Punica: 다중 테넌트 LoRA 서비스 시스템

Lequn Chen [*1] Zihao Ye [*1] Yongji Wu [*2] Danyang Zhuo [2] Luis Ceze [1] Arvind Krishnamurthy [1]

초록

저랭크 적응(Low-rank adaptation, LoRA)은 사전 훈련된 모델들을 특정 도메인에 맞게 적용하는 중요하고 인기 있는 방법이 되었습니다. 우리는 공유 GPU 클러스터에서 여러 LoRA 모델들을 운영할 수 있는 시스템인 Punica를 제시합니다. Punica는 다양한 LoRA 모델들의 GPU 연산을 배치 처리할 수 있는 새로운 CUDA 커널 설계를 포함하고 있습니다. 이를 통해 GPU는 여러 개의 서로 다른 LoRA 모델들을 서비스하면서 기본 pre-trained 모델 하나만을 유지할 수 있으며, GPU 효율성을 메모리 및 컴퓨팅 양면에서 크게 향상시킵니다. 우리의 스케줄러는 공유 GPU 클러스터에서 다중 테넌트 LoRA 서비스 워크로드를 통합합니다. 고정된 크기의 GPU 클러스터에서, 우리의 평가는 Punica가 토큰당 2 ms의 지연 시간을 추가하는 동안 최신 LLM 서비스 시스템과 비교하여 여러 LoRA 모델 서비스에서 12배 높은 처리량을 달성한다는 것을 보여줍니다. Punica는 오픈 소스이며 https://github.com/punica-ai/punica 에서 사용할 수 있습니다.

1 서론

저랭크 적응(Low-rank adaptation, LoRA) (Hu et al., 2022)은 최소한의 훈련 데이터를 사용하여 사전 훈련된 대규모 언어 모델(Large Language Models, LLMs)을 특정 도메인 작업에 특화시키는 데 점점 더 인기를 끌고 있습니다. LoRA는 사전 훈련 모델의 가중치를 유지하고, Transformer 구조의 각 층에 훈련 가능한 랭크 분해 행렬을 도입하여 훈련해야 할 매개변수 수를 대폭 줄이고, 테넌트들이 다양한 LoRA 모델들을 낮은 비용으로 훈련할 수 있게 합니다. LoRA는 많은 인기 있는 파인튜닝 프레임워크들(Mangrulkar et al., 2022)에 통합되었습니다. 그 결과, 기계학습(ML) 제공업체들은 그들의 테넌트들을 위해 다양한 전문 LoRA 모델들을 동시에 서비스해야 합니다.

흔히 LoRA 모델들을 처음부터 독립적으로 훈련된 것처럼 서비스하는 것은 GPU 자원을 낭비합니다. 각 LoRA 모델을 서비스하기 위해 k개의 GPU가 필요하다고 가정할 경우, n개의 서로 다른 LoRA 모델을 서비스하는 것은 $k \times n$개의 GPU가 필요할 것처럼 보입니다. 이 직관적인 접근 방식은 이러한 LoRA 모델들이 동일한 사전 훈련 모델에서 유래했다는 점을 고려할 때, 가중치 사이의 잠재적 연관성을 간과합니다.

우리는 효율적인 시스템이 여러 개의 서로 다른 LoRA 모델들을 서비스하기 위해 세 가지 설계 지침을 따라야 한다고 믿습니다. (G1) GPU는 비싸고 희귀한 자원이므로, 전반적인 GPU 이용률을 높이기 위해 다중 테넌트 LoRA 워크로드들을 적은 수의 GPU에 통합할 필요가 있습니다. (G2) Yu et al., 2022가 이미 지적한 바와 같이, 배치 처리는 성능과 GPU 이용률을 향상시키기 위해 ML 워크로드를 통합하는 가장 효과적인 접근 방법 중 하나입니다. 그러나 배치 처리는 동일한 모델에 대한 요청이 있을 때만 작동합니다. 따라서 우리는 서로 다른 LoRA 모델들에 대해 배치 처리를 가능하게 할 필요가 있습니다. (G3) 디코드 단계는 모델 서비스 비용의 가장 큰 요소입니다. 그러므로 우리는 디코드 단계의 성능에만 주력할 필요가 있습니다. 모델 서비스의 다른 측면들은 그다지 중요하지 않으며, 예를 들어 LoRA 모델 가중치의 온디맨드 로딩과 같은 간단한 기술을 적용할 수 있습니다.

이 세 가지 지침에 근거하여, 우리는 공유 GPU 클러스터에서 LoRA 모델들을 위한 다중 테넌트 서비스 프레임워크인 Punica를 설계하고 구현합니다. 주요한 새로움은 새로운 CUDA 커널인 'Segmented Gather Matrix-Vector Multiplication (SGMV)' 설계입니다. SGMV는 다수의 서로 다른 LoRA 모델들의 동시 실행을 위한 GPU 연산을 배치 처리할 수 있게 합니다. SGMV 덕분에 GPU는 메모리에 사

전 훈련 모델의 단일 복사본만 저장하면 되므로 메모리 및 컴퓨팅 측면에서 GPU 효율성을 크게 향상시킵니다. 우리는 이 새로운 CUDA 커널을 최신 시스템 최적화 기술들과 결합합니다.

SGMV는 서로 다른 LoRA 모델들로부터의 요청을 배치 처리할 수 있으며, 놀랍게도 우리는 동일한 LoRA 모델들을 배치 처리할 때와 서로 다른 LoRA 모델들을 배치 처리할 때 성능 차이가 거의 없음을 관찰합니다. 동시에, LoRA 모델들의 온디맨드 로딩은 밀리초 수준의 지연 시간만을 가집니다. 이는 Punica에게 GPU상에서 이미 실행 중인 LoRA 모델들에 구애받지 않고 사용자 요청들을 적은 수의 GPU에 통합할 유연성을 제공합니다.

Punica는 이에 따라 다음과 같은 두 가지 방식으로 다중 테넌트 워크로드를 스케줄합니다. 새로운 요청에 대해, Punica는 요청을 활성화된 소수의 GPU로 라우팅하여 그들이 전체 용량에 도달하도록 합니다. 기존의 GPU가 완전히 활용될 때에만, Punica는 추가적인 GPU자원할당합니다. 이미 존재하는 요청들에 대해서는, Punica는 주기적으로 그들을 이주시킵니다

그림 1. Prefill 단계와 Decode 단계에서의 배치 영향

병합(consolidation). 이를 통해 Punica에 할당된 GPU 자원을 해제할 수 있습니다.

우리는 NVIDIA A100 GPU 클러스터에서 Llama2 7B, 13B, 70B 모델에서 조정된 LoRA 모델들을 평가합니다(Touvron 외, 2023). 같은 양의 GPU 자원을 주어졌을 때, Punica는 최첨단 대형 언어모델(Large Language Model, LLM) 서버 시스템들과 비교하여 토큰당 2 ms의 지연을 추가하는 대신 $12x$ 더 높은 처리량을 달성합니다.

이 논문은 다음과 같은 기여를 합니다:

- 다양한 LoRA 모델들의 요청을 배치 처리하는 기회를 식별합니다.
- 여러 LoRA 모델들을 동시에 실행하는 효율적인 CUDA 커널을 설계하고 구현합니다.
- 다중 임차인(Tenant) LoRA 작업 부하를 통합하는 새로운 스케줄링 메커니즘을 개발합니다.

2 배경(BACKGROUND)

먼저, 트랜스포머(Transformer) 모델을 위한 텍스트 생성 과정을 소개합니다. 그 다음에는 트랜스포머 모델의 저랭크 적응(Low-Rank Adaptation, LoRA)에 대해 설명합니다.

2.1 트랜스포머와 텍스트 생성(Transformer and Text Generation)

트랜스포머 기반 대형 언어모델(LLM)은 토큰(token)의 시퀀스를 다룹니다. 토큰은 대략 영어 단어의 3/4에 해당합니다. LLM의 작동은 두 단계로 구성됩니다. Prefill 단계는 사용자 프롬프트(prompt)를 받아서 다음 토큰을 생성하고 키-값 캐시(Key-Value cache, KvCache)를 생성합니다. Decode 단계는 토큰과 KvCache를 받아서 또 하나의 토큰을 생성하고 KvCache에 열(column)을 추가합니다. Decode 단계는 반복적인 과정입니다. 생성된 토큰은 다음 단계의 입력이 됩니다. 이 과정은 시퀀스 종료 토큰이 생성될 때까지 끝나지 않습니다.

트랜스포머 블록은 자기주의(self-attention) 층과 다중층 퍼셉트론(Multilayer Perceptron, MLP)을 포함합니다. 프롬프트의 길이를 s라 가정하고, 주의력(attention) 헤드 차원을 d라고 합시다. Prefill 단계에 대해, 자기주의 층의 계산은 $(s,d) \times (d,s) \times (s,d)$이며, MLP의 계산은 $(s,h) \times (h,h)$입니다. Decode 단계의 경우, s가 과거 시퀀스 길이를 나타낸다고 가정할 때, 자기주의 층의 계산은 $(1,d) \times (d,s+1) \times (s+1,d)$이고, MLP의 계산은 $(1,h) \times (h,h)$입니다. Decode 단계는 입력이 단일 벡터(single vector)이기 때문에 낮은 GPU 사용률을 가집니다.

그림 1은 서로 다른 배치 크기에 대한 prefill 단계와 decode 단계의 지연시간을 보여줍니다. GPU의 계산 능력은 prefill 단계 동안 완전히 사용됩니다. Prefill 지연은 배치 크기에 비례합니다. 하지만 decode 단계에서는 그렇지 않습니다. 배치 크기를 1에서 32로 증가시킬 때, 짧은 시퀀스의 경우

decode 단계 지연은 11 ms에서 13 ms로, 긴 시퀀스의 경우 17 ms에서 34 ms로 증가합니다. 이는 배치 처리가 decode 단계에 대한 GPU 사용률을 크게 개선할 수 있는 기회임을 의미합니다. Orca(Yu 외, 2022)는 이 기회를 활용하여 효율적인 대형 언어모델 서비스 시스템을 구축했습니다. 이러한 배치 처리는 decode 단계가 긴 출력 길이 응답에 대한 서버 지연시간을 주로 결정하기 때문에 특히 중요합니다.

2.2 저랭크 적응(Low-Rank Adaptation, LoRA)

미세 조정(Fine-tuning)을 통해 사전 학습된 모델을 새로운 영역이나 작업에 적응시키거나 새로운 학습 데이터로 개선할 수 있습니다. 그러나 대형 언어모델(LLM)은 크기가 크므로 모든 모델 매개변수를 미세 조정하는 것은 자원이 많이 듭니다.

저랭크 적응(Low-Rank Adaptation, LoRA)(Hu et al., 2022)은 미세조정(fine-tuning) 동안에 필요한 파라미터의 수를 크게 줄이는 것을 가능하게 만듭니다. 주요한 관찰 결과는, 사전 학습된 모델(pre-trained model)과 미세조정된 모델 사이의 가중치 차이가 저랭크(low rank)임을 발견한 것입니다. 이 가중치 차이는 두 개의 작고 밀집된 행렬의 곱으로 표현될 수 있습니다. 그래서 LoRA 미세조정은 작고 밀집된 신경망을 학습하는 것과 비슷해집니다. 형식적으로, 사전 학습된 모델의 가중치를 $W \in \mathbb{R}^{h_1 \times h_2}$라고 가정합시다. LoRA 미세조정은 두 행렬 $A \in \mathbb{R}^{h_1 \times r}$와 $B \in \mathbb{R}^{r \times h_2}$를 학습합니다. 여기서 r은 LoRA 랭크(LoRA Rank)입니다. $W + AB$는 미세조정된 모델을 위한 새로운 가중치입니다. LoRA 랭크는 보통 원래 차원보다 훨씬 작습니다(예를 들어, 4096 대신에 16). 빠른 미세조정 외에도, LoRA는 매우 낮은 스토리지와 메모리 부담을 가집니다. 각 미세조정된 모델은 모델 가중치에 겨우 0.1%에서 1%만큼을 추가합니다. LoRA는 보통 변환기 계층(transformer layer)(Dettmers et al., 2023) 내의 모든 밀집된 투영에 적용됩니다. 여기에는 주의 메커니즘(attention mechanism) 내의 질의-키-값-출력 투영과 다층 퍼셉트론(MLP)이 포함됩니다. 자기주의 연산(self-attention operation) 자체는 어떠한 가중치도 포함하지 않는다는 점을 주목하세요.

공유된 GPU 클러스터에서는 어떻게 다중 테넌트 LoRA 모델을 효율적으로 운영할까요? LoRA는 대규모 언어 모델(LLM)을 미세조정하기 위한 효율적인 알고리즘을 제공합니다. 이제 질문은, 이러한 LoRA 모델들을 효율적으로 어떻게 서비스할까 입니다. 한 가지 접근 방법은 각 LoRA 모델을 독립된 모델로 간주하고 전통적인 LLM 서버 시스템(예를 들어, vLLM)을 사용하는 것입니다. 하지만, 이는 서로 다른 LoRA 모델들 사이의 가중치 공유를 간과합니다.

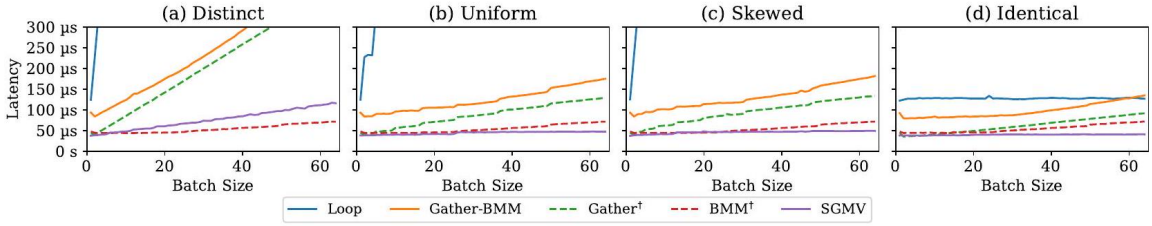

그림 8. LoRA 연산자 구현에 대한 마이크로벤치마크. † 참고를 위해 Gather와 BMM은 각각 별도로 측정됩니다.

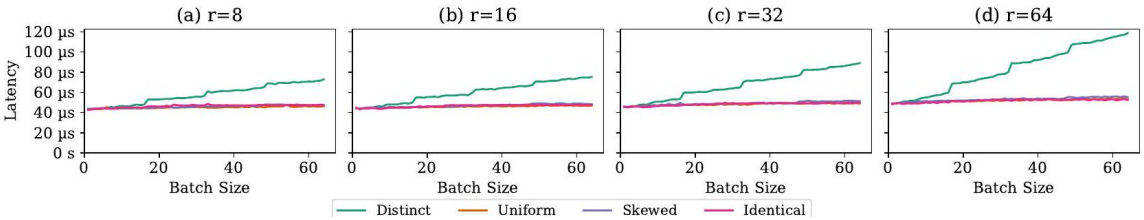

그림 9. 다양한 LoRA 랭크에 대한 LoRA 연산자 마이크로벤치마크.

전반적으로 SGMV는 작업 부하에 상관없이 기본 구현을 크게 능가합니다.

우리는 또한 Testbed #1에서 다른 LoRA 랭크에 대한 마이크로벤치마크를 실행합니다. 그림 9는 LoRA 랭크 8, 16, 32, 64의 지연 시간을 보여줍니다. Distinct 경우에서는 지연 시간이 점차 증가합니다. 단일 요청 배치에 대한 지연 시간은 네 가지 랭크에 대해 모두 약 42μ s인 반면, 배치 크기 64는 각각 72μs, 75μs, 89μs 및 118μs까지 올라갑니다. 가중치 공유가 있는 작업 부하(Uniform, Skewed, Identical)의 경우에는, 배치 크기 1에서 64까지 지연 시간이 거의 동일하며, 약 42μ s에서 45μs까지입니다.

변환기 계층 벤치마크 다음으로, LoRA 연산자를 통합한 후 변환기 계층의 성능을 평가하였습니다. 대규모 언어 모델(LLM)이 대략 변환기 계층의 쌓임으로 이루어져 있기 때문에, 계층 성능은 전체 모델 성능을 결정합니다. 우리는 7B와 13B 모델 구성과 512 및 2048의 시퀀스 길이를 기반으로 Testbed #1에서 계층 벤치마크를 실행합니다. 그림 10은 계층 지연 시간을 나타냅니다. 시퀀스 길이가 짧을수록 배치 효과가 더 강합니다. 시퀀스 길이가 512일 때, 배치 크기가 1에서 32로 증가하면 지연 시간은 단지 72% 증가합니다. 시퀀스가 더 길면, 자기주의 연산은 더 오래 걸리게 되며, 이는 계층별 배치 효과를 줄여줍니다.

커널 마이크로벤치마크와는 대조적으로, 계층 지연 시간은 다른 작업 부하에서 대략 같습니다. 이것은 LoRA 추가 기능의 계산 시간이 기본 밀집 투영과 자기주의 연산에 비해 작기 때문입니다. 이런 LoRA-모델-무관 성능 특성은 우리가 다양한 LoRA 모델을 마치 하나의 모델처럼 일정을 잡도록 하게 해주며, 우리의 일정 알고리즘은 그 때 개별 LoRA 모델 배치에 초점을 맞추는 대신 전반적인 처리량에 집중할 수 있게 합니다. 이것이 바로 우리가 Punica를 어떻게 설계했는지에 대한 것입니다.

7.2 텍스트 생성

다음으로, 우리는 Punica와 기준 시스템들의 텍스트 생성 성능을 연구합니다.

단일 GPU에서 7B와 13B 모델 서비스
Testbed #1에서 단일 GPU를 사용하여 Punica와 기준 시스템들의 텍스트 생성을 평가합니다. 단일 GPU 성능은 클러스터 전반의 배포를 위한 기본 사례로 쓰입니다. 우리는 1000개의 요청(약 101k 토큰을 생성)을 생성하고 각 시스템을 선착순 배치에 제한합니다. 최대 배치 크기는 모든 시스템에 대해 32로 설정됩니다. Punica는 다른 LoRA(LoRA) 모델 간에 배치를 수행할 수 있고, 기준 시스템들은 동일한 LoRA 모델에 대한 요청만 배치할 수 있습니다.

그림 11(a) 및 (b)는 각각 7B 모델과 13B 모델에서의 결과를 보여줍니다. Punica는 작업 부하에 관계없이 일관되게 높은 처리량을 제공합니다. Punica는 각각 7B 모델에서 1044tok/s, 그리고 13B 모델에서 693 tok/s의 성능을 달성합니다. 대부분의 기준 시스템들도 Identical(Identical) 경우에는 비교적 높은 처리량을 달성할 수 있지만, 여러 LoRA 모델이 있을 때 성능이 저하됩니다.

Distinct(Distinct) 경우에는, 모든 기준 시스템들이 배치 크기 1로 실행되므로 처리량이 낮습니다. Uniform(Uniform)과 Skewed(Skewed) 경우에는 기준 시스템들의 대부분의 배치들이 매우 작은 배치 크기(1-3)를 가지고 있어, 이것이