LLM 서빙에서 Multi-GPU 병렬화 성능 분석

최민석[01], 장성[1], 김기현[1], 김영재[†]
서강대학교 컴퓨터공학과
{minseok209, jesjsjes, kion777, youkim}@sogang.ac.kr

Analysis of Performance in Multi-GPU Parallelization for LLM Serving

Minseok Choi[01], Seong Jang[1], Kihyun Kim[1], Youngjae Kim[†]
Department of Computer Science and Engineering, Sogang University

요 약

대규모 언어 모델(LLM) serving에서는 여러 GPU를 사용하더라도 어떤 단위로 분할하느냐에 따라 성능 특성이 크게 달라진다. Tensor Parallelism(TP), Pipeline Parallelism(PP), Data Parallelism(DP), Prefill-Decode Disaggregation(PD)은 각각 tensor 연산, layer 구간, 요청, prefill/decode phase를 나누는 방식이며, 서로 다른 이득과 오버헤드를 가진다. 본 연구는 vLLM 기반 Llama-3.2-1B-Instruct serving 환경에서 TP=2, PP=2, DP=2, PD를 동일 조건으로 비교하고, synthetic 워크로드와 BurstGPT trace 워크로드[4]를 함께 사용해 입출력 길이 분포에 따라 각 방식의 성능이 어떻게 달라지는지 분석하였다. 실험 결과, 병렬화 방식의 우위는 워크로드와 성능 지표에 따라 달라졌다. PD는 decode-heavy에서 가장 높은 throughput을 보였으나 prefill-heavy에서는 prefill 측 병목으로 평균 TTFT가 크게 증가하였다. TP는 general에서 throughput과 first-token latency의 균형이 가장 좋았다. DP 실험에서는 긴 context와 높은 동시 요청 조건에서 throughput이 거의 증가하지 않는 반면 TTFT와 E2E latency가 증가하여, queueing delay와 KV cache pressure가 병목임을 확인하였다. GPU utilization 분석에서는 DP가 일부 조건에서 평균 처리량은 높더라도 GPU 활용률 불균형(최대 94%)으로 E2E latency가 커질 수 있음을 보였다. 따라서 본 연구는 LLM serving에서 입력 길이, 출력 길이, 요청 부하, TTFT/ITL 요구사항에 따라 병렬화 방식을 선택해야 함을 보인다.

1. 서 론

대규모 언어 모델(LLM)의 활용이 증가하면서 LLM 추론 요청을 효율적으로 처리하는 serving 시스템의 중요성이 커지고 있다. LLM serving에서는 단일 요청의 latency뿐 아니라 다수의 요청을 일정 시간 안에 처리하는 throughput도 중요하다. 특히 LLM은 모델 가중치와 KV cache가 GPU memory를 크게 차지하므로 여러 GPU를 어떻게 활용하는지가 전체 serving 성능에 직접적인 영향을 준다.

LLM 서빙을 최적화 하기 위해서는 다양한 병렬화 방식이 존재한다. 병렬화 방식으로는 기존 Multi-GPU serving에서는 Tensor Parallelism(TP), Pipeline Parallelism(PP), Data Parallelism(DP)이 사용된다. 최근에는 prefill과 decode의 서로 다른 특성을 이용하기 위해 Prefill-Decode Disaggregation(PD)이 제안되었다.

기존 연구들은 이러한 병렬화 방식의 일부 trade-off를 분석해왔다. DistServe[1]와 Splitwise[2]는 prefill과 decode의 자원 특성이 다르며 두 phase를 분리하면 TTFT와 TPOT/ITL을 독립적으로 다룰 수 있음을 보였다. Seesaw[3]는 prefill과 decode가 서로 다른 병렬화 전략을 선호할 수 있음을 분석하였다. 그러나 실제 serving 선택 관점에서 볼 때, 같은 워크로드 조건에서 각 병렬화 방식을 독립적으로 실행하여 비교하고, 어떤 워크로드에서 어느 방식이 유리하며 그 차이가 어떤 구조적 원인에서 비롯되는지 분석한 연구는 충분하지 않다.

본 연구는 이 문제를 다루기 위해 vLLM 기반 단일 노드 2-GPU serving 환경에서 TP=2, PP=2, DP=2, PD를 동일 조건으로 비교한다. 워크로드는 decode-heavy, general, prefill-heavy로 구성하여 입력 길이와 출력 길이의 차이가 병렬화 방식의 성능에 미치는 영향을 확인하였다. 실험 결과, 동일한 두 GPU 환경에서도 워크로드의 입출력 길이와 요청 부하에 따라 병렬화 방식별 성능 우위와 오버헤드가 달라짐을 확인하였다.

본 연구는 2026년 과학기술정보통신부 및 정보통신기획평가원의 AI중심대학사업 지원을 받아 수행되었음(2026-0-00036)

† Corresponding Author

2. 배경지식 및 연구동기

2.1 LLM 추론과정

LLM은 입력 문장을 토큰 시퀀스로 변환한 뒤 이전까지의 토큰들을 바탕으로 다음 토큰을 예측하는 방식으로 출력을 생성한다. LLM의 추론은 크게 두 단계로 나뉜다. 먼저 prefill에서는 입력 prompt 전체를 한 번에 처리하여 각 layer의 attention 계산에 필요한 key와 value를 만들고 이후 재사용할 수 있도록 KV cache에 저장한다. 다음 decode에서는 이 KV cache를 참조하면서 새로운 출력 토큰을 한 step씩 순차적으로 생성한다.

2.2 기존 병렬화 기법과 Prefill-Decode disaggregation

LLM serving에서 같은 수의 GPU를 사용하더라도 모델 weight, tensor 연산, layer 묶음, 요청, prefill/decode phase 중 무엇을 어떤 단위로 나누는지에 따라 통신량, 메모리 사용량, batching 효율, scheduling overhead가 달라진다

기존 Multi-GPU 병렬화 기법은 주로 TP, PP, DP로 구분할 수 있다. Tensor Parallelism(TP)는 각 layer 내부의 weight tensor를 여러 GPU에 분할 저장하고, 각 GPU가 연산의 일부를 수행한 뒤 통신을 통해 합치는 방식이다. Pipeline Parallelism(PP)는 모델의 연속된 layer 묶음을 여러 layer 분할 구간으로 나누고, 각 구간을 서로 다른 GPU에 배치하여 구간 사이에서 activation을 전달하는 방식이다. Data Parallelism(DP)은 동일한 모델을 여러 GPU에 복제한 후, 각 요청을 개별 GPU로 분산하여 병렬 처리하는 방식이다.

LLM serving에서는 모델이나 요청을 어떻게 나누는지뿐 아니라 한 요청 안에서 발생하는 prefill과 decode를 어떻게 다룰 것인지도 중요하다. LLM 추론과정에서 prefill은 상대적으로 compute-intensive한 특성이 강하고 decode는 memory bandwidth 및 KV cache capacity에 더 민감하므로 서로 다른 병목을 가진다. 따라서 두 단계를 같은 GPU 또는 같은 GPU 그룹에서 함께 batching/scheduling 하면 두 단계가 서로 간섭하

여 latency가 증가한다.

이러한 문제를 완화하기 위해 제안된 방식이 Prefill-Decode disaggregation(PD)이다. prefill과 decode가 서로 다른 병목을 가진다는 점을 이용하여 PD에서는 prefill을 담당하는 인스턴스가 입력 prompt를 처리하여 첫 token과 KV cache를 생성하고, decode를 담당하는 인스턴스가 전달받은 KV cache를 이용해 이후 token을 생성한다.

2.3 기존 연구의 한계

각 분할의 특성이 서로 다르기 때문에 그 중 하나가 항상 우월하다고 보기 어렵다. TP는 layer마다 all-reduce가 필요해 prefill-heavy 워크로드에서는 통신 비용이 커지고, PP는 decode 단계에서 token별 순차 실행이 반복되면서 pipeline bubble, stage 간 activation 전달, layer 분할 불균형의 영향이 있다. DP는 모델을 GPU마다 복제하므로 KV cache 공간이 줄어들고, PD는 KV cache transfer와 prefill/decode 처리량 불균형이라는 추가 비용이 발생한다.

이처럼 각 병렬화 방식은 서로 다른 이득과 오버헤드를 가진다. 기존 연구 DistServe와 Splitwise는 prefill/decode phase 분리를 통해 goodput과 latency를 개선할 수 있음을 보였고, Seesaw는 prefill과 decode 단계가 서로 다른 병렬화 전략을 선호할 수 있음을 보였다. 그러나 Seesaw의 핵심 목표는 DP/TP/PP/PD를 동일한 serving 워크로드에서 고정 configuration 후보로 비교하는 것이 아니라, prefill과 decode 단계에 서로 다른 병렬화 전략을 동적으로 적용하는 model re-sharding 시스템을 제안하는 데 있다. 따라서 동일한 모델과 요청 조건에서 DP, TP, PP, PD를 각각 실행하고, 어떤 워크로드에서 어느 방식이 우세하며 그 성능 차이가 어떤 구조적 원인에서 비롯되는지 분리해 분석한 연구는 충분하지 않다.

이러한 분석은 LLM serving 시스템을 구성할 때 병렬화 방식을 선택하기 위한 실험적 기준을 제공한다. 같은 GPU 자원이라도 입력/출력 길이와 동시성에 따라 처리량 차이가 크게 벌어지므로, 워크로드별로 어느 분할이 유리한지를 사전에 알면 워크로드에 맞는 병렬화 방식을 선택하는 데 도움을 줄 수 있다. 또한 phase별 또는 부하별로 분할을 전환하려면 각 구간별로 분할방식의 성능과 오버헤드 비교에 대한 연구가 필요하다. 성능 저하의 지배적 원인을 분리하면, 통신 최적화, KV cache 관리, batch scheduling, phase 분리 중 어느 부분을 개선해야 효과가 큰지 판단할 수 있다. 본 연구는 위 trade-off를 같은 모델, 요청 조건에서 비교하여, 워크로드별로 어느 분할이 우세한지 분석한다.

3. 실험 환경 및 비교 기준

3.1 실험환경

실험은 NVIDIA RTX 2080 Super 8GB × 2가 장착된 단일 노드 서버에서 수행하였다. 두 GPU는 PCIe Gen3 x8(peer access 미지원)로 연결되어 있다. 모델은 Llama-3.2-1B-Instruct이고 추론 엔진은 vLLM 0.16.0을 사용하였다.

표 1 : 실험환경

Component	Specification
GPU	NVIDIA RTX 2080 Super 8GB × 2
CUDA Version	12.8
vLLM	0.16.0
모델	Llama-3.2-1B-Instruct

3.2 워크로드

입력 워크로드는 표2와 같이 구성하였다. 입력 길이, 출력 길이, 요청 부하가 serving 성능에 미치는 영향을 분석하기 위해 vLLM benchmark의 random dataset을 사용한 synthetic 워크로드와 실제 trace 기반 BurstGPT 워크로드를 함께 사용하였다. synthetic 워크로드의 각 요청은 지정된 입력 토큰 길이만큼 random token prompt를 가지며, 출력 토큰 길이도 워크로드별로 고정하였다.

초당 요청 도착률(QPS)는 {1,2,4,8,inf} 다섯 단계로 설정하였다. QPS가 1과 2인 경우는 낮은 부하 조건, 4와 8은 높은 부하 조건이다. inf는 요청 도착률을 제한하지 않고 가능한 한 빠르게 요청을 주입하는 saturation 조건으로 사용하였다. 실제 burst arrival은 BurstGPT timestamp 간격을 재생한 T-burst-replay에서 별도로 평가하였다. BurstGPT trace 워크로드의 평균 입출력 분포는 표2와 같다. 각 모드, 워크로드, QPS 조합마다 200개의 요청을 발생시켰다. T-burst-replay는 trace의 timestamp 간격을 재생한 burst arrival 워크로드이다.

표 2 : 워크로드

구분	워크로드	평균 입출력 토큰	특성
Synthetic	general	1024 / 128	중간 길이 입력/출력
	prefill-heavy	2048 / 32	prefill 지배
	decode-heavy	128 / 512	decode 지배
Trace	T-general	638 / 224	실제 분포
	T-prefill-heavy	2159 / 132	긴 입력
	T-decode-heavy	848 / 1166	긴 출력
	T-burst-replay	672 / 215	burst tail

초당 요청 도착률(QPS)는 {1,2,4,8,inf} 다섯 단계로 설정하였다. QPS가 1과 2인 경우는 낮은 부하 조건, 4와 8은 높은 부하 조건이다. inf는 요청 도착률을 제한하지 않고 가능한 한 빠르게 요청을 주입하는 burst 조건으로 사용하였다. 각 모드, 워크로드, QPS 조합마다 200개의 요청을 발생시켰다.

3.3 측정 지표

성능 비교를 위해 표3의 지표를 사용하였다. Throughput 지표는 각 전략이 동일 시간 동안 얼마나 많은 요청 또는 토큰을 처리할 수 있는지를 나타내며, latency 지표는 개별 요청이 사용자 관점에서 얼마나 빠르게 응답되는지를 나타낸다.

표 3 : 측정지표

지표	단위	의미
Request throughput	req/s	초당 완료 요청 수
Total token throughput	tok/s	입력 및 출력 토큰을 합산한 처리량
TTFT	ms	Time-To-First-Token
ITL	ms	Inter-Token Latency
GPU utilization	%	GPU 활성 시간 비율

LLM serving에서는 첫 토큰이 출력되기까지의 시간과 이후 토큰이 생성되는 간격이 사용자 체감 품질에 직접적인 영향을 주므로 TTFT와 ITL을 함께 분석하였다. GPU utilization은 200ms 주기로 nvidia-smi를 통해 수집하여 두 GPU의 자원 활용 패턴을 비교하였다.

3.4 측정 방법

각 병렬화 방식은 동일한 모델과 동일한 워크로드 조건에서 독립적으로 실행하였다. Benchmark client는 모든 방식에 대해 동일한 HTTP 기반 LLM serving endpoint로 요청을 전송하도록

구성하였으며, client 측 요청 생성 방식은 고정하고 server 내부의 병렬화 구성만 변경하였다. DP, TP, PP는 하나의 serving endpoint를 구성하여 측정하였다. DP의 경우 요청은 두 GPU 인스턴스에 라운드로빈(round-robin) 방식으로 교대 분배되며, 각 GPU는 독립적인 요청 큐를 관리한다. PD는 prefill과 decode를 별도의 serving 인스턴스로 분리한 구조로 구현하였다. Prefill 인스턴스는 입력 prompt를 처리하여 첫 token과 KV cache를 생성하고, decode 인스턴스는 전달받은 KV cache를 이용해 이후 output token 생성을 수행한다. 본 실험 환경에서는 GPU 간 직접 P2P 전송을 사용할 수 없었으므로, KV cache 전달은 host를 경유하는 통신 경로를 통해 수행되었다. 따라서 PD 결과는 prefill/decode 분리 효과와 함께 KV cache 전달 및 중계 경로의 오버헤드가 포함된 end-to-end serving 성능으로 해석한다.

4. 실험 결과 및 분석

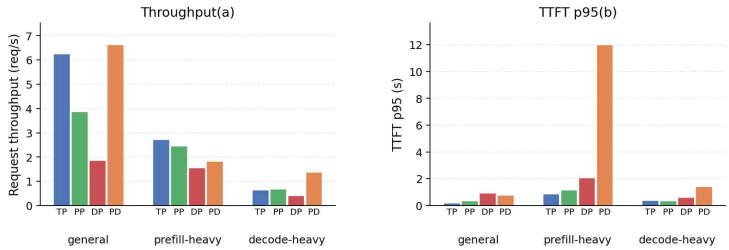

그림 1 BurstGPT trace 워크로드 결과(QPS 8)

4.1 성능 및 오버헤드 분석

그림1(a)은 QPS=8 조건에서 BurstGPT trace 워크로드 request throughput을 비교한 결과이다. 병렬화 방식의 성능 우위는 워크로드와 성능 지표에 따라 달라졌다. decode-heavy에서는 PD가 가장 높은 throughput을 보였다. 이는 입력 prompt가 짧고 output token이 긴 조건에서 decode 단계가 전체 성능을 지배하기 때문이다. prefill과 decode를 서로 다른 GPU 인스턴스에 고정 배치하므로, decode 인스턴스가 긴 prefill 작업과 같은 batch에서 간섭받지 않고 전달받은 KV cache를 기반으로 output token 생성을 지속할 수 있다. 반면 prefill-heavy에서는 긴 prompt로 인해 prefill 인스턴스가 병목이 되어 PD가 처리량 측면에서 우세하지 않았고, 그림 1(b)에서 보이듯 TTFT도 크게 증가하였다. 다만 ITL은 200ms를 상회한 다른 병렬화 방식과 달리 10ms 이하를 유지하여 토큰 생성 지연은 낮게 나타났다. 따라서 prefill-heavy에서는 긴 prefill로 인한 TTFT 증가가 전체 응답성을 지배하여 decode 분리의 이득이 제한되었다.

GPU utilization 측정 결과, TP와 PP는 두 GPU의 utilization 차이가 2% 이내인 반면 DP는 prefill-heavy에서 불균형이 94%에 달했다. prefill-heavy처럼 입력 길이 편차가 큰 워크로드에서는 처리 시간이 다른 요청이 round robin 방식으로 분배되어 워크로드의 분배의 불균형이 있기 때문이다. PD는 prefill GPU의 idle이 input 길이에 따라 결정적으로 달라져 입력/출력 비율이 변하는 워크로드에서는 한쪽 GPU가 idle하는 구조적 비효율이 확인되었다. 한편 PP는 general에서 평균 ITL이 다른 방식보다 컸다. PP는 TP와 달리 layer마다 all-reduce를 수행하지 않고 stage 경계에서만 activation을 전달하므로 통신 횟수 자체는 적다. 그러나 decode 단계에서 토큰을 한 step씩 순차적으로 생성하기 때문에 매 step마다 두 stage를 차례로 통과해야 하고 이 과정에서 pipeline bubble과 stage 간 activation 전달 지연이 반복적으로 누적된다. 이로 인해 PP는 통신 횟수가 적다는 이점에도 불구하고 ITL이 커지는 결과를 보였다. PD는 가장 높은 request throughput을 보였지만, TTFT p95가 786ms로 크게 증가하였다. TP는 throughput이 PD보다 약 6% 낮았지만, TTFT p95는 PD의 1/4 수준에 불과하였다. 이는 동일한 워크로드에서도 throughput을 우선하는지, first-token latency를 우선하는지에 따라 적합한 병렬화 방식이 달라짐을 의미한다.

4.2 DP의 병목 확인

DP는 동일한 모델을 여러 GPU에 복제하여 요청을 분산 처리하므로 요청 수준 병렬성이 좋지만, 모델 가중치가 각 GPU에 중복 저장되어 KV cache에 사용할 수 있는 메모리 여유가 줄어든다. 이를 확인하기 위해 DP에서 긴 입력 context와 높은 동시 요청 조건을 주고 성능 변화를 측정하였다. 실험 결과, 입력 1024/출력512 general 워크로드에서 request throughput은 7.7% 감소하였다. 반면 TTFT는 46배 증가, latency도 3.5배 증가하였다. 입력2048/출력256 prefill-heavy 워크로드에서도 throughput은 26% 증가, TTFT는 14배 증가하였다. 또한 서버 내부 대기열에 누적된 요청 수가 증가하여 추가 동시 요청이 실제 처리량 향상보다 queueing delay 증가로 이어졌음을 확인하였다. KV cache 점유율은 입력 길이와 동시 요청 수가 증가할수록 커졌다. 이는 DP가 짧은 출력과 적당한 요청 부하에서는 강한 baseline이 될 수 있지만, 긴 context와 높은 동시 요청 조건에서는 latency가 민감한 serving에 불리해질 수 있음을 보여준다.

5. 결론

본 연구는 RTX 2080 Super 2-GPU 환경에서 네가지 분할전략을 같은 조건으로 비교하고, 그 결과를 해석하였다. PD는 decode-heavy에서 강하고 prefill-heavy에서는 TTFT 측면에서 불리하지만, 모든 워크로드에서 ITL은 가장 안정적이었다. 이는 PD의 이득이 ITL과 TTFT에 비대칭적으로 나타난다는 것을 의미하며, PD가 prefill-heavy에서 보인 TTFT 저하는 KV transfer 비용뿐 아니라 GPU 활용 비대칭에서 비롯되므로, PD는 input/output 비율이 일정한 환경에서 효율이 극대화된다. General 에서 TP는 throughput과 TTFT 간 균형이 가장 좋아, first-token latency와 처리량을 함께 중시하는 서비스에 적합하였다. PD는 더 높은 throughput을 달성하지만 TTFT가 크므로, 응답 대기 시간보다 처리량을 우선하는 배치성 워크로드에 더 적합하다. DP는 짧은 output과 적당한 부하에서는 좋은 기법이지만, 긴 context와 높은 동시성에서는 GPU 활용률 불균형(94%)으로 latency가 커질 수 있어, throughput이 가장 높은 방식이 응답 시간 안정성 관점에서 항상 유리하지는 않다.

참고 문헌

[1] Y. Zhong et al. “DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving,” in 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), pp. 193-210, 2024..

[2] P. Patel et al. “Splitwise: Efficient Generative LLM Inference Using Phase Splitting,” in 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA), pp. 118-132, 2024.

[3] Q. Su et al. “Seesaw: High-throughput LLM Inference via Model Re-sharding,” Proceedings of Machine Learning and Systems, vol. 7, pp. 1-15, 2025.

[4] Y. Wang et al., “BurstGPT: A Real-World Workload Dataset to Optimize LLM Serving Systems,” Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ‘25), Vol. 2, pp. 5831-5841, 2025.