전자통신동향분석, 40권 5호 2025, 1-12

LLM 추론 성능 향상을 위한 Disaggregated Prefill 기술 동향

Trends in Disaggregated Prefill for Enhancing LLM Inference Performance

안후영 (H.Y. Ahn, ahnhy@etri.re.kr) 슈퍼컴퓨팅시스템연구실 책임연구원
윤정은 (J.E. Yoon, jeyoon@etri.re.kr) 슈퍼컴퓨팅시스템연구실 석사후연수연구원
이수광 (S.K. Lee, 2sookwang@etri.re.kr) 슈퍼컴퓨팅시스템연구실 연구원
김경민 (K.M. Kim, kyungmin.kim@etri.re.kr) 슈퍼컴퓨팅시스템연구실 선임연구원
이훈순 (H.S. Lee, hunsoon@etri.re.kr) 슈퍼컴퓨팅시스템연구실 책임연구원
박준혁 (J.H. Park, vzx00770@etri.re.kr) 슈퍼컴퓨팅시스템연구실 석사후연수연구원
안신영 (S.Y. Ahn, syahn@etri.re.kr) 슈퍼컴퓨팅시스템연구실 책임연구원/실장

ABSTRACT

Recent large language models (LLMs) require substantial computational and memory resources, and the tradeoff between latency and throughput becomes more pronounced as input/output sequence lengths and user concurrency increase. Traditional model-centric optimization techniques offer limited solutions to system-level bottlenecks. In particular, in multiuser environments, resource contention and interference between the prefill and decode stages can significantly degrade the overall inference performance. This study highlights disaggregated prefill as a system-level approach to address these challenges. By decoupling the LLM inference pipeline into distinct prefill and decode stages, and executing them independently on different accelerators, this technique reduces the inference latency and maximizes resource utilization. We review state-of-the-art research from 2024 to 2025 and provide a system-level architectural perspective on how disaggregated prefill mitigates performance bottlenecks in LLM inference. The approach is analyzed through four key technical components: (1) prefill/decode instance configuration, (2) key-value cache transmission, (3) key-value cache management, and (4) request scheduling. Our analysis confirms that disaggregated prefill is particularly effective in improving large-scale LLM inference performance, especially in multiuser and heterogeneous hardware environments.

KEYWORDS AI, Disaggregated Prefill, Heterogeneous Computing, Inference, LLM

* DOI: https://doi.org/10.22648/ETRI.2025.J.400501
* 본 연구는 한국전자통신연구원 기본사업의 지원을 받아 수행되었음[25ZS1100, 이종 가속기 기반 대규모 AI 병렬컴퓨팅 기술 개발].

I. 서론

초거대 언어모델(LLM: Large Language Model)은 다양한 과제에서 뛰어난 성능을 보이며 생성형 인공지능의 핵심 기술로 자리매김하고 있다. 특히 GPT[1], DeepSeek[2], Gemini[3] 등 수십억에서 수천억 개의 매개변수를 갖는 초대형 모델이 등장함에 따라, 이러한 모델을 기반으로 한 인공지능 서비스에서 실시간 추론의 중요성이 더욱 주목받고 있다. 그러나 LLM 추론 성능을 향상시키는 일은 여러 가지 기술적 도전과제를 수반한다.

첫째, LLM 추론 과정은 계산 집약적(Compute-Intensive)일 뿐만 아니라 메모리 집약적(Memory-Intensive) 특성을 함께 지닌다. 특히 입력과 출력 시퀀스의 길이가 증가할수록 연산량과 메모리 접근 횟수가 기하급수적으로 늘어나는 경향이 있다. 이러한 특성은 다수의 사용자에게 대규모 모델을 실시간으로 제공해야 하는 추론 시스템에서 응답 지연(Latency)의 증가와 처리량(Throughput)의 감소라는 문제로 이어진다.

둘째, 실제 서비스 환경은 다수 사용자의 다중 발화(Multi-Turn) 요청이 유입되는 다중 사용자(Multi-Tenant) 구조로, 서로 다른 길이와 복잡도를 가진 요청들이 동시에 처리된다. 이 과정에서 Prefill 단계가 완료되지 않은 요청이 Decode 단계를 지연시키거나, 긴 시퀀스가 전체 연산 자원을 독점함으로써 시스템 전반의 성능을 저하시키는 간섭(Interference) 현상이 발생할 수 있다. 이러한 문제는 단일 모델 수준의 최적화만으로는 해결이 어려운 LLM 추론 시스템의 구조적인 한계에 해당한다.

셋째, 기존 LLM 추론 성능 최적화 기법은 대부분 모델 구조 단순화나 양자화(Quantization)와 같이 모델 내부에 초점을 맞춘 접근이 주를 이루었다. 반면, Prefill과 Decode 작업을 처리하는 하드웨어 가속기 자원 간의 경합(Resource Contention)과 가속기별 처리 시간 비대칭성과 같은 문제를 해결하려는 시스템 아키텍처 차원의 접근은 상대적으로 부족한 실정이다.

이러한 배경에서 최근 연구들은 LLM의 추론 과정을 구성요소 단위(Prefill과 Decode)로 분해하고, 이를 분리(Disaggregate) 처리하는 기술, 즉 Disaggregated Prefill 접근법에 주목하고 있다.

본고는 2024년부터 2025년 사이에 발표된 주요 선도연구들을 분석함으로써, 이 새로운 아키텍처 패러다임이 기존 LLM 추론 시스템의 구조적 한계를 어떻게 극복하고 있는지를 체계적으로 고찰하고자 한다.

II. Disaggregated Prefill 배경 및 개요

1. 배경

LLM을 활용한 추론 서비스에서는 사용자에게 빠르고 정확한 응답을 제공하는 것이 중요하다. 이를 위해 최근에는 단순한 처리량이나 지연시간만을 고려하기보다는, Goodput이라는 지표가 성능의 핵심 기준으로 제시되고 있다. Goodput은 응답을 생성하는 전체 과정에서 TTFT(Time To First Token)와 TPOT(Time Per Output Token)를 모두 줄이는 것을 목표로 하며, 단위 시간당 유효한 출력 결과를 얼마나 많이 생성할 수 있는지를 나타낸다. 따라서 Goodput이 높을수록 사용자의 요구를 더 빠르고 효율적으로 처리할 수 있는 시스템이라 할 수 있다.

LLM 추론은 일반적으로 두 개의 주요 단계인 Prefill과 Decode로 구성된다. 이 중 Prefill 단계는 사용자의 입력 프롬프트와 이전 문맥(Context)을 기반으로 디코더의 Self-Attention에 사용할 KV(Key-Value) Cache를 생성하는 연산 집약적인 처리 단계이며, Decode 단계는 Prefill에서 생성된 KV Cache

를 활용해 한 번의 반복(Iteration)에 한 개의 토큰을 생성하는 메모리 집약적인 처리 단계이다. Decode 단계에서는 전체 KV Cache를 다시 만들지 않고, 새로운 토큰에 해당하는 부분만 생성해서 기존 KV Cache에 덧붙이는 방식으로 동작한다.

이처럼 각 단계의 연산 특성이 상이하지만, 기존 시스템들은 Prefill과 Decode 단계를 동일한 종류의 CPU(Central Processing Unit), GPU(Graphic Processing Unit), 또는 NPU(Neural Processing Units) 가속기에서 수행해 왔다[4,5]. 그 결과, 기존 LLM 추론 시스템에서는 연산 간섭과 자원 활용의 비효율성이 발생하며, 이는 전반적인 추론 성능 저하로 이어지는 한계로 작용한다.

2. 개요

앞 장에서 설명했듯이, 기존 LLM 시스템의 한계는 Prefill과 Decode가 서로 다른 성격의 연산임에도 불구하고 동일한 자원에서 수행된다는 점이다. 예를 들어, Prefill은 연산 부하가 매우 크기 때문에 다수의 코어와 대용량 연산 자원을 요구하는 반면, Decode는 낮은 연산 부하를 가지지만 많은 메모리 대역폭을 요구한다. 이러한 특성의 차이로 인해 두 작업이 동시에 수행될 경우 자원 간섭이 발생하며, 특히 다수의 사용자 요청이 동시에 들어올 경우 처리 지연이 심화된다. 이로 인해 Goodput은 저하되고, 시스템이 SLO(Service Level Objective)를 만족시키지 못할 가능성이 커진다.

Disaggregated Prefill은 LLM 추론에서 Prefill 단계와 Decode 단계를 분리한 뒤, 각 단계에 최적화된 연산 자원에 배치함으로써 자원 간섭을 줄이고 성능을 향상시키는 기술이다. 예를 들어, Prefill은 고성능 GPU에서 처리하고, Decode는 메모리 접근 지연(Latency)에 민감한 NPU에서 수행할 수 있다.

이러한 이기종 자원 분리 구조를 통해 여러 가지 효과를 기대할 수 있다. 첫째, 대규모 연산 자원에 Prefill 작업을 집중시킴으로써 전체 처리량이 획기적으로 향상된다. 둘째, 메모리 최적화가 이루어진 자원에서 Decode를 분리 수행함으로써 응답 지연을 최소화할 수 있다. 셋째, 연산 특성에 따라 적합한 자원을 선택적으로 활용함으로써 시스템 자원의 효율성이 증대된다. 마지막으로, 이기종 환경에서의 워크로드 분리는 시스템의 확장성을 높여, 더 많은 사용자 요청을 유연하고 효율적으로 처리할 수 있게 한다.

Disaggregated Prefill은 특히 이기종 가속기로 구성된 고성능 컴퓨팅 환경이나 클라우드 기반 대규모 추론 서비스에서 중요한 아키텍처 전략으로 자리 잡고 있으며, 향후 LLM 추론의 핵심 기술 요소로 부상하고 있다.

III. Disaggregated Prefill 기술 동향

그림 1은 Disaggregated Prefill 기본 구조로, A. 스케줄러, B. Prefill 인스턴스, C. Decode 인스턴스, D. KV Cache 전송, E. 배치(Batch)로 구성된다. 이때, A. 스케줄러는 사용자들의 요청을 분할하여 배치로 구성하는 A-1. Router와 각 요청에 적합한 Prefill, Decode 인스턴스 쌍을 결정하는 A-2. Selector로 구성된다.

기존의 Attention 메커니즘은 KV Cache를 연속된 GPU 메모리 공간에 저장하기 때문에 요청의 최대 길이에 맞춰 메모리를 사전에 할당해야 했다[6,7]. 이로 인해 GPU 메모리에 내부 및 외부 단편화(Fragmentation)가 발생하며, 실제 메모리 사용률이 20~40%에 불과한 한계가 존재한다[8]. 이러한 문제를 개선하기 위해 KV Cache를 고정 크기의 블록과 페이지 단위로 분할하여 비연속적으로 관리하는

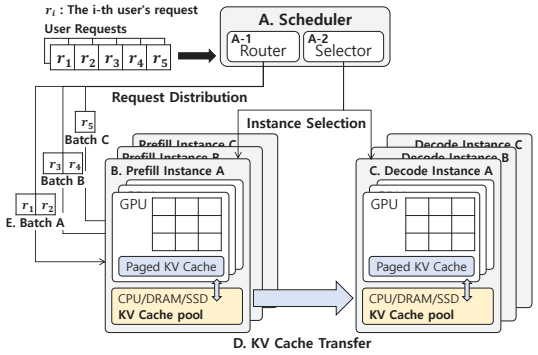

그림 1 Disaggregated Prefill 기본 구조

PagedAttention 기법이 제안되었다[8].

다수의 선도연구에서는 효율적인 KV Cache 관리를 위해 해당 기술을 채택하고 있으며, 그림 1의 Prefill 및 Decode 인스턴스 내부에 표시된 파란색 블록인 Paged KV Cache는 PagedAttention 기반 KV Cache를 나타낸다.

각 인스턴스 내부의 KV Cache 풀(Pool)은 GPU의 VRAM(Video Random Access Memory)뿐만 아니라 CPU의 주메모리, SSD 등 다양한 저장 매체를 활용하여 KV Cache를 저장 및 관리하는 저장소이며, 연구마다 구현 방식에 차이가 존재한다.

이러한 구조에서 고성능 LLM 추론을 수행하기 위해서는 다음과 같은 요소 기술이 요구된다.

첫째, Prefill 및 Decode 인스턴스를 구성하는 기

표 1 Disaggregated Prefill 선도연구 목록 및 요소 기술

참고 문헌	시스템명	요소 기술				출판연도
		P/D Instance 구성	KV Cache 전송	KV Cache 관리	요청 스케줄링	
[9]	Splitwise	정적 (III.1.1절)	KV Cache 계산과 전송의 중첩 (III.2.1절)	Cache Store 미사용 (III.3.1절)	작업 특성 미 고려 (JSQ) (III.4.1절)	'24
[10]	DistServe	정적 (III.1.1절)	네트워크 성능을 고려한 전송 구조 변경 (III.2.2절)	Cache Store 미사용 (III.3.1절)	작업 특성 미 고려 (JSQ) (III.4.1절)	'24
[11]	TetriInfer	동적 (III.1.2절)	-	Cache Store 미사용 (III.3.1절)	작업 특성 미 고려 (JSQ) (III.4.1절)	'24
[12]	Loongserve	동적 (III.1.2절)	-	Cache Store 미사용 (III.3.1절)	워크로드 특성 및 GPU 메모리 사용량 고려 (III.4.2절)	'24
[13]	Mooncake	정적 (III.1.1절)	-	Cache Store 사용 (III.3.2절)	Cache-Aware (III.4.3절)	'25
[14]	P/D-Serve	동적 (III.1.2절)	전송 효율 향상을 위한 자료구조 (Continuous Buffer) (III.2.3절)	Cache Store 미사용 (III.3.1절)	인스턴스 로컬 대기열을 제거한 스케줄링 (III.4.4절)	'24
[15]	HexGen-2	정적 (III.1.1절)	-	Cache Store 미사용 (III.3.1절)	-	'25
[16]	MemServe	동적 (III.1.2절)	전송 효율 향상을 위한 자료구조 (Huge Page) (III.2.3절)	Cache Store 사용 (III.3.2절)	Cache-Aware (III.4.3절)	'24

술, 둘째, Prefill 인스턴스에서 생성된 KV Cache를 Decode 인스턴스로 전달하는 KV Cache 전송 기술, 셋째, 다양한 저장 매체를 활용해 KV Cache를 효율적으로 관리하는 KV Cache 관리 기술, 마지막으로, 여러 사용자의 요청을 배치로 구성하고 이를 적절한 인스턴스에 할당하는 스케줄링 기술이다.

표 1은 본고에서 조사한 선도연구들의 요소 기술 구현 방식을 정리한 것이다. 각 선도연구는 요소 기술을 구현하는 데 있어 서로 다른 접근 방법을 제시하고 있다. 본 장에서는 이러한 요소 기술의 구현 동향을 분석하고, 그 접근 방식 및 기술적 특성을 고찰하고자 한다.

1. P/D Instance 구성 기술

Disaggregated Prefill 구조에서는 워크로드의 요청 길이 분포, 요청의 도착 패턴, SLO를 분석하여 Prefill과 Decode 인스턴스 간의 최적 비율을 결정해야 한다. 이러한 인스턴스 구성은 전체 시스템의 성능을 좌우하는 핵심 요소가 되며, 구성 방법은 크게 정적 구성과 동적 구성 두 가지로 구분할 수 있다.

1.1 정적 인스턴스 구성

정적 인스턴스 구성 방식은 LLM 서빙 시스템 시작 전에 사전 워크로드 프로파일링을 기반으로 Prefill, Decode 인스턴스의 비율을 미리 결정한다.

Splitwise, DistServe[9,10]는 학습용 요청 데이터, 목표 SLO, 사용 모델 등을 기반으로 최적의 인스턴스 비율을 탐색하는 시뮬레이터를 구현하였다. Mooncake[13]는 워크로드의 특성 및 성능 지표를 바탕으로 최적의 비율을 결정한다. HexGen-2[15]는 그래프 기반 알고리즘을 통해 각 워크로드에 적합한 인스턴스를 구성한다. 이때, GPU들을 통신 대역폭과 메모리 용량을 기반으로 그룹화하고 그룹 간 통신 대역폭이 가장 큰 그룹들을 Prefill과 Decode 쌍으로 할당한다.

1.2 동적 인스턴스 구성

하지만 실제 서비스 환경에서는 요청 패턴과 길이가 유동적으로 달라진다. 이러한 측면에서 정적 인스턴스 구성 방식은 새로운 워크로드가 감지되었을 때, 인스턴스 비율을 다시 학습하고 배포해야 하므로 변화를 실시간으로 반영하기 어렵다. 이러한 문제를 해결하고자 동적으로 인스턴스를 구성하는 방법이 연구되고 있다.

TetriInfer[11]는 GPU 작업 부하를 모니터링하여 지난 1분간 부하가 10% 미만이면 인스턴스 역할을 전환하는 방법을 제안한다. 이 방법은 동적으로 인

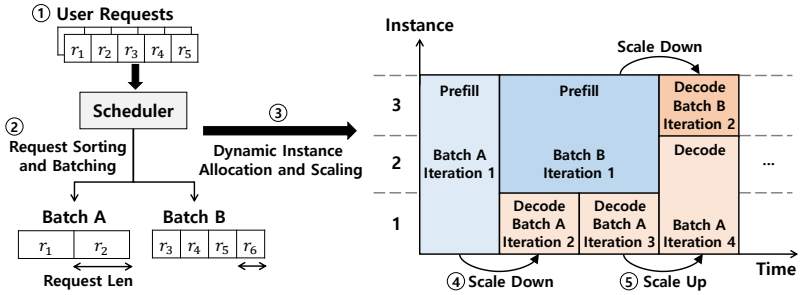

그림 2 (a) 사용자 요청 길이 기반 배치 구성 (b) 동적 P/D 인스턴스 할당

스턴스 비율을 조정할 수 있지만, 100ms 주기로 모니터링하여 실시간성이 부족한 단점이 있다.

Loongserve[12]는 추론 단계 및 요청 프롬프트 길이에 따라 인스턴스의 구성을 실시간 변경한다. 그림 2(a)는 사용자 요청의 길이를 기반으로 배치를 구성하고, 이에 따라 그림 2(b)는 인스턴스를 동적으로 할당하는 과정을 예시로 나타낸다. 사용자 요청이 들어오면 길이를 기반으로 요청을 분할하여 배치를 구성하고 Batch A와 같이 길이가 긴 요청들로 구성된 경우, Prefill에서 더 많은 계산이 요구되므로 Prefill 인스턴스의 개수를 3개로 최대화한다. Batch B와 같이 짧은 길이의 요청들로 구성된 배치는 상대적으로 계산량이 작으므로 2개의 Prefill 인스턴스로 처리한다. 반면, Decode는 계산 작업 부하가 Prefill보다 상대적으로 낮으므로 인스턴스의 개수를 1개로 감소시킨다. 이때, Decode 단계는 반복이 진행될수록 KV Cache 크기가 증가하므로 인스턴스의 개수를 2개로 증가시켜 GPU 리소스 효율성을 최적화한다.

P/D-Serve[14]는 각 워크로드에 적합한 비율을 가진 xPyD 인스턴스 그룹을 구성하고, 이를 콘텐츠 및 트래픽 변화에 따라 동적으로 조정한다. 콘텐츠는 프롬프트 엔지니어링으로 인한 프롬프트 길이와 생성 토큰 수를 의미하고, 트래픽은 동일한 워크로드의 요청량을 의미한다. 콘텐츠 특성이 변화할 경우, 그림 3(a)와 같이 xPyD 인스턴스 그룹 내에서 각 인스턴스의 비율을 조정한다.

예를 들어, 프롬프트 길이가 증가하면 Prefill 작업량이 증가하므로, 그림 3(a)의 Group A처럼 Prefill 인스턴스 수를 확장한다. 반대로 생성되는 토큰 수가 줄어들면 Decoding 작업량이 감소하므로, Group B와 같이 Decode 인스턴스 수를 축소한다.

트래픽이 변화하는 경우, xPyD 인스턴스 비율은 유지하면서 그룹 단위로 스케일링을 수행한다. 예를 들어 4P2D 인스턴스 그룹에 적합한 워크로드의 요청량이 두 배로 증가하면, 그림 3(b)와 같이 기존 그룹과 동일한 구성으로 추가 그룹을 생성하여 처리 용량을 늘린다. 이와 같은 방식은 콘텐츠와 트래픽 변화에 따라 최적의 자원을 할당하여 시스템의 전체 추론 성능을 향상시킨다.

MemServe[16]는 인스턴스 상태를 실시간 모니터링하고 장애를 감지한다. 또한, 런타임 중 인스턴스를 동적으로 추가하거나 제거할 수 있어 특정 인스턴스에 장애가 발생할 경우 이를 대체 인스턴스로 신속히 교체하여 중단없이 안정된 서비스 제공하게 한다.

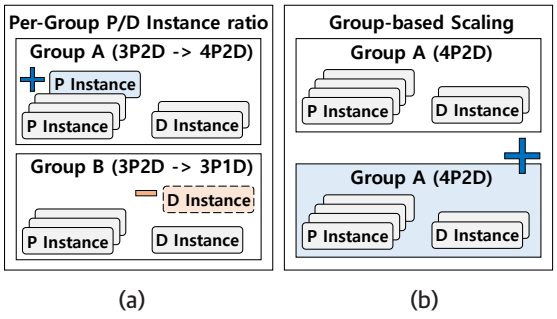

그림 3 워크로드 특성 기반 인스턴스 동적 구성:
(a) 콘텐츠 변화에 따른 그룹 내 xPyD 인스턴스 비율 조정
(b) 트래픽 변화에 따른 xPyD 인스턴스 그룹 단위 스케일링

2. KV Cache 전송 기술

Disaggregated Prefill 구조에서는 Prefill 인스턴스에서 생성된 KV Cache를 Decode 인스턴스로 전송하는 지연시간이 전체 시스템의 주요 병목이 되므로 해당 지연시간을 줄이는 것이 중요한 최적화 대상이다.

2.1 KV Cache 계산과 전송의 중첩

그림 4(a)의 KV Cache 계산 후 전송 방식에서는

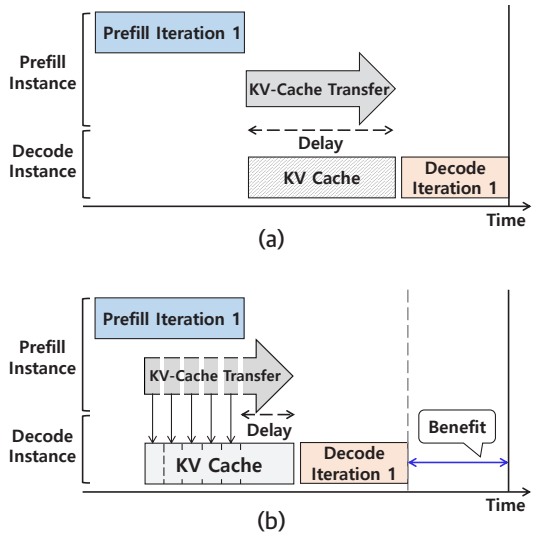

그림 4 (a) KV Cache 계산 후 전송 방식
(b) KV Cache 계산과 전송 중첩 방식

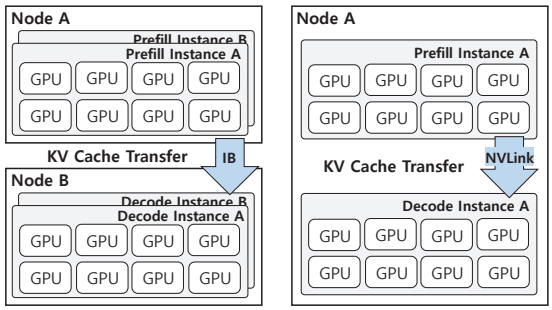

그림 5 KV Cache 전송 구조:
(a) 노드 간 전송 (b) 노드 내 전송

Prefill 단계가 완료되고 첫 번째 토큰이 생성된 이후에 KV Cache의 전송이 시작된다. 이때 KV Cache의 크기는 프롬프트의 토큰 수에 비례하므로, 프롬프트가 길어질수록 전송 지연이 증가하는 문제가 발생한다.

이 문제를 해결하기 위해 Splitwise[9]는 그림 4(b)와 같이 Prefill 계산과 KV Cache 전송을 중첩 수행하는 Layer-Wise Transfer 기법을 제안한다. 이 기법에서는 모델의 각 레이어 계산이 완료되는 즉시 해당 레이어의 KV Cache를 생성하고, 이를 비동기적으로 전송하는 동시에 다음 레이어의 Prefill 계산을 병렬로 수행한다.

그 결과, 그림 4(b)에서 보이는 것처럼 각 레이어의 KV Cache가 점진적으로 Decode 인스턴스로 전달되며, 마지막 레이어의 KV Cache와 첫 번째 토큰만 수신되면 즉시 Decode 단계를 시작할 수 있어 전체 지연시간을 효과적으로 단축한다.

또한, 프롬프트가 짧은 경우에는 전송 오버헤드가 적으므로 Prefill 계산 완료 후 전송하는 방식을 사용하며, 프롬프트가 긴 경우에는 계산과 전송을 병렬로 수행하여 프롬프트 길이에 따라 전송 방식을 달리함으로써 전송 성능 최적화를 도모한다.

2.2 네트워크 성능을 고려한 전송 구조 변경

DistServe[10]는 KV Cache 전송 시 시스템의 네트워크 환경에 따라 전송 구조를 유동적으로 조정하는 방식을 제안한다. 예를 들어, InfiniBand(IB)와 같은 고속 네트워크 환경에서는 노드 간 통신 오버헤드가 작으므로 KV Cache 전송으로 인한 지연이 상대적으로 적다. 따라서, 고속 네트워크 환경에서는 그림 5(a)와 같이 Prefill 인스턴스를 노드 A에 Decode 인스턴스를 노드 B에 분리하여 배치한다.

반면, 노드 간 통신 대역폭이 제한된 환경에서는 그림 5(b)와 같이 Prefill과 Decode 인스턴스를 동일 노드 내에 배치하고, NVLink와 같은 고속 인터커넥트를 활용하여 인스턴스 간 KV Cache를 전송하는 방식을 적용한다. 이처럼 네트워크 성능에 따라 전송 구조를 적절히 변경함으로써 KV Cache 전송 지연을 효과적으로 줄일 수 있다.

2.3 전송 효율 향상을 위한 자료구조 최적화

이러한 최적화에도 불구하고, KV Cache 전송 과

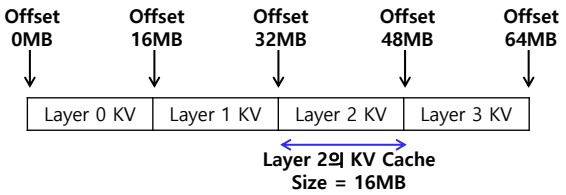

그림 6 Contiguous Buffer 구조

정에서 PagedAttention의 블록 단위 구조로 인해 전송 효율성에 한계가 발생한다. 각 블록을 개별적으로 전송할 경우, 블록 수에 비례하여 네트워크 호출 횟수가 증가하며, 이로 인해 각 전송 시 발생하는 제어 오버헤드가 누적되어 전체 전송 성능이 저하된다[14,16].

P/D-Serve[14]는 이러한 문제를 개선하기 위해 Block-Free D2D KV Cache 전송 방식을 제안한다. 이는 Prefill 인스턴스에서 KV Cache를 그림 6과 같이 Contiguous Buffer로 구성하여 일괄 전송하고, Decode 인스턴스에서 이를 다시 Block으로 복원하는 방식이다.

이를 통해 특정 레이어만 선택적으로 전송하거나 전체 모델의 KV Cache를 일괄 전송할 수 있다. 따라서, 네트워크 호출 횟수를 줄이고 전송 제어 오버헤드를 제거할 수 있어 D2D(Device-to-Device) 대역폭 활용률을 향상시킬 수 있다.

MemServe[16]는 Continuous Buffer와 유사하게 다수의 KV Cache 블록들을 하나의 연속된 큰 블록인 Huge Page로 병합하여 전송함으로써 전송 오버헤드를 효과적으로 줄인다.

3. KV Cache 관리 기술

3.1 KV Cache Store를 사용하지 않는 기술

Without Cache Store 방식은 각 인스턴스의 GPU VRAM에 KV Cache를 저장한 후, GPU 간 P2P(Point-to-Point) 통신을 통해 CPU를 경유하지 않고 직접 KV Cache를 전송하는 구조를 사용한다. 이때 노드 간 전송에는 RDMA(Remote Direct Memory Access)를, 노드 내 전송에는 NVLink와 같은 고속 인터커넥트를 활용하여 통신 지연을 최소화한다. 이러한 방식은 낮은 지연 시간으로 높은 처리량을 달성할 수 있어, 여러 연구에서 효과적인 방법으로 채택되고 있다[9-12,14,15].

3.2 KV Cache Store를 사용하는 기술

Without Cache Store 방식은 KV Cache 재사용 측면에서 한계를 가진다. 각 노드는 생성된 KV Cache를 자신의 GPU VRAM에만 저장하는 구조이기 때문에 다른 노드에서 생성된 캐시를 활용할 수 없다. 예를 들어, 노드 A가 8,000토큰 분량의 문서를 처리하며 KV Cache를 생성하더라도 동일한 문서에 대한 후속 요청이 노드 B로 전달되면 해당 캐시를 활용할 수 없어 초기 단계부터 재계산해야 한다. 더욱이 GPU VRAM의 용량은 제한적이기 때문에 생성된 KV Cache를 장기간 저장하기 어렵고, 메모리 확보를 위해 기존 캐시를 삭제해야 하는 상황이 발생한다. 이러한 구조는 이후 동일한 요청에 대한 캐시 재사용 가능성을 낮추어 전체 시스템의 효율성을 저하시킬 수 있다.

이러한 한계를 극복하기 위해 Mooncake와 Memserve[13,16]는 With Cache Store 방식을 제안한다. 이 방식에서는 그림 1과 같이 인스턴스 내부에 KV Cache 풀을 구성하여, 생성된 KV Cache를 GPU 메모리뿐만 아니라 CPU 주메모리 및 SSD와 같은 다양한 저장 매체에도 저장한다. 이를 통해 모든 인스턴스 간에 KV Cache를 공유할 수 있으며, 다계층 메모리 및 저장 자원의 계층적 활용을 통해 메모리를 더욱 효율적으로 관리할 수 있다. 이로 인해 KV Cache의 재사용률이 크게 향상되며 전체 처리 시

간 또한 단축되는 효과를 얻을 수 있다. With Cache Store 방식을 대표하는 시스템인 Mooncake의 구조 및 스케줄링 방식에 대해서는 4.3절에서 자세히 설명한다.

4. 요청 스케줄링 기술

그림 1의 A는 스케줄러가 다수 사용자의 요청을 분할하여 배치(Batch)를 구성한 후, 각 배치를 적절한 P/D 인스턴스에 할당하고 요청을 스케줄링하는 과정을 보인다. 이때 사용되는 스케줄링 전략은 TTFT 및 TPOT 지연 시간에 직접적인 영향을 미치며, 전체 시스템 성능을 결정짓는 핵심 요소 중 하나이다.

4.1 작업 특성을 고려하지 않는 스케줄링

참고문헌 [9-11]에서는 Prefill 인스턴스 선택 시 JSQ(Join The Shortest Queue) 방식을 적용하였다. JSQ는 각 인스턴스의 대기열(Queue)에 포함된 요청 수를 작업량으로 간주하여, 가장 짧은 대기열을 가진 인스턴스를 선택하는 방식이다. Decode 인스턴스는 현재 수행 중인 작업의 부하를 기준으로 가장 낮은 부하를 가진 인스턴스를 선택하는 Least Loaded 방식을 사용했다.

그러나 JSQ 방식은 요청 간 처리 시간 차이를 고려하지 않는다는 한계가 있다. 특히, 프롬프트 길이나 KV Cache 재사용 가능성 등 처리 시간에 영향을 미치는 요소를 고려하지 않아 비효율적인 인스턴스 할당이 발생할 수 있다. 예를 들어, 인스턴스 A는 대기열에 3개의 요청이 존재하지만 각 요청은 1,000토큰 수준이며, 90%의 KV Cache를 재사용할 수 있어 빠르게 처리되는 반면, 인스턴스 B는 대기열에 1개의 요청만 존재하지만 해당 요청은 50,000토큰으로 구성되어 있고 캐시 재사용이 불가능하여 더 많은 처리가 필요하다. 이 경우 JSQ 방식은 인스턴스 B를 선택하게 되며, 결과적으로 전체 시스템 처리 성능이 저하될 수 있다. 이러한 문제를 해결하기 위한 스케줄링 방식들 제안되고 있으며, 이에 대해서는 4.2절, 4.3절에서 소개한다.

4.2 워크로드 특성 및 GPU 메모리 사용량을 고려한 스케줄링

Loongserve[12]는 그림 2와 같이, 스케줄러가 시퀀스 길이가 유사한 요청들을 하나의 배치로 구성함으로써 KV Cache 재사용률을 높이는 방식을 채택한다. 또한 인스턴스 할당 시, 각 인스턴스의 GPU 잔여 메모리 용량을 고려하여 메모리 여유 공간이 적은 인스턴스부터 우선적으로 요청을 할당한다. 이러한 방식은 메모리 파편화를 효과적으로 방지할 수 있으며, 전체 GPU 메모리의 사용률을 향상시키는 데 이바지한다.

4.3 KV Cache Aware 스케줄링

Mooncake[13]는 각 인스턴스에 KV Cache 저장소를 구성하여, 인스턴스 간 전역 캐시 정보를 공유할 수 있도록 설계된 시스템이다.

그림 7은 이러한 구조를 기반으로 동작하는 KV

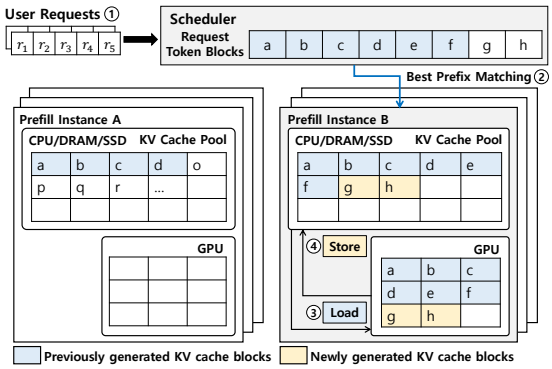

그림 7 KV Cache-Aware 인스턴스 선택 방식

Cache-Aware 인스턴스 선택 방식을 보여준다. 요청이 도착하면 스케줄러는 모든 Prefill 인스턴스에 저장된 KV Cache와 해당 요청의 입력 시퀀스를 비교하여, 재사용 가능한 Prefix의 길이를 계산한다. 여기서 재사용 가능한 Prefix란 서로 다른 요청들의 입력 시퀀스 중 동일한 구간을 의미하며, 두 요청이 동일한 Prefix를 포함하는 경우, 해당 구간에 대해 미리 계산된 KV Cache를 공유할 수 있다.

이를 기반으로 스케줄러는 각 인스턴스의 Prefill 처리 시간, 대기 시간, 그리고 KV Cache 전송 시간을 종합적으로 고려하여, 가장 짧은 예상 처리 시간을 제공하는 인스턴스를 선택한다. 선택된 인스턴스는 이전에 생성된 KV Cache 블록(파란색 블록: a, b, c, d, e, f)을 KV Cache 풀에서 GPU 메모리로 로드(Load)하여 재사용하고, 새롭게 필요한 부분(노란색 블록: g, h)만 추가로 계산한다. 이후 전체 결과는 다시 KV Cache 풀에 저장(Store)되어 모든 인스턴스 간에 공유된다. 이러한 방식은 이후 유사한 요청이 도착했을 때 KV Cache의 중복 계산을 방지하고 이를 재사용함으로써 전체 추론 시간을 효과적으로 단축하는 데 이바지한다.

4.4 인스턴스 로컬 대기열을 제거한 스케줄링

참고문헌 [9-11]에서는 스케줄러가 미리 정의된 기준에 따라 인스턴스를 선택하고, 해당 인스턴스의 로컬 대기열에 요청을 삽입하여 순차적으로 처리했다. 그러나 이러한 방식은 선택된 인스턴스에 이미 다수의 요청이 처리 중인 경우, 새로운 요청이 해당 대기열에서 대기해야 하므로, 상대적으로 빠르게 처리될 수 있는 요청들의 불필요한 대기 시간이 증가하는 문제가 발생한다[14].

이러한 문제를 해결하기 위해 P/D-Serve[14]는 그림 8과 같이, Prefill 인스턴스의 로컬 대기열을 제거하고, 각 인스턴스가 요청을 직접 수락하거나 거부할 수 있는 On-Demand Forwarding 스케줄링 방식을 제안한다. 요청이 도착하면, Gateway에 위치한 스케줄러는 각 인스턴스의 SSE(Server Sent Event) 연결 수를 기준으로 정렬하여, 가장 적은 연결 수를 가진 Prefill 인스턴스를 선택한 뒤 해당 요청을 전달한다. 선택된 Prefill 인스턴스는 현재 처리 중인 배치 크기 (b)와 사전에 정의된 최대 배치 크기 (bp)를 비교하여, 새로운 요청의 수락 여부를 실시간으로 결정하고 Prefill 단계를 수행한다. 만약 선택된 인스턴스가 요청을 거부할 경우, 스케줄러는 후보 인스턴스들을 순차적으로 탐색하며, Timeout 임곗값 내에서 유휴 상태인 인스턴스를 찾을 때까지 반복한다. 이러한 방식은 불필요한 대기 시간을 방지하여 시스템의 처리 성능을 향상시킨다.

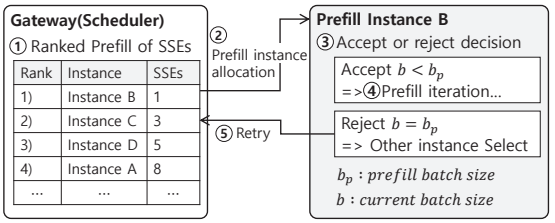

그림 8 On-Demand Forwarding 스케줄링

IV. 결론

LLM 추론 성능을 향상시키기 위해 PagedAttention[8], Continuous Batching[6], Prefix Caching[16], Chunked Prefill[17] 등 다양한 핵심 기술이 제안됐다. 최근에는 이러한 기술들에 더해, 다중 사용자와 이기종 가속기의 활용이 일반화되고 있는 대규모 LLM 추론 시스템에서 Prefill 단계와 Decode 단계를 서로 다른 컴퓨팅 자원에서 분리하여 처리하는 Disaggregated Prefill 기술이 주목받고 있다. 이 방식은 LLM 추론의 성능과 자원 활용 효율성을 동시에 극대화할 방안으로 활발히 연구되고 있다.

특히, Disaggregated Prefill 기술은 다음의 네 가지 핵심 요소 기술을 통해 LLM 추론 성능을 향상시킨다. 첫째, P/D 인스턴스 구성 기술에서는 Prefill과 Decode 인스턴스를 동적으로 구성하여 요청 처리의 유연성을 높인다. 둘째, KV Cache 전송 기술에서는 KV Cache 계산과 전송의 중첩 실행, 네트워크 성능을 고려한 전송 구조 개선, 전송 효율을 높이기 위한 자료 구조 최적화를 통해 병목을 완화한다. 셋째, KV Cache 관리 기술에서는 KV Cache Store를 활용하여 KV Cache를 재활용함으로써 Prefill 단계의 불필요한 연산을 줄인다. 넷째, 요청 스케줄링 기술에서는 워크로드 특성 및 GPU 메모리 사용량, KV Cache 히트율을 고려하고, 인스턴스 로컬 대기열 제거를 통한 스케줄링 기법을 활용해 연산 자원의 활용도를 극대화한다.

ETRI 슈퍼컴퓨팅시스템연구실에서는 LLM 추론 성능 향상을 위한 이종 가속기 기반 대규모 AI 병렬 컴퓨팅 기술 개발을 수행 중이다. 본 연구는 다양한 제조사의 GPU 및 NPU를 최적으로 조합하여 활용함으로써 LLM 추론 성능과 자원 활용 효율을 획기적으로 향상시키는 것을 목표로 한다. 본 연구에서 개발되는 기술은 이기종 연산 자원의 통합 활용을 통한 글로벌 수준의 고성능 LLM 추론 기술 확보를 가능하게 할 뿐만 아니라, 국산 AI 인프라의 경쟁력 강화에도 이바지할 것으로 기대된다.

용어해설

Goodput LLM 추론 시 TTFT와 TPOT를 동시에 줄여 SLO를 만족시키는 성능 지표, 해당 값이 클수록 시스템이 주어진 시간 내에 더 많은 유효한 출력을 생성함

PagedAttention KV Cache를 GPU 메모리 공간에 연속적으로 저장하는 대신 비연속적인 블록 단위로 관리하여 메모리 단편화 문제를 해결하여 사용 효율 향상시키는 기술

Continuous Batching 기존 Static Batching과 달리 토큰 생성 단계별로 반복(Iteration) 단위 스케줄링과 선택적(Selective) 배칭을 수행하여 서로 다른 요청을 하나의 배치로 처리함으로써 GPU 활용률을 높여 추론 지연 시간을 줄이고 처리량을 극대화하는 기술

Prefix Caching 요청들 간에 중복된 Prefill 연산을 피하기 위해 이전 요청의 KV Cache를 저장했다가 이후 요청의 Prefix가 겹치는 경우 이를 재활용하여 추론 속도를 향상시키는 기술

Chunked Prefill Decode 작업을 우선 스케줄링하고 남은 배치 영역에 Prefill 작업을 잘라서 포함시켜 Decode 작업과 병렬로 Prefill 작업을 수행하여 추론 성능을 향상시키는 기술

Disaggregated Prefill LLM 추론 시 Prefill 과정을 특정 장치(예: GPU)에서 수행하고, Decode는 별도의 장치(예: NPU)에서 처리하는 방식으로, 연산을 분리(Disaggregate)하여 자원 활용률을 높이고 추론 성능을 향상시키는 기술

Heterogeneous Computing CPU, GPU, NPU 등 서로 다른 아키텍처를 가진 컴퓨팅 자원을 조합하여 작업을 병렬로 처리하는 방식으로, 각 장치의 특성에 최적화된 연산을 분리하여 수행함으로써 시스템 전체의 처리 효율성과 성능을 극대화하는 기술

참고문헌

[1] J. Achiam et al., “Gpt-4 technical report,” arXiv preprint, 2023. doi: 10.48550/arXiv.2303.08774

[2] A. Liu et al., “Deepseek-v3 technical report,” arXiv preprint, 2024. doi: 10.48550/arXiv.2412.19437

[3] G. Team et al., “Gemini: a family of highly capable multimodal models,” arXiv preprint, 2023. doi: 10.48550/arXiv.2312.11805

[4] Tenstorrent Website. https://tenstorrent.com/

[5] S.J. Moon et al., “Lpu: A latency-optimized and highly scalable processor for large language model inference,” IEEE Micro, vol. 44, no. 6, 2024, pp. 17-33.

[6] G.I. Yu et al., “Orca: A Distributed Serving System for Transformer-Based Generative Models,” in Proc. USENIX Symp. Oper. Syst. Des. Implement., (Carlsbad, CA, USA), Jul. 2022.

[7] FasterTransformer. https://github.com/NVIDIA/FasterTransformer

[8] W.S. Kwon et al., “Efficient Memory Management for Large Language Model Serving with PagedAttention,” in Proc. CM Symp. Oper. Syst. Principles, (Koblenz, Germany), Oct. 2023, pp. 611-626.

[9] P. Patel et al., “Splitwise: Efficient generative llm inference using phase splitting,” in Proc. Annu. ACM/IEEE Int. Symp. Comput. Archit., (Buenos Aires, Argentina), Jun. 2024, pp. 118-132.

[10] Y. Zhong et al., “DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving,” in Proc. USENIX Symp. Oper. Syst. Des. Implement., (Santa Clara, CA, USA), Jul. 2024.

[11] C. Hu et al., “Inference without interference: Disaggregate llm inference for mixed downstream workloads,” arXiv preprint, 2024. doi: 10.48550/arXiv.2401.11181

[12] B. Wu et al., “Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism,” in Proc. ACM SIGOPS Symp. Oper. Syst. Principles, (Austin, TX, USA), Nov. 2024, pp. 640-654.

[13] R. Qin et al., “Mooncake: Trading more storage for less computation—a KVCache-centric architecture for serving LLM chatbot,” in Proc. USENIX Conf. File Storage Technol., (Santa Clara CA USA), Feb. 2025, pp. 155-170.

[14] Y. Jin et al., “P/D-serve: Serving disaggregated large language model at scale,” arXiv preprint. 2024. doi: 10.48550/arXiv.2408.08147

[15] Y. Jiang et al., “HexGen-2: Disaggregated Generative Inference of LLMs in Heterogeneous Environment,” arXiv preprint, 2025. doi: 10.48550/arXiv.2502.07903

[16] C. Hu et al., “Memserve: Context caching for disaggregated llm serving with elastic memory pool,” arXiv preprint, 2024. doi: 10.48550/arXiv.2406.17565

[17] A. Agrawal et al., “Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve,” in Proc. USENIX Symp. Oper. Syst. Des. Implement., (Santa Clara, CA, USA), Jul. 2024.