를 보이는 alexnet, GROMACS 응용으로 구성되었다. 그림 11은 GPU_light

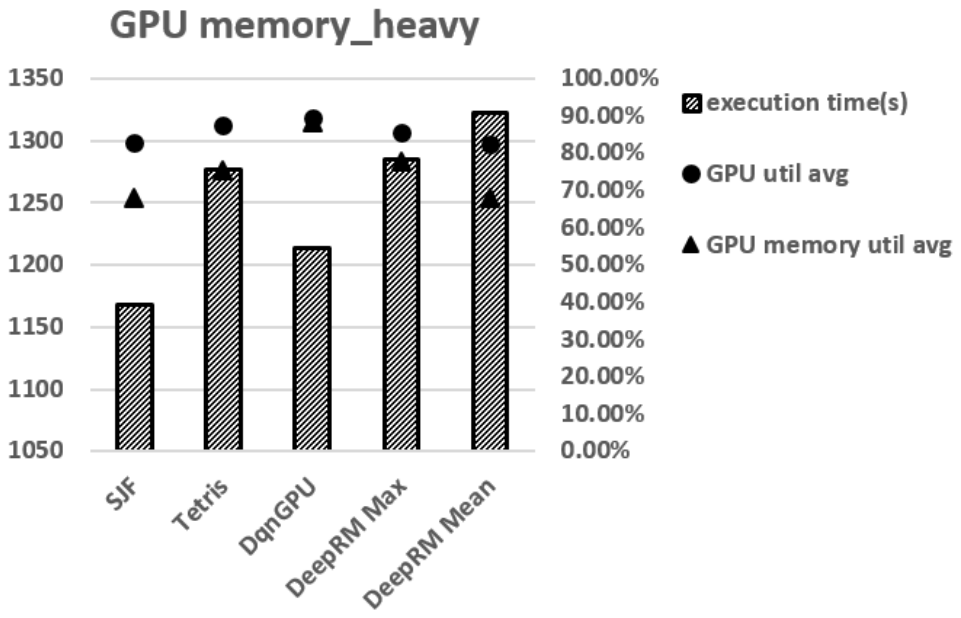
워크로드의 실행 결과를 보여준다. GPU 활용도가 낮은 응용들을 실행한 경우, SJF 방식의 배치가 2295초로 가장 낮은 성능과 자원 활용도를 가진다. 이는 SJF 방식으로 응용이 단일 배치 되어 공동 수행이 가능한 다른 기법에 비해 수행 시간이 늘어난 것으로 보인다. 응용 GROMACS의 경우 alexnet에 비해 수행시간이 길지만 GPU 메모리를 적게 사용하기 때문에 한번 배치 시 최대 3개의 작업이 수행 가능하기 때문이다. 본 연구에서 제안한 배치 기법이 1214초, GPU 활용도 89.31%, GPU 메모리 활용도 88.42%로 성능이 가장 좋으며, 강화학습을 통한 배치로 GPU 메모리를 최대 11755MB까지 사용할 수 있다.



<그림 11> GPU_light 워크로드의 실행 결과

GPU memory_heavy 워크로드는 평균 50% 이상의 GPU 메모리

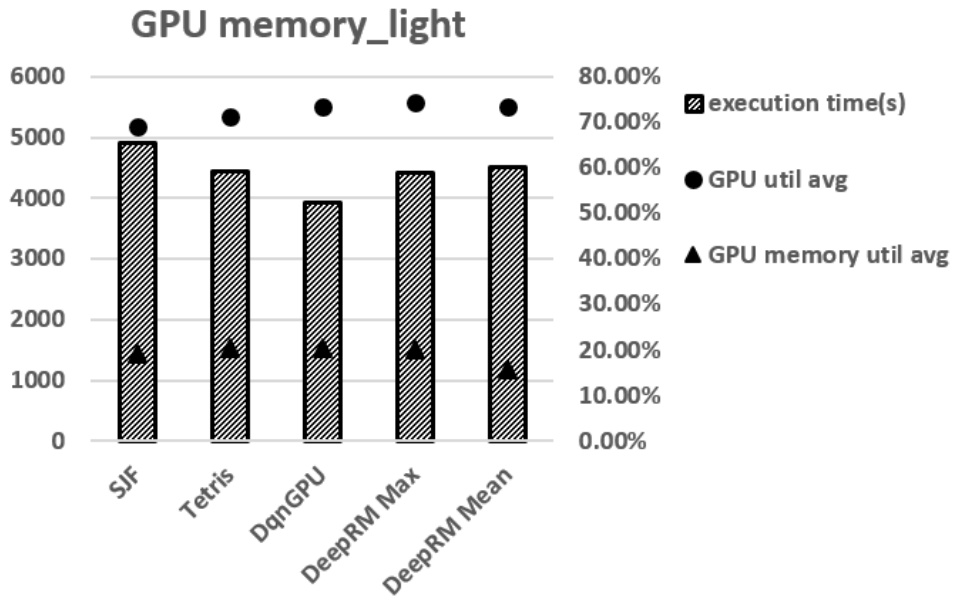
활용도를 보이는 vgg11, vgg16, resnet50, resnet101, resnet152 응용으로 구성되었다. 그림 12는 GPU memory_heavy 워크로드의 실행 결과를 나타낸다. SJF의 경우 수행 시간이 가장 짧으며, 그 다음으로 DqnGPU의 수행 시간이다. 수행 시간이 늘어난 이유는 실행한 응용들이 GPU 활용도도 높아 공동 배치되었을 경우 자원 경쟁을 유발시켰을 것으로 보인다. 그러나 다른 공동 수행 배치 기법과 비교하였을 때 1214초로 가장 짧은 수행 시간과 89.31%, 88.42%로 가장 높은 자원 활용도를 가지는 것을 알 수 있다.



<그림 12> GPU memory_heavy 워크로드의 실행 결과

GPU memory_light 워크로드는 평균 50% 미만의 GPU 메모리 활용도를 보이는 alexnet, googlenet, QMCPACK, HOOMD, GROMACS

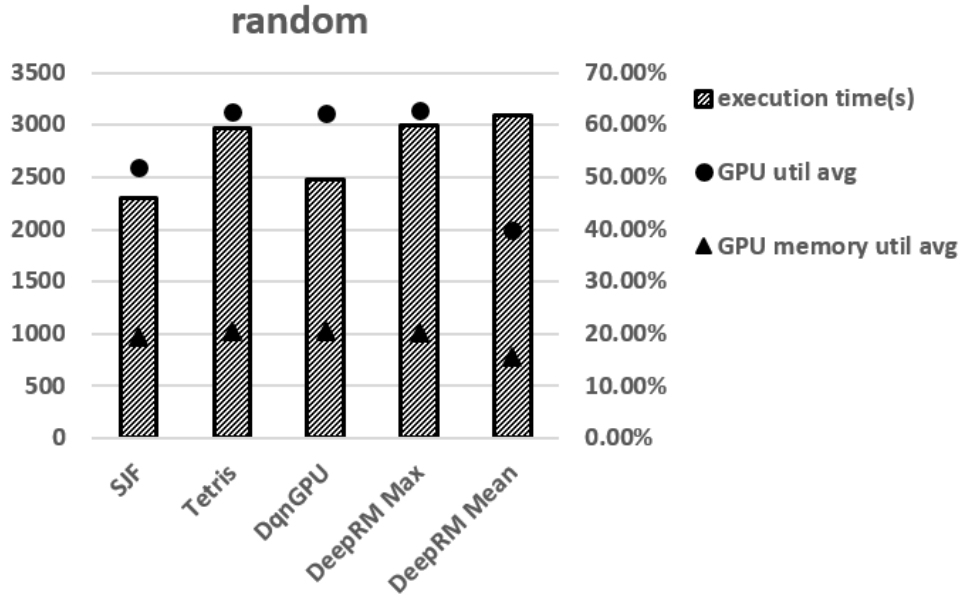
응용으로 구성되었다. 그림 13은 GPU memory_light 워크로드의 실행 결과를 나타내며, 본 연구에서 제안하는 기법이 수행 시간 측면에서 가장 우수한 것을 알 수 있다. 3928초로 가장 짧은 수행 시간을 가지며 이 때 학습 결과를 통해 GPU 메모리 활용도는 낮으나 수행시간이 긴 GROMACS, QMCPACK 응용들이 공동 배치되어 수행되었기 때문이다.



<그림 13> GPU memory_light 워크로드의 실행 결과

Random 워크로드는 실험에 사용한 모든 응용들로 구성하였다. 그림 14는 random 워크로드를 수행했을 때의 결과를 보여준다. 본 연구에서 제안하는 기법으로 수행하였을 경우 2481초, GPU 활용도 62.16%, GPU 메모리 활용도 20.45%의 성능을 보인다. 수행 시간 측면에서 다른 공동 실행 배치 기법보다 좋은 성능을 보인다. 그러나

GPU 활용도는 Tetris, DeepRM Max 기법보다 다른 워크로드 실험과 비교 하였을 때 낮으며, SJF와 비교하였을 때 GPU 메모리 활용도도 비슷한 것을 볼 수 있다. 여러 응용이 존재할 때 다양한 자원 사용 특성이 존재하기 때문에 공동 수행 시 자원 경쟁의 영향이 다양하게 반영될 수 있음을 알 수 있다.

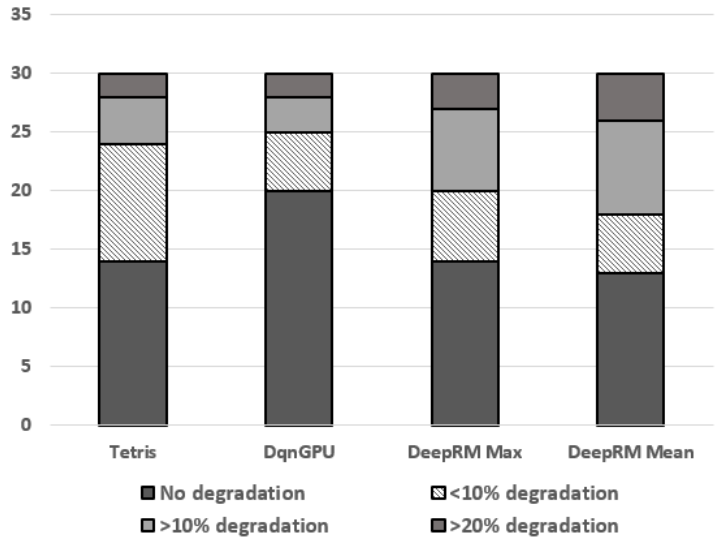


<그림 14> Random워크로드의 실행 결과

2) 작업 속도 저하 분석

위의 실험에서 사용한 random 워크로드를 사용하여 성능 평가를 위해 공동 수행 배치 기법 4개에 대한 총 30개 작업의 속도 저하를

분석하였다. 단일 실행한 응용의 수행 시간을 기준으로 속도 저하 없음(오차 2%), 속도 저하 10%미만, 10% 이상 20% 미만, 20% 이상으로 나누어 평가하였다.



<그림 15> 4개 기법에 대한 30개 작업의 속도 저하

그림 15는 random 워크로드 실행 시 4개 기법에 대한 각 작업의 속도 저하를 보여준다. 본 연구에서 제안하는 기법은 총 20개의 작업이 속도 저하가 발생하지 않았다. 수행 시간 별 변하는 자원의 사용량에 따라 학습하기 때문에 실제 자원을 초과하지 않는 범위 내에서 배치하여 자원 경쟁으로 발생하는 속도 저하가 작은 것으로 보인다. 그러나 발생하는 속도 저하는 공동 수행 시 고려하지 않은 자원 외에도 실행하는 응용의 특성, 다양한 조건이나 환경에서 영향을 받을 수 있음을 의미한다. Tetris, DeepRM Max 기법의 경우 14개의 작업에서

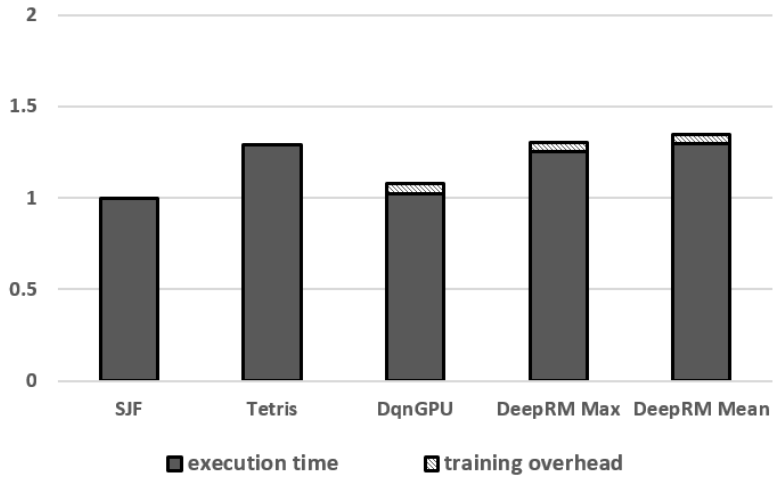
속도 저하가 발생하지 않았다. DeepRM Mean의 경우 속도 저하 10%이상에서 총 12개로 가장 많이 해당된다. DeepRM Mean의 경우 자원 평균 값에 따라 배치하기 때문에 공동 실행 시 발생하는 OOM 문제로 작업 속도 저하가 많이 발생하였다.

3) 트레이닝 오버헤드 비교

전체 수행 시간에 트레이닝 시간이 걸리는 오버헤드를 평가하기 위해 random 워크로드를 사용하여 비교하였다. 전체 수행 시간은 수행 시간이 가장 짧은 SJF를 기준으로 정규화 하여 비교한다. 전체 응용의 자원 사용 이력은 존재한다고 가정할 때, 처음 작업 배치를 위한 트레이닝 시간을 포함한 전체 수행 시간을 비교한다. DqnGPU, DeepRM Max, DeepRM Mean 기법의 경우 강화 학습을 통해 나온 결과에 따라 배치하기 때문에 트레이닝 시간이 포함된다.

그림 16은 각 배치 기법의 트레이닝 오버헤드의 결과를 보여준다. 수행 시간 정규화 값은 Tetris, 본 연구에서 제안하는 기법, DeepRM Max, DeepRM Mean 각각 약 1.294, 1.081, 1.303, 1.346이다. 제안하는 기법은 트레이닝 시간이 123초로 약 0.054이며 트레이닝 시간을 뺀 수행 시간은 1.027이다. SJF 와 수행 시간의 결과를 비교하였을 때 성능 차이가 작은 것을 확인할 수 있다. 이는 작업 셋의 처음 트레이닝 시간을 제외하고 그 후 다음 작업을 배치할 경우에는 성능 차이가 거의 없다는 것을 의미한다. 자원 활용도도 높아질 뿐만 아니라 작업 수행하면서 온라인으로 트레이닝을 계속 진행한다면 정확도도 높아져 다음 작업 셋 수행 성능이 좋아질 수 있음을 의미한다.

DeepRM Max의 트레이닝 시간은 0.048이며 Tetris와 DeepRM Max의 수행 시간만 비교 했을 시에 1.294, 1.254로 DeepRM Max의 경우가 더 우수하다. 따라서 처음 트레이닝 오버헤드를 제외하고 강화 학습을 적용한 작업 배치 기법이 의미가 있다.



<그림 16> 트레이닝 오버헤드 비교

4) 단독 배치와 공유 배치 가격 비교

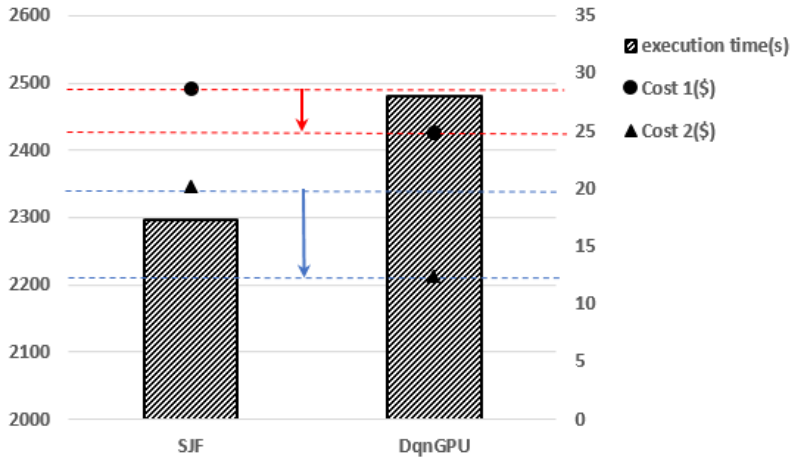
워크로드나 실행하는 응용에 따라 단독 배치하여 실행할 때의 성능이 좋을 수 있다. 그러나 앞의 실험을 통해서 공유 배치하는 것이 모든 워크로드에서 자원 활용도 측면에서 우수함을 확인하였다. 이는 클러스터 운영에 있어서 이점이 있음을 의미한다. 사용자 측면에서 자원의 공유 배치의 장점을 설명하기 위해 클라우드 업체에서 제공하는 GPU 인스턴스를 기준으로 SJF와 DQNPGPU의 가격을 비교하였다.

GPU 공유 인스턴스에 대한 가격이 존재하지 않으므로 현재 제공되는 비슷한 GPU 카드에 따라 인스턴스 가격을 비교하였다. 표 4는 Amazon EC2에서 제공하는 GPU 인스턴스 가격이다. G3 인스턴스는 NVIDIA Tesla M60, G4 인스턴스는 NVIDIA T4 GPU 카드를 제공한다. GPU 메모리 자원 측면(Cost 1)에서 g4dn.4xlarge 는 g3.xlarge 2개와 같다고 가정하면, g4dn.4xlarge를 공유하였을 때 0.602$까지 절감할 수 있다. 또한 vCPU, 메모리 자원 측면(Cost 2)에서 g4dn.4xlarge가 g4dn.xlarge 4개와 같다고 가정하면, g4dn.4xlarge를 공유하여 사용하였을 때 0.301$까지 가격 절감이 가능하다.

<표 5> Amazon EC2 GPU 인스턴스 가격

Name	GPU	vCPU	메모리 (GB)	GPU 메모리 (GB)	GPU cores	Cost($)
g3.xlarge	1	4	30.5	8	2,048	0.75
g4dn.xlarge	1	4	16	16	2,560	0.526
g4dn.4xlarge	1	16	64	16	2,560	1.204

앞의 random 워크로드를 실행했을 때를 기준으로 위에서 가정한 Cost에 따라 가격을 비교하였다. 그림 17은 SJF와 DQNGPU의 가격을 비교한 것이다. Cost 1의 경우, 공유 배치하는 인스턴스를 사용하였을 경우, 단독 인스턴스를 사용했을 때 보다 약 29.86% 정도 비용 절감이 가능하다. Cost 2의 경우 역시 단독 인스턴스를 사용했을 때 보다 공유하였을 때 약 49.98% 정도 비용을 절감할 수 있다.



<그림 17> SJF와 DqnGPU 가격 비교

VI. 결론

본 연구에서는 GPU 클러스터 환경에서 응용의 작업 이력을 반영한 강화 학습 DQN 기반 작업 배치 기법을 제안한다. 강화 학습은 변화하는 자원 사용을 포함하는 작업 이력이 반영된 DNN을 통해 학습하며 다중 작업 배치 행동 값을 얻는다. 제안하는 데이터 배치 기법을 기반으로 실제 다양한 특성을 가지는 응용을 대상으로 공동 배치 및 자원을 공유하도록 수행하였다. 특성을 분류한 워크로드를 생성하여 비교 대상 기법인 단일 배치, 다중 배치, 강화 학습을 적용한 다중 배치 기법과 비교하였다. 실험을 통해 제안하는 기법이 낮은 성능 저하와 자원 활용도를 가짐을 보임으로써 의미 있음을 보였다. 또한 발생할 수 있는 트레이닝 오버헤드를 분석하여 전체 성능에 끼치는 영향이 작음을 증명하였다. 또한 단독 배치, 공유 배치 시의 GPU 자원 가격 비교를 하여 GPU 공유 배치의 이점을 설명하였다. 향후 연구로는 클러스터 자원 환경을 추가하고, 실행 하는 응용의 자원 환경을 변경하여 수행하여 변경된 작업 이력과 강화 학습과의 적용에 대해 고려할 것이다. 또한 처음 실행 시 발생하는 트레이닝 오버헤드를 줄이고자 처음 배치를 위한 오프라인 작업 배치가 포함된 기법으로 확장하고자 한다.

참 고 문 헌

[1] Amazon EC2, https://aws.amazon.com/ec2/

[2]Nimbix. https://www.nimbix.net/cloud-computing-nvidia/

[3]Microsoft Azure, https://docs.microsoft.com/en-au/azure/virtual-machines/windows/sizes-gpu

[4]Alibaba, https://www.alibabacloud.com/ko/product/gpu

[5]Kubernetes, https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/

[6]Mesos, http://mesos.apache.org/documentation/latest/gpu-support/

[7]Liu, Ming, et al. "Understanding the virtualization" Tax" of scale-out pass-through GPUs in GaaS clouds: An empirical study." 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2015.

[8]MPS, https://docs.nvidia.com/deploy/mps/index.html

[9]Chang, Chia-Chen, et al. "A kubernetes-based monitoring platform for dynamic cloud resource provisioning." GLOBECOM 2017-2017 IEEE Global Communications Conference. IEEE, 2017.

[10]Gu, Jing, et al. "GaiaGPU: Sharing GPUs in Container Clouds." 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, 2018.

[11]Song, Shengbo, et al. "Gaia Scheduler: A Kubernetes-Based Scheduler Framework." 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, 2018.

[12]Hong, Cheol-Ho, Ivor Spence, and Dimitrios S. Nikolopoulos. "FairGV: fair and fast GPU virtualization." IEEE Transactions on Parallel and Distributed Systems 28.12 (2017): 3472-3485.

[13]I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, and M. Valero. Enabling preemptive multiprogramming on gpus. In (ISCA-14), Minneapolis, USA, 2014.

[14]Y. Ukidave, C. Kalra, D. Kaeli, P. Mistry, and D. Schaa. Runtime support for adaptive spatial partitioning and inter-kernel communication on gpus. In (SBAC-PAD),2014, pages 168-175. IEEE, 2014.

[15]Chang, Chia-Chen, et al. "A kubernetes-based monitoring platform for dynamic cloud resource provisioning." GLOBECOM 2017-2017 IEEE Global Communications Conference. IEEE, 2017.

[16]Diab, Khaled M., M. Mustafa Rafique, and Mohamed Hefeeda. "Dynamic sharing of GPUs in cloud systems." 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. IEEE, 2013.

[17]Gu, Jing, et al. "GaiaGPU: Sharing GPUs in Container Clouds." 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, 2018.

[18]Song, Shengbo, et al. "Gaia Scheduler: A Kubernetes-Based Scheduler Framework." 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, 2018.

[19]Julia Lawall, Sasha Levin, "Building Stable Kernel Trees with Machine Learning", Linux Plumbers Conference 2018

[20]2019 USENIX Conference on Operational Machine Learning, https://www.usenix.org/conference/opml19

[21]Rossi, Fabiana, Matteo Nardelli, and Valeria Cardellini. "Horizontal and vertical scaling of container-based applications using reinforcement learning." 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). IEEE, 2019.

[22]Mao, Hongzi, et al. "Resource management with deep reinforcement learning." Proceedings of the 15th ACM Workshop on Hot Topics in Networks. ACM, 2016.

[23]Bao, Yixin, Yanghua Peng, and Chuan Wu. "Deep Learning-based Job Placement in Distributed Machine Learning Clusters." IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE, 2019.

[24]Xu, Xin, et al. "Characterization and prediction of performance interference on mediated passthrough GPUs for interference-aware scheduler." 11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19). 2019.

[25]Ukidave, Yash, Xiangyu Li, and David Kaeli. "Mystic: Predictive scheduling for gpu based cloud servers using machine learning." 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2016.

[26]Li, Xiaqing, et al. "Performance analysis of gpu-based convolutional neural networks." 2016 45th International Conference on Parallel Processing (ICPP). IEEE, 2016.

[27]Abadi, Martín, et al. "Tensorflow: A system for large-scale machine learning." 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). 2016.

[28]LAMMPS, https://lammps.sandia.gov/

[29]QMCPACK, https://qmcpack.org/

[30]Phull, Rajat, et al. "Interference-driven resource management for GPU-based heterogeneous clusters." Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing. ACM, 2012.

[31]NVIDIA GPU Cloud, https://ngc.nvidia.com/

[32]Dutreilh, Xavier, et al. "Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow." ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems. 2011.

[33]Barrett, Enda, Enda Howley, and Jim Duggan. "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud." Concurrency and Computation: Practice and Experience 25.12 (2013): 1656-1674.

[34]Reinforcement Learning : an introduction, 2nd 3d. Cambridge

[35]Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[36]deepRM, https://github.com/hongzimao/deeprm

[37]R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In NIPS, 1999.

[38]GROMACS, http://www.gromacs.org/

[39]Hoomd, http://glotzerlab.engin.umich.edu/hoomd-blue/

[40]R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. SIGCOMM '14, pages 455–466, New York, NY, USA, 2014. ACM.

[41]Alibaba fake GPU, https://github.com/AliyunContainerService/gpushare-scheduler-extender

[42]Nvidia docker 컨테이너, https://github.com/NVIDIA/nvidia-docker

[43] Duato, José, et al. "rCUDA: Reducing the number of GPU-based accelerators in high performance clusters." *2010 International Conference on High Performance Computing & Simulation*. IEEE, 2010.

[44] Ilager, Shashikant, et al. "GPU PaaS Computation Model in Aneka Cloud Computing Environments." *Smart Data: State-of-the-Art Perspectives in Computing and Applications* (2019): 19.

[45] Toosi, Adel Nadjaran, Richard O. Sinnott, and Rajkumar Buyya.

"Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using Aneka." *Future Generation Computer Systems* 79 (2018): 765-775.

[46] Tang, Xiongchao, et al. "Spread-n-share: improving application performance and cluster throughput with resource-aware job placement." *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2019.

ABSTRACT

A Job Placement Using Reinforcement Learning in GPU Virtualization Environment

Jisun Oh
Department of Computer Science
The Graduate School
Sookmyung Women's University

Graphics Processing Units (GPU) are widely used for high-speed processes in the computational science areas of biology, chemistry, meteorology, etc. and the machine learning areas of image and video analysis. Recently, data centers and cloud companies have adopted GPUs to provide them as computing resources. Because the majority of cloud providers allocate the GPU resource to users in an exclusive access method, the allocated GPU resource may not be all used. Although the method of allocating a GPU resource to multiple users for sharing can increase the resource utilization, performance

degradation may occur in individual jobs because of interference between different jobs. It is difficult for a cloud provider to predict or control the performance of various applications executed on various cloud resources by considering their characteristics heuristically. Therefore, an intelligent job placement technique is required to minimize the interference between different jobs and increase resource utilization.

This study defines the resource utilization history of applications and proposes a reinforcement learning-based job placement technique, which uses it as an input. For resource utilization history learning, a deep reinforcement learning model (DQN) is used. As a result of learning, the current resource's state is not exceeded, and the resource is still provided by predicting which commonly placed jobs will have less impact on the total performance when executed simultaneously. This approach prevents the performance degradation of applications with diverse execution characteristics and increases the resource utilization by executing the applications while sharing the resources. The superiority of this study is demonstrated by using the proposed learning method and other methods to analyze workloads with various resource utilization characteristics. Through the experiments, it is proven that the proposed method facilitates a reduction of the total execution time and the effective use of resources, while the maintaining performance.

Key words: GPU, Application Job History, DQN Learning, Interference Prediction, Multi-job Placement