제15장. 다중 노드 추론, 병렬 처리, 디코딩 및 라우팅 최적화

LLMs은 계속해서 파라미터 수가 에 달하는 거대한 규모로 확장되고 있습니다. 특히, 내장된 전문가 게이트 메커니즘을 통해 다수의 전문 서브네트워크("전문가")를 결합하는 혼합 전문가(MoE) LLMs의 등장으로 모델 파라미터 크기는 수백억에서 수조 단위로 증가했습니다. 주어진 입력에 대해 활성화되는 매개변수는 극히 일부에 불과하지만, 이처럼 거대한 모델 규모에서 추론을 수행하려면 작업 부하를 여러 GPU에 분산해야 합니다.

본 장에서는 최신 NVIDIA GPU를 활용하여 이러한 초대형 LLMs에 대한 효율적이고 고성능의 다중 노드 추론을 수행하기 위한 고급 최적화 기법에 초점을 맞춥니다. 하드웨어와 알고리즘 혁신을 모두 활용하여 지연 시간을 최소화하고 처리량을 극대화하는 분산 추론 시스템 설계 방식을 논의할 것입니다.

먼저 추론 작업 부하를 독립적으로 조정 가능한 별개의 단계로 분할하는 분산 프리필 및 디코드(PD, 분산 PD) 아키텍처 에 대해 논의합니다. 다음으로 데이터, 텐서, 파이프라인, 전문가, 컨텍스트와 같은 핵심 추론 중심 병렬화 전략을 탐구하고, 이를 조합하여 다수의 GPU에 걸쳐 대규모 모델을 서비스하는 방법을 살펴봅니다.

이후 메두사(Medusa), 이글(EAGLE), 초안-검증(draft-and-verify) 방식 등 추측적 디코딩 기법을 다룹니다. 이는 기존 자동회귀형 대규모 언어모델(LLMs)의 단일 토큰 생성 방식과 달리, 추론 과정에서 다중 토큰을 생성 및 평가할 수 있게 하여 순차적 디코딩 병목 현상을 극복합니다. 또한 출력 형식 강제 적용을 위한 제약 디코딩(예: 사용자 정의 JSON 스키마)과 MoE 모델의 동적 라우팅 전략을 논의하여 시스템의 전문가 게팅 및 부하 분산 효율성을 개선합니다.

분리형 프리필 및 디코드 아키텍처

앞서 언급한 바와 같이, 현대 LLaMA의 추론 워크플로우 는 프리필(prefill)과 디코드(decode)라는 두 가지 서로 다른 단계로 구성됩니다. 우리는 이 단계를 분리하기 위해 *분산형 프리필 및 디코드*를 구현할 수 있습니다. 이를 통해 프리필 및 디코드 클러스터를 독립적으로 확장할 수 있으며(서로 다른 하드웨어 플랫폼에서도 가능), 이 장 후반부에 자세히 설명하듯이 대규모 LLaMA 서비스 성능을 크게 향상시킬 수 있습니다.

크로스 벤더 또는 크로스 아키텍처 배포 시에는 양측 간 KV 캐시 레이아웃과 데이터 유형(dtypes)이 일치해야 합니다. 실제 운영 환경에서는 프리필(prefill)과 디코딩(decode)을 호환 가능한 GPU 계열에서 수행해야 합니다. 이렇게 하면 동일한 수치 형식을 사용하여 KV 캐시 전송과 데이터 재사용을 손쉽게 구현할 수 있습니다.

프리필 단계에서 모델은 입력 prompt 전체(종종 수천, 수만, 심지어 수백만 개의 토큰)를 단일 전방 전달(forward pass)로 처리하여 LLM이 계산한 초기 숨겨진 상태를 생성합니다. 그런 다음 입력 prompt의 모든 토큰에 대한 어텐션 키-값(KV) 캐시를 채웁니다. 그림 15-1은 분산 프리필과 디코드가 KV 캐시를 공유하고 KV 전송을 계산과 중첩시키는 방식을 보여줍니다.

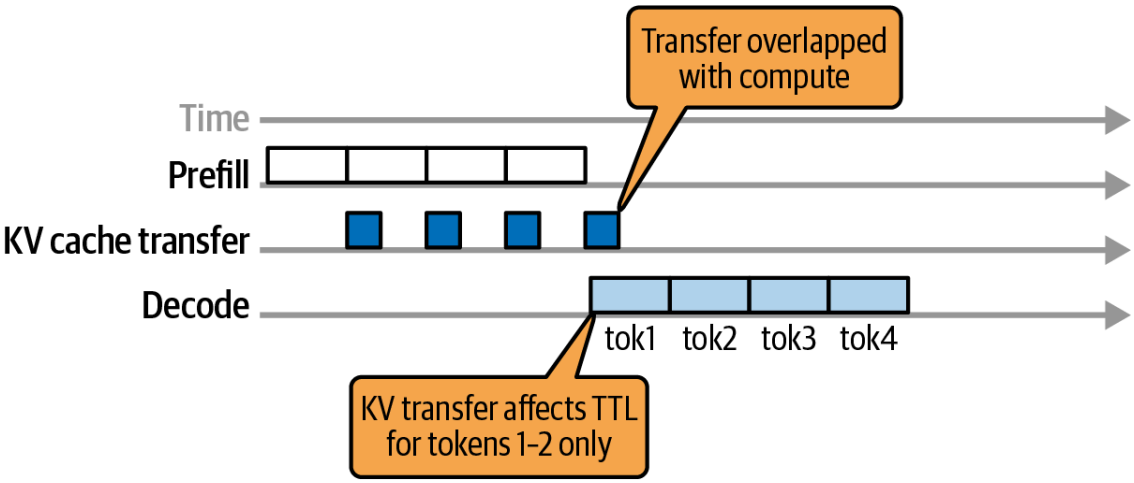

그림 15-1. KV 캐시를 공유하고 계산과 중첩되는 KV 전송을 통해분산된 프리필 및 디코딩

디코드 단계에서 모델은 시퀀스의 각 새 토큰을 예측하기 위해 자동회귀 생성을 수행합니다. 이는 이전에 생성된 모든 토큰의 캐시된 어텐션 KV 표현을 활용하여 이루어집니다.

추측적 디코딩(Speculative decoding)은 단일 배치에서 여러 토큰을 사전 생성함으로써 자동 회귀 디코딩( ) 프로세스를 가속화합니다. 동시에 해당 토큰들의 정확성을 검증합니다. 이는 표준 토큰별 자동 회귀 디코딩의 순차적 특성을 줄여줍니다. 추측적 디코딩은 잠시 후 다루겠습니다.

프리필-디코드 간섭

기존 LLM 추론 시스템은 이 두 단계를 동일한 노드에 배치하고 모든 계산을 단순히 일괄 처리했습니다. 그러나 이러한 단순한 접근 방식은 흔히 *프리필-디코드 간섭이라고* 불리는 *문제를* 초래합니다. 예를 들어, 긴 prompt 프리필은 GPU를 점유하여 다른 요청의 시간 민감한 디코딩 작업을 지연시킬 수 있으며, 그 반대의 경우도 마찬가지입니다.

프리필과 디코드를 동일한 노드에 배치하면 서로 매우 다른 특성을 가진 이 두 단계에 대해 단일 스케줄링 및 자원 할당 전략을 강요하게 됩니다. 프리필은 대규모 병렬 연산으로 구성됩니다. 반면 디코드는 수많은 소규모 순차적 연산을 요구합니다. 결과적으로 시스템은 한 단계의 성능을 다른 단계보다 우선시하거나 양쪽 요구를 모두 충족시키기 위해 하드웨어를 과도하게 프로비저닝해야 합니다.

분리된 프리필 및 디코드 아키텍처( )에서는 프리필과 디코드 단계가 서로 다른 GPU 풀에 할당됩니다. 이는 두 워크로드 간의 직접적인 간섭을 제거합니다( ). DistServe 시스템은 프리필과 디코드를 분리함으로써 TTFT 및 TPOT 제약 조건 내에서 처리된 유효 요청량(goodput requests)이 최대 7.4배 증가했으며(지연 시간 SLO 최대 12.6배 개선), 이를 보고했습니다.

프리필 및 워커 노드 독립적 확장

상호 간섭을 제거할 수 있다면( ), 긴 프리필 계산으로 인해 디코딩 작업이 지연되는 자원 "사망 시간"을 줄일 수 있습니다. 반대의 경우도 마찬가지입니다. 이렇게 하면 GPU가 유휴 상태로 있는 시간을 줄이고 더 유용한 작업을 수행하는 시간을 늘릴 수 있습니다. 이는 주어진 지연 시간 목표(즉, *굿스루*)에서 활용도와 유용한 처리량을 증가시킵니다.

프리필과 디코드를 별도로 확장하려면 프리필 처리를 위한 노드 집합과 디코드 처리를 위한 노드 집합을 각각 할당하면 됩니다. 두 클러스터는 프리필 작업자에서 디코드 작업자로 인코딩된 prompt 상태(또는 어텐션 KV 캐시)를 전송할 때만 통신합니다(그림 15-2 참조).

여기서 별도의 GPU 작업자가 입력 prompt를 처리하는 프리필 단계와 출력 토큰을 반복적으로 생성하는 디코드 단계를 각각 담당하는 모습을 확인할 수 있습니다. 프리필 단계의 출력에는 prompt용 KV 캐시가 포함되며, 이는 다음 토큰 생성을 위해 디코드 작업자로 전송됩니다.

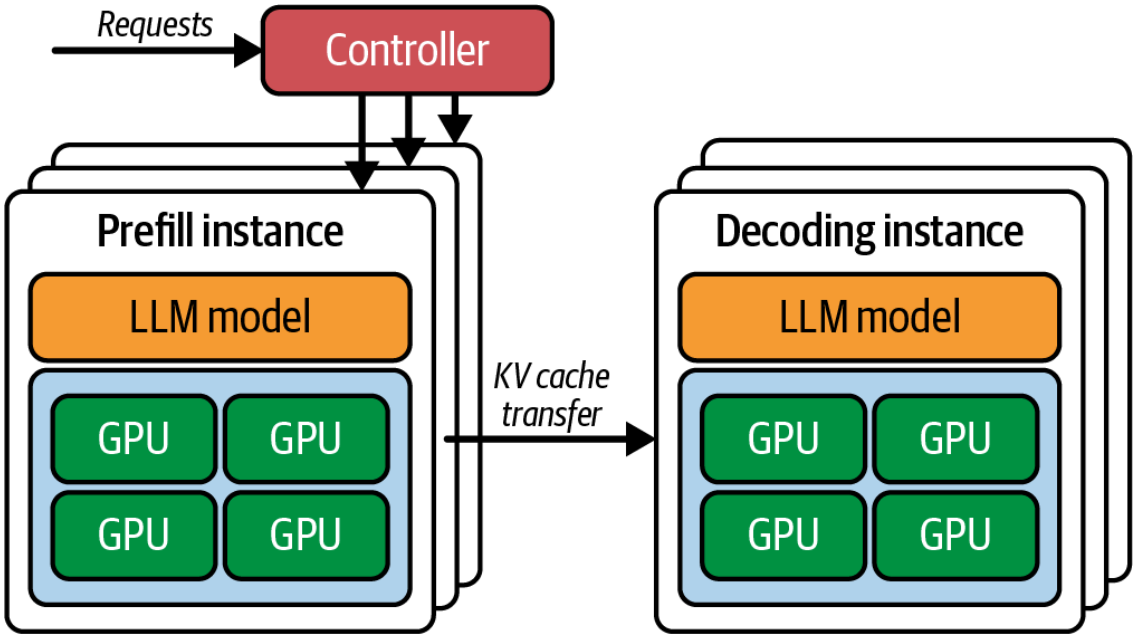

그림 15-2. 분산 추론: 프리필 풀(숨겨진 상태 + KV) → NVLink/NVSwitch(노드 내) 또는 GPUDirect RDMA(노드 간)를 사용한 KV 핸드오프 → 디코드 풀

별도의 GPU 풀을 할당함으로써 시스템은 프리필과 디코딩 파이프라인을 병렬로 동시에 가동합니다. 실제 적용에서 분리는 엄격한 지연 시간 제약 하에서도 처리량을 크게 향상시키는 것으로 입증되었습니다. 일부 연구에 따르면 프리필과 디코딩 단계를 분리하면 상당한 성능 향상이 가능하지만, 결과는 중간 수준의 개선부터 처리량(goodput)이 수 배 증가하는 경우까지 다양합니다. 결과는 워크로드와 네트워크 구조에 크게 좌우됩니다.

이러한 분리를 통해 각 단계는 처리량 또는 지연 시간에 대해 독립적으로 최적화 및 확장될 수 있습니다. 프리필 GPU에서 프리필을 공격적으로 배치하여 디코딩 성능(예: 디코딩 지연 시간 증가)에 부담을 주지 않으면서 처리량을 극대화할 수 있습니다. 또한 별도의 클러스터를 통해 각 단계에 특화된 병렬 처리 설정, 인스턴스 수, 스케줄링 정책을 조정할 수 있습니다.

지연 시간(TTFT) 및 처리량(TPOT)에 미치는 영향

디코딩 GPU는 더 낮은 배치 크기 로 실행하거나 특수 스케줄링을 적용하여 스트리밍 생성을 위한 *출력 토큰당 소요 시간*(TPOT)을 최소화할 수 있습니다. 예를 들어, 긴급한 디코딩 작업을 우선 처리하는 스케줄러를 사용하여 대기열 지연을 방지할 수 있습니다.

이 분리는 각 단계마다 성능 기대치가 다르기 때문에 효과적입니다. 프리필 단계의 *첫 토큰 도달 시간*(TTFT)은 낮은 지연 시간을 위해 최적화되는 반면, 디코드 단계는 낮은 토큰당 처리 시간(TPOT)과 안정적인 스트리밍 지연 시간을 우선시합니다. 종단 간 처리량은 주로 동시성과 스케줄링에 의해 결정됩니다. 기존 설정에서는 TTFT와 토큰당 TPOT 지연 시간 사이에서 타협해야 했습니다. 분리를 통해 두 SLO 목표를 동시에 달성할 수 있습니다.

KV 전송 및 전수 상호 연결 단계 동안 NVLink 및 NIC 사용률을 모니터링하십시오. 목표는 별도의 스트림과 이벤트를 사용하여 복제 및 계산을 중첩시키는 것입니다.

KV 핸드오프는 멀티 GPU 및 멀티노드 전송을 위한 고대역폭 상호연결을 사용하므로 추가 지연이 최소화됩니다. 이러한 상호연결에는 NVLink, NVSwitch, InfiniBand, GPUDirect RDMA를 사용하는 이더넷(RoCE on Ethernet)이 포함됩니다. 예를 들어, ConnectX-8 SuperNIC (800GbE급)을 탑재한 다중 노드 클러스터는 GPUDirect RDMA를 통해 포트당 최대 800Gb/s를 제공합니다. 이는 호스트 매개 통신 경로에 비해 KV 전송 시간을 크게 단축합니다. 또한 프리필-디코드 분산을 최적화하고 MoE 전-대-전 성능을 향상시키기 위해 GPU당 1개의 NIC를 배포하는 것이 권장됩니다.

KV 캐시 데이터 전송 및 NIXL

분산 시스템은 prompt 처리 완료 후 중간 결과(최종 숨겨진 상태와 KV 캐시)를 프리필 작업자에서 디코드 작업자로 전송하기 위해 커넥터 나 스케줄러를 사용합니다. 이 인계 과정은 일부 통신 오버헤드를 발생시키지만, 클러스터의 상호 연결이 고대역폭(예: NVLink 또는 InfiniBand)인 경우 이 오버헤드는 자원 경합 제거로 얻는 이점에 비해 미미합니다.

실제 적용 시 NVIDIA의 NIXL 라이브러리는 토폴로지와 정책에 따라 NVLink/NVSwitch, RDMA 또는 호스트 스테이징 경로를 자동 선택하여 전송 오버헤드를 최소화합니다. 예를 들어, 제4장에서 소개된 NVIDIA의 NIXL 라이브러리는 KV 캐시 전송에 가장 빠른 경로인 를 자동으로 선택합니다. NIXL은 Dynamo 및 vLLM을 포함한 프레임워크 및 추론 엔진과 통합됩니다. NIXL-vLLM 통합 구조는 그림 15-3에 표시되어 있습니다.

구체적으로, LMCache와 NIXL은 지원 경로로서 vLLM의 분산 프리필링에 통합됩니다. NIXL은 또한 NVIDIA Dynamo 및 TensorRT-LLM에서 피어 투 피어 GPU 상호 연결 및 RDMA를 사용하여 KV 캐시 데이터를 전송하는 데 사용됩니다.

노드 내에서 NIXL은 호스트 스테이징 없이 NVLink 및 NVSwitch를 통해 장치 간 전송을 수행합니다. 노드 간에는 NIXL이 InfiniBand 또는 RoCEv2를 통한 GPUDirect RDMA를 사용하여 호스트 복사를 피합니다. 이러한 경로는 멀티기가바이트 페이로드에서도 KV 캐시 핸드오프 지연 시간을 낮게 유지합니다.

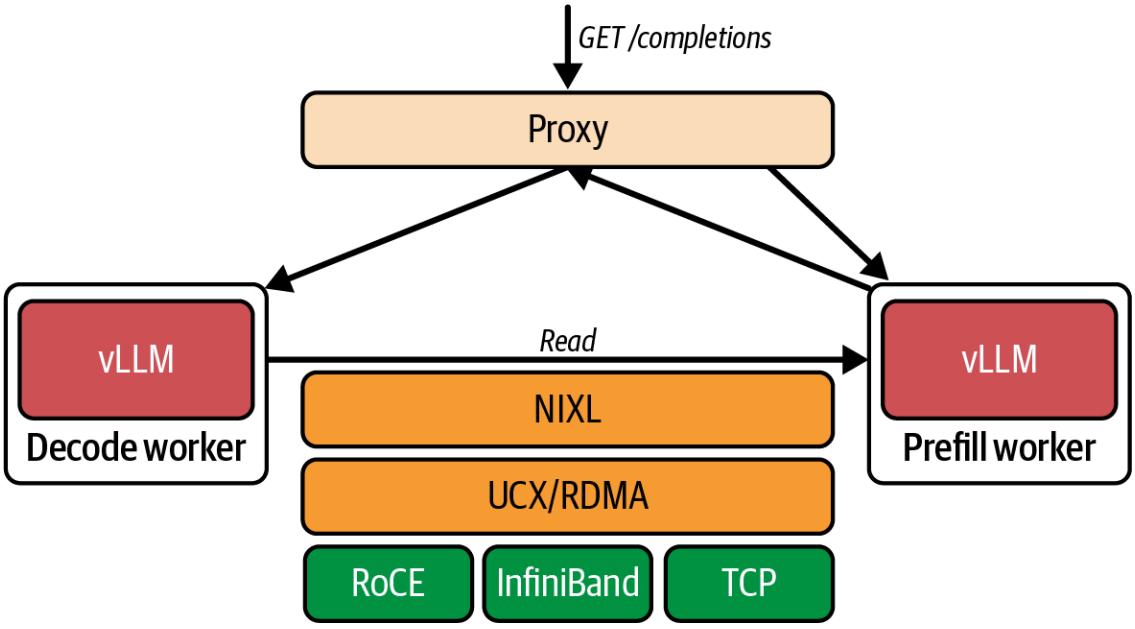

그림 15-3. vLLM 추론 엔진에서 NIXL을 통한KV 캐시 데이터 전송; 노드 내: NVLink/NVSwitch(장치 간); 노드 간: ConnectX급 NIC 사용 GPUDirect RDMA(InfiniBand/RoCEv2)

프리필 및 디코드 작업자의 배치는 패브릭을 따라야 합니다. 동일 노드 배치 시 CUDA 피어 액세스를 통해 NVLink 및 NVSwitch로 KV 전송을 유지하며, 크로스 노드 배치 시 InfiniBand 또는 RoCEv2를 통한 GPUDirect RDMA를 사용해야 합니다. NVIDIA Dynamo는 NIXL과 통합되어 노드 간 GPU, CPU 메모리, 스토리지 간 KV 캐시를 이동하며, vLLM은 LMCache 및 NIXL을 통해 통합되어 분산 프리필링을 수행합니다.

패브릭 또는 가상화 계층이 직접 피어 액세스를 차단할 경우, NIXL은 비최적 경로인 호스트 스테이징 경로로 대체될 수 있습니다. 배포 환경에서 항상 종단 간 KV 전송 시간을 검증하십시오.

쿠버네티스를 통한 분산 프리필 및 디코드 배포

고급 배포 환경에서는 쿠버네티스와 같은 클러스터 오케스트레이션 시스템 이 부하 및 입력 특성에 따라 GPU 풀 할당을 동적으로 전환하거나 풀을 별도로 확장할 수 있습니다. 예를 들어, 많은 사용자가 매우 긴 prompt와 상대적으로 작은 출력(예: 대용량 문서 요약 사용 사례)을 가지고 도착하면, 쿠버네티스는 일시적으로 할당을 변경하여 프리필 풀에서 더 많은 GPU를 사용하도록 할 수 있습니다. 이렇게 하면 디코드 단계에 할당된 GPU 수가 줄어듭니다.

반대로, 매우 긴 출력을 요청하는 사용자가 많을 경우(예: 긴 추론 체인, "단계별 사고" 등), 더 많은 GPU를 디코드 풀로 전환할 수 있습니다. 두 경우 모두 각 작업자 유형별로 새 인스턴스를 확장할 수 있습니다.

그림 15-4는 오픈소스 llm-d 프로젝트를 사용한 분산형, 변형 자동 확장기( ) 기반 쿠버네티스 vLLM 클러스터를 보여줍니다. 별도의 프리필 및 디코드 워커로 구성됩니다. vLLM은 두 인스턴스를 실행하고 LMCache 및 NIXL을 사용해 KV를 전달함으로써 분산 프리필링을 구현하지만, llm-d는 쿠버네티스 네이티브 오케스트레이션을 통해 분산 서비스 및 KV 인식 라우팅으로 이를 확장합니다. 이 다이어그램은 풀 내 프리필 및 디코드 워커의 복제본 수를 업데이트하는 '변형 자동 *확장기*(*variant autoscaler*)'라는 구성 요소를 보여줍니다.

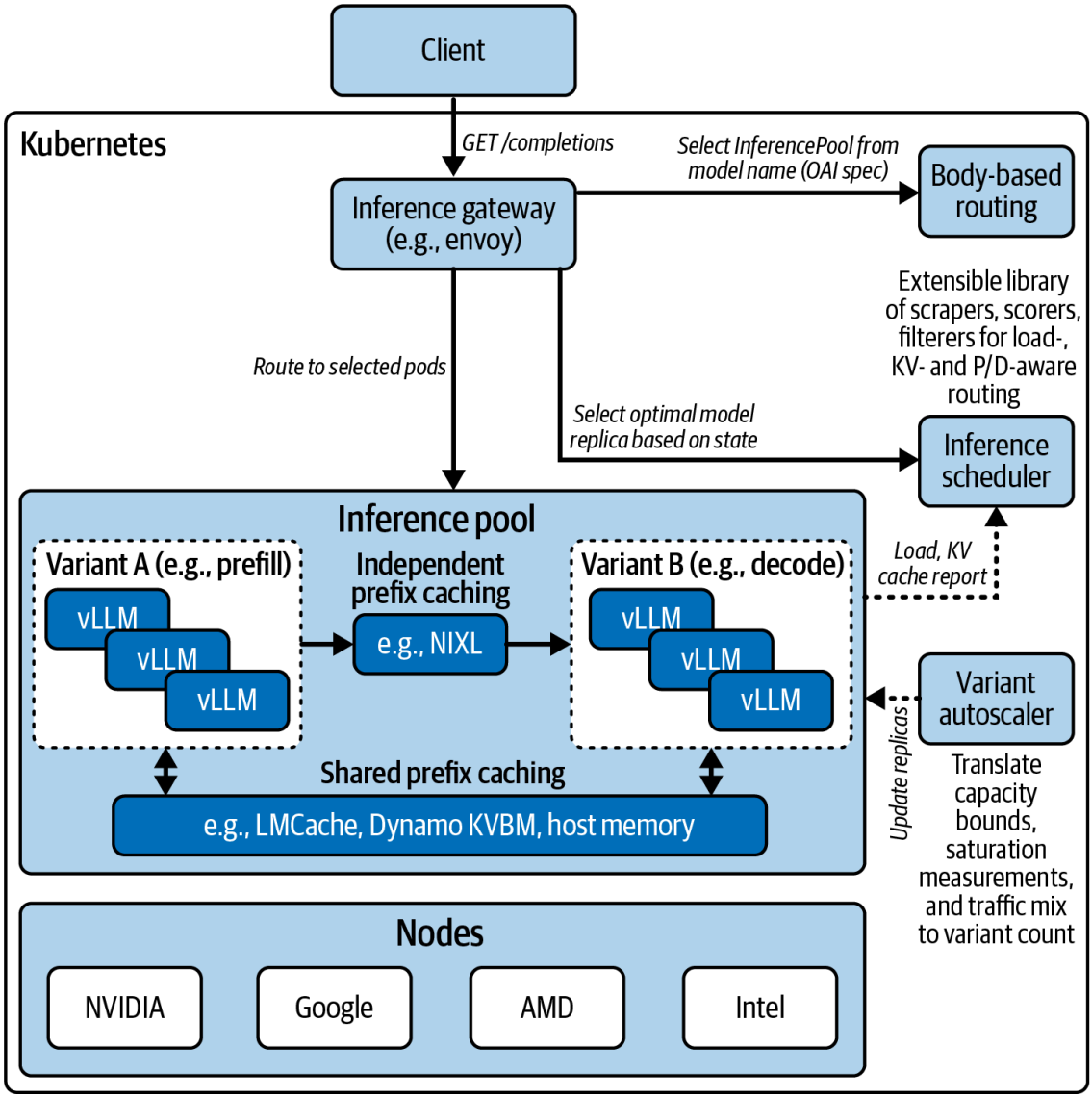

그림 15-4. 오픈소스 llm-d 프로젝트를 활용한 별도의 프리필 및 디코드 작업자로 구성된쿠버네티스 기반 vLLM 클러스터; 변형 오토스케일러가 프롬프트-응답 혼합 비율에 따라 프리필/디코드 복제본 수 조정; LMCache 및 NIXL을 통한 KV 이동

현대적인 추론 배포 환경에서는 모든 노드가 동일한 런타임 및 코드베이스(예: vLLM, SGLang, NVIDIA Dynamo 등)를 공유하므로 프리필과 디코딩 기능을 모두 수행할 수 있습니다. 클러스터 오케스트레이터는 시작 시점에 프리필 또는 디코딩 중 특정 역할을 정적으로 할당하고, 작업자 수명 주기 전반에 걸쳐 동적으로 역할을 재할당할 수 있습니다.

전반적으로 분산 프리필/디코드 아키텍처는 고처리량, 저지연 LLM 서비스의 기반을 제공합니다. 그러나 중간 데이터(긴 prompt의 KV 캐시 기준 수 기가바이트 규모)를 전송 및 관리해야 하므로 복잡성이 발생합니다. 스케줄링도 더 복잡해지지만, 초대형 규모에서는 하드웨어 효율 활용의 이점이 큽니다.

제17장과 제18장에서는 고급 스케줄링, 라우팅 및 배포 최적화를 포함한 분산형 PD를 위한 추가 기술에 대해 더 깊이 탐구합니다.

대규모 MoE 모델 서비스용 병렬화 전략

대규모 MoE 모델을 효율적으로 서비스하려면 제한된 GPU 메모리로 인해 다양한 형태의 병렬화가 필요합니다. 텐서 병렬화, 파이프라인 병렬화, 전문가 병렬화, 데이터 병렬화, 컨텍스트 병렬화 등 핵심 병렬화 전략을 분석합니다. 또한 이러한 전략을 결합하여 LLM을 다수의 GPU에 분산시키는 방법도 논의합니다. 표 15-1은 이러한 전략의 개요 , 일반적인 사용 사례, 추론과 관련된 각 전략에 대한 상세 설명을 제공합니다.

표 15-1. LLM 추론을 위한병렬화 전략

병렬 처리 전략	분할 기준	사용 사례	장점	단점
텐서 병렬 처리	각 레이어 내 (신경망 가중치 행렬을 GPU 간 분할)	단일 모델이 너무 크거나 여러 GPU에 걸쳐 레이어 내의 무거운 계산을 가속화해야 하는 경우	컴퓨팅 바운디드 계층에서의 거의 선형적인 가속도 계산과 중첩된 올-리듀스 통신으로 인한 오버헤드 감소	각 계층에서의 빈번한 통신(올리듀드) 고대역폭 상호연결(NVLink/NVSwitch) 필요 노드 간 및 느린 네트워크에서 높은 지연 시간으로 인해 효율성 저하 (텐서 병렬 그룹을 단일 노드 내에 유지할 것을 권장)
파이프라인 병렬 처리	다른 GPU에 다른 레이어 배치 (레이어 순서로 모델 분할)	하나의 GPU에 들어가지 않는 극도로 깊은 모델 레이어 간 메모리 확장성	모델 상태 분산 허용 마이크로배칭을 사용하여 서로 다른 사용자/요청의 여러 토큰을 동시에 처리 긴 시퀀스의 경우 여러 레이어를 병렬 처리 가능(대규모 배치 또는 긴 입력에 대한 처리량 향상)	파이프라인 채우기/비우기 지연(버블) 추가 — 토큰 단위 디코딩에는 비효율적 구현이 더 복잡하고 활성화 메모리 사용량이 증가함(파이프라인 단계 간 중간 활성화 값 저장 필요)
전문가 병렬 처리	서로 다른 GPU에 서로 다른 MoE 적용 (토큰별 스파스 활성화)	다수의 전문가로 구성된 대규모 MoE 모델 모델 매개변수를 GPU 간	사실상 무제한 모델 크기 가능—총 매개변수는 GPU 수에 비례하여 확장	각 MoE 레이어에서 높은 런타임 통신 오버헤드(전체 대 전체) 게이팅이 고르지 않을 경우 잠재적 부하 불균형 발생

병렬 처리 전략	분할 기준	사용 사례	장점	단점
		에 분할해야 함	각 GPU는 토큰의 일부만 계산함(스파스 컴퓨팅) 높은 매개변수 수가 모델 용량/품질 향상	각 GPU는 통신 비용을 상쇄하기 위해 전문가당 충분한 작업량(토큰)을 보유해야 함
데이터 병렬화	전체 모델을 여러 GPU에 복제하여 각 GPU에서 서로 다른 요청 처리	모델 배포가 확정된 후 처리량 확장(동시 요청 증가) 다중 인스턴스 서비스로 다수 사용자 지원	거의 선형적인 처리량 확장성 구현이 간편함(모델 분할 불필요)	개별 쿼리(즉, *쿼리당 지연 시간*)에 대한 지연 시간 개선 없음 메모리 사용량 증가(각 복제본이 전체 메모리를 사용) 상태 저장 방식(예: 캐시)인 경우 일관성 처리 필요 또는 상태 비저장 모델 호출 사용
컨텍스트 병렬 처리	각 레이어에서 입력 시퀀스 토큰을 GPU 간에 분할	초장문 시퀀스(예: 100k+ 토큰) 처리 시 prompt 지연 시간 및 GPU당 메모리 감소	긴 컨텍스트 사전 채우기에 대해 거의 선형적인 속도 향상 달성 KV 캐시 분할을 통해 단일 GPU 메모리를 초과하는 컨텍스트 처리 가능	파티션 간 어텐션을 처리하기 위해 맞춤형 어텐션 알고리즘 필요 경계 토큰에 대해 레이어당 통신 추가 추가 통신 오버헤드로 인해 매우 긴 컨텍스트(100k+ 토큰)에 유리함

Blackwell NVL72에서 노드 내 TP(Transition Point)의 경우, TP 그룹을 단일 NVSwitch 도메인 내에 유지하는 것을 선호합니다. 추가 홉을 피하기 위해 토폴로지가 허용할 때만 랙 간 확장을 수행하십시오.

이러한 병렬화 전략은 모델 가중치와 데이터가 GPU 간에 어떻게 분할되는지 정의합니다( ). 그림 15-5는 다양한 병렬화 전략 및 일반적인 전략 조합에 따른 분할 방식을 보여줍니다.

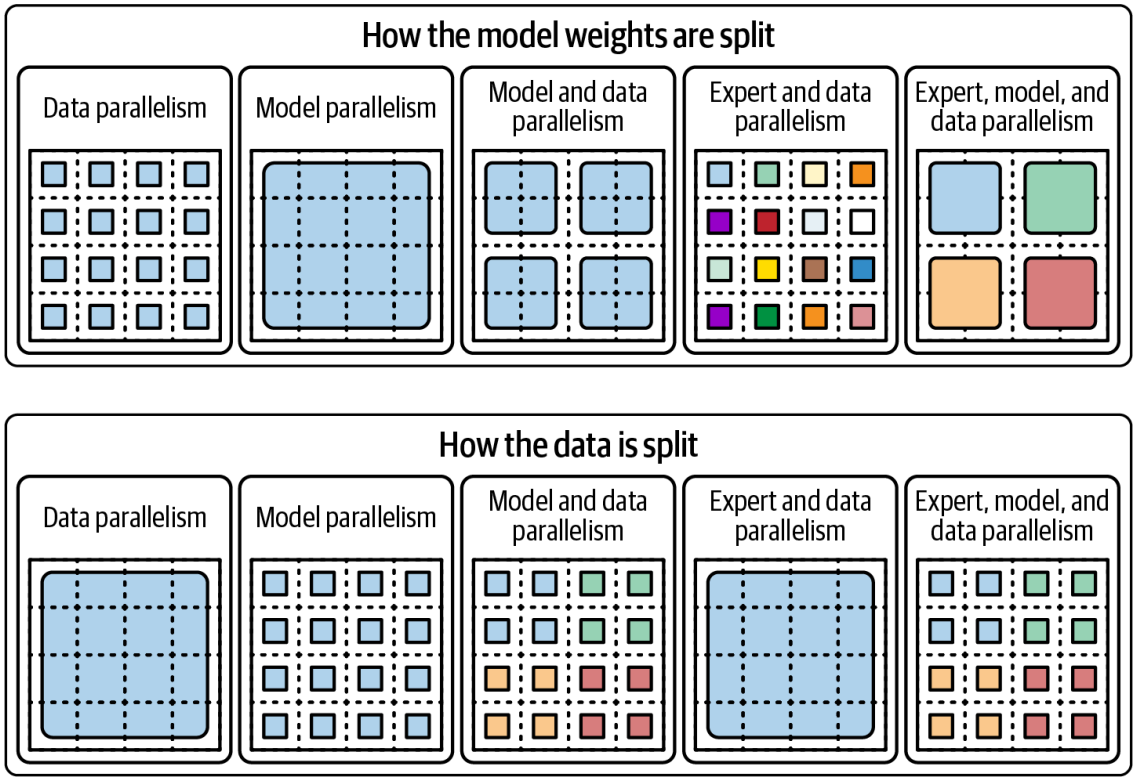

그림 15-5. GPU 간모델 가중치 및 입력 데이터 분할

텐서 병렬 처리

텐서 병렬 처리 (TP)는 신경망 각 레이어 내의 텐서 연산( )을 여러 GPU에 분할합니다. 예를 들어, 트랜스포머 레이어의 대규모 행렬 곱셈은 열 또는 행 단위로 분할되어 두 개 이상의 GPU에서 병렬로 계산될 수 있습니다. 이후 이 GPU들은 부분 출력을 집계하기 위해 올-리듀스(all-reduce)를 수행하며 결과를 교환합니다.

TP는 모델의 *숨겨진* 레이어(*hidden layer)*가 단일 GPU 메모리에 들어가지 않을 정도로 클 때, 또는 동일한 레이어를 여러 GPU에서 병렬로 활용하여 단일 모델 인스턴스의 가속을 원할 때 일반적으로 사용됩니다. 이는 각 레이어의 계산에 대해 모든 GPU를 동기화 상태로 유지하므로 효율성을 위해 극히 높은 GPU 간 대역폭이 필요합니다.

이상적으로는 TP는 고대역폭 NVLink 및 NVSwitch 상호 연결로 연결된 GPU에서 실행됩니다. 5세대 NVLink(최대 1.8TB/s의 GPU 간 양방향 집계 대역폭 및 고급 네트워크 토폴로지 지원)를 사용하는 다중 GPU Blackwell 시스템에서 TP는 8~16개의 GPU 노드에 걸쳐 효율적으로 확장될 수 있으며, NVL72 랙의 경우 최대 72개의 GPU까지 확장 가능합니다. 참고로 GB200/GB300 NVL72 랙 내에서

NVLink 스위치는 72개 GPU 도메인 내에서 약 130TB/s의 총 GPU 대역폭을 제공합니다. 이는 랙 내 대역폭으로서는 매우 큰 규모입니다.

활성화 값의 올-리듀스(all-reduce)와 같은 집합 통신이 계산에 비해 빠를 경우, TP는 모델의 계산 집약적 부분에 대해 거의 선형적인 속도 향상을 제공합니다. 추론에서는 텐서 병렬 처리가 주로 어텐션 투영(attention projections)과 피드포워드 다층 퍼셉트론(MLP) 네트워크의 대규모 행렬 곱셈에 적용됩니다.

한 토큰의 활성화 값은 상대적으로 작기 때문에, 이러한 올-리듀스 통신은 NVLink에서 큰 비용을 차지하지 않습니다. 따라서 TP는 모델 가중치를 근사화하지 않고도 거대한 밀집 모델을 여러 GPU에 분할하는 일반적인 전략입니다.

우리는 주로 단일 MoE 전문가나 트랜스포머 레이어가 단일 GPU에 비해 너무 큰 모델을 서비스하기 위해 TP를 사용합니다. 그러나 각 레이어의 계산을 여러 GPU에 분산시켜 병렬화함으로써 지연 시간을 줄이고자 할 때도 TP를 활용합니다.

그러나 특정 규모를 넘어서면 TP 효율이 떨어진다는 점을 유의해야 합니다. 특히 InfiniBand나 이더넷으로 연결된 노드나 랙 간에 사용할 때 더욱 그렇습니다. 실제로는 노드 내 통신이나 NVLink 도메인 내 노드 간 통신에 TP를 사용하는 것이 가장 좋습니다.

NVIDIA의 GB200/GB300 NVL72와 같은 최신 멀티 GPU 랙은 상호 연결 대역폭을 포화시키지 않으면서 훨씬 더 큰 규모로 TP를 사용할 수 있게 합니다. NVLink 스위치는 NVL72당 72개의 GPU까지 랙 전체에 걸쳐 NVLink 도메인을 확장합니다. NVLink 스위치 시스템은 최대 8개의 랙(576개 GPU, 576개 GPU = 8개 랙 × 랙당 72개 GPU)에 걸쳐 완전 연결된 NVLink 패브릭을 확장할 수 있습니다. 이는 단일 NVL72 내부뿐만 아니라 여러 랙에 걸쳐 대규모 병렬 처리 전략을 가능하게 합니다. 예를 들어, 8랙 토폴로지에서는 8랙에 걸쳐 576개의 GPU(576개 GPU = 8랙 × 랙당 72개 GPU)를 사용하여 576-way 텐서 병렬 처리를 확장할 수 있습니다.

랙 간에 TP를 확장하는 것도 가능하지만, 가능한 경우 단일 NVLink/NVSwitch 아일랜드 내에서 토폴로지를 고려하여 TP 그룹 크기를 선택하는 것이 좋습니다. 이렇게 하면 불필요한 랙 간 스위치 지연 시간을 피하고 전체 시스템 효율성을 높일 수 있습니다.

파이프라인 병렬 처리

파이프라인 병렬 처리 (PP)는 모델을 *레이어별로* 서로 다른 GPU에 분할합니다. 예를 들어, 60개 레이어의 트랜스포머 기반 모델에서 GPU 0은 레이어 1~20을,

GPU 1은 레이어 21~40을, GPU 2는 레이어 41~60을 담당할 수 있습니다.

토큰 시퀀스를 처리할 때 데이터는 순차적으로 GPU를 통과합니다. 즉, GPU 0이 레이어 1~20을 계산한 후 중간 활성화 값을 GPU 1로 전달하여 레이어 21~40을 처리하는 식입니다. 이를 통해 단일 가속기에 수용하기에 너무 깊은 모델도 분산 처리할 수 있습니다.

추론 시 PP는 모델을 여러 배치로 분할하여 처리량(throughput)을 향상시킬 수 있습니다. 이는 조립 라인과 유사한 방식입니다. 긴 입력 토큰 시퀀스를 처리하는 *프리필* (*prefill*) 단계에서 PP는 시퀀스의 서로 다른 부분을 엇갈리게 레이어로 스트리밍함으로써 높은 GPU 활용도를 달성합니다.

반면, 한 번에 하나의 토큰을 생성하는 *디코드* 단계에서는 순수 PP의 이점이 상대적으로 적습니다. 각 새 토큰이 여전히 파이프라인 단계를 순차적으로 통과해야 하기 때문입니다. 이로 인해 파이프라인 버블(유휴 기간)이 발생하며, 각 토큰마다 이전 단계가 후속 단계 완료를 기다리게 됩니다.

파이프라인 버블을 줄이기 위해 구현체들은 마이크로배칭을 사용합니다. 이는 파이프라인이 서로 다른 요청의 여러 토큰을 동시에 처리할 수 있게 합니다. 그럼에도 파이프라인 병렬화는 주로 메모리 확장성에 도움이 되며, 매우 큰 모델이 GPU 간에 레이어를 분할할 수 있게 합니다. 또한 처리량에도 도움이 되는데, 특히 큰 배치 크기나 긴 입력을 처리할 때 그렇습니다.

PP는 파이프라인 단계 간 전송 오버헤드로 인해 단일 항목의 종단 간 지연 시간을 증가시키는 경향이 있습니다. 따라서 PP는 일반적으로 지연 시간보다는 모델 용량 이유로 선택됩니다. 이는 모델을 메모리에 수용할 수 있기 때문입니다. 단일 GPU로 전체 모델을 수용할 수 없는 경우, 약간의 지연 시간 증가가 허용됩니다.

추가로 PP는 메모리와 속도 균형을 위해 TP 같은 다른 병렬화 전략과 함께 사용되는 경우가 많습니다. 대규모 MoE 모델을 서비스할 때, 2~4개의 GPU에 걸쳐 파이프라인 병렬화를 사용하여 깊은 레이어 스택을 분할하는 동시에, 넓은 레이어 내 연산 처리는 TP에 의존할 수 있습니다.

PP는 레이어 간( ) 모델 분할을, TP는 레이어 내 모델 분할을 수행합니다. TP는 주로 단일 GPU에 수용하기에 레이어 또는 전문가(expert)가 너무 넓은 모델에 사용되며, PP는 단일 GPU에 수용하기에 너무 깊은 모델에 주로 사용됩니다. TP와 PP는 결합될 수 있으며, 이는 잠시 후 살펴보겠습니다.

전문가 병렬 처리(EP)

전문가 병렬화(EP)는 MoE 아키텍처에 특화된 기술입니다. MoE 레이어에는 다수의 전문가 네트워크(피드포워드 하위 레이어)가 존재합니다. 각 입력 토큰에 대해 하나 또는 소수의 전문가만 활성화됩니다. 이는 서로 다른 전문가들을 서로 다른 GPU에 분산 배치하는 데 자연스럽게 적합합니다.

예를 들어, MoE 레이어에 16개의 전문가가 있고 4개의 GPU가 있다면 각 GPU는 4개의 전문가를 호스팅할 수 있습니다. 추론 중 토큰이 해당 MoE 레이어에 도달하면 내부 게이트 네트워크가 해당 토큰에 대해 상위 두 전문가(예: 1위와 2위)를 선택합니다. 그런 다음 토큰의 데이터는 해당 두 전문가를 소유한 GPU로 전송되어 처리됩니다. 그런 다음 결과를 다시 결합하여 다음 토큰을 생성합니다(그림 15-6 참조).

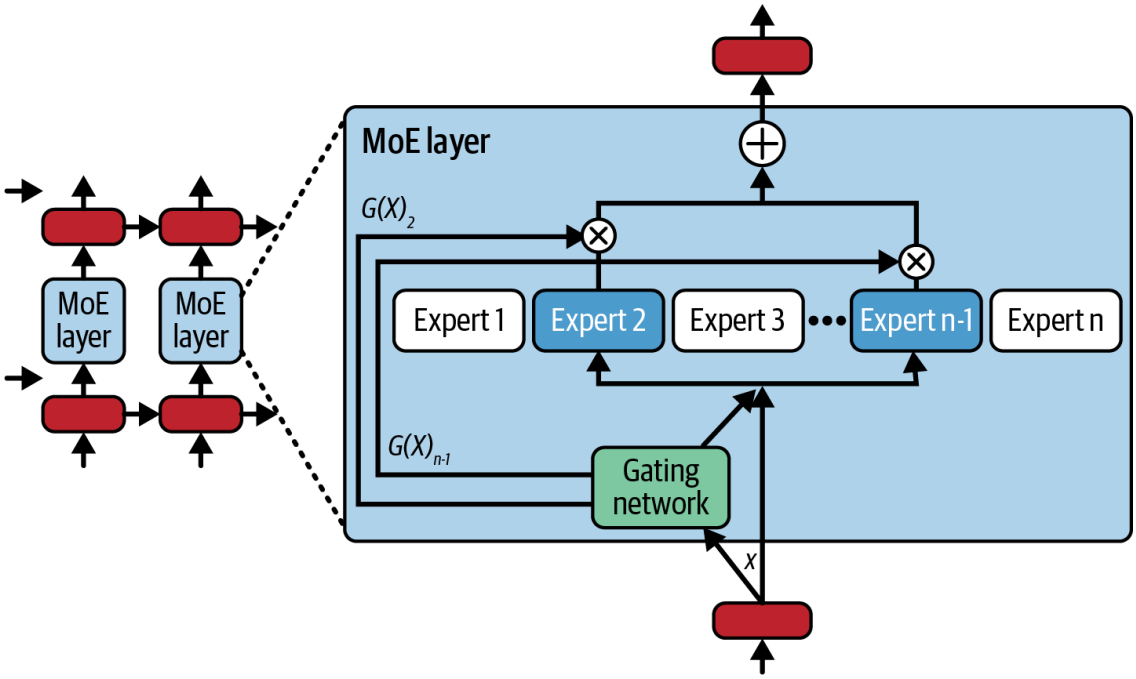

그림 15-6. 전문가 혼합 (MoE) 통신 (출처: *https://oreil.ly/pzn5t*)

이를 구현하려면 토큰(기술적으로는 활성화 벡터)이 할당된 전문가의 GPU로 전달되도록 GPU 간에 동적으로 셔플되는 전-대-전(all-to-all) 통신 패턴이 필요합니다. 각 전문가가 할당된 토큰에 대한 출력을 계산한 후, 토큰 출력을 원래 순서로 반환하기 위해 또 다른 전-대-전 통신이 수행됩니다.

이러한 동적 라우팅은 각 MoE 레이어마다 통신 집약적 단계를 도입합니다. 특히 전문가/GPU 수가 증가할 경우 신중하게 최적화하지 않으면 성능 병목 현상이 발생할 수 있습니다. 장점은 전문가 병렬 처리가 모델의 총 용량을 GPU 수에 거의 선형적으로 확장할 수 있게 한다는 점입니다.

100개의 GPU에 분산된 100명의 전문가로 구성된 MoE를 고려해 보십시오. 각 토큰이 두 명의 전문가만 활성화한다면, 토큰당 계산량은 더 작은 밀집형 모델과 유사합니다. 따라서 전문가 병렬 처리는 모델 가중치가 다수의 GPU에 분할되고 각 GPU가 특정 레이어에서 토큰의 일부만 처리하기 때문에, 일부 대규모 MoE 모델이 서비스될 수 있게 하는 핵심 요소입니다.

규모가 커지고 MoE 모델에서 적절히 균형 잡힌 전문가 집합을 사용하면 모든 전문가가 동시에 활성화될 가능성이 높습니다. 따라서 개별 토큰은 소수의 전문가만 활성화하지만, 모든 최종 사용자의 모든 토큰을 종합하면 모든 전문가가 활성화되어 모든 GPU 리소스를 동시에 소비하고 경쟁하게 됩니다.

효율적인 전문가 병렬 추론에는 세심한 부하 분산과 함께 고대역폭 상호 연결(랙 내 통신용 NVLink/NVSwitch, 노드 간 통신용 InfiniBand/RDMA 등)이 필요합니다. 현대적 MoE 추론 프레임워크는 NCCL 같은 고속 집단 통신 라이브러리를 사용합니다. GPU당 여러 전문가를 그룹화하여 통신 단계를 줄이는 것이 종종 도움이 됩니다.

현대적인 MoE 추론은 종종 각 토큰을 두 전문가에게 할당하는 상위 2개 게이트 방식을 사용합니다. 통신을 줄이기 위해 자주 짝을 이루는 전문가들을 동일한 GPU 또는 컴퓨팅 노드에 그룹화할 수 있습니다. 예를 들어, 토큰이 두 전문가를 사용하는 경우 자주 짝을 이루는 두 전문가를 동일한 GPU 또는 노드에 배치하면 많은 토큰 할당이 단일 GPU 또는 노드에 머물게 되어 통신 트래픽을 국소화하고 오버헤드를 줄일 수 있습니다.

또 다른 기법은 각 토큰이 단일 전문가에게만 할당되는 탑-1 게이팅입니다. 이는 전문가 출력 수가 두 배로 증가하는 앞서 설명한 탑-2 게이팅에 비해 통신량을 줄여줍니다. 탑-1 게이팅은 더 빠르지만 모델 품질 저하와 부하 불균형을 초래할 수 있습니다.

구글의 GLaM은 모멘트 평균(MoE) 모델 훈련 중 전문가 활용 균형을 달성하기 위해 로드 밸런싱 손실(및 게이트 노이즈)을 도입했습니다. 이를 기반으로 연구자들은 실시간 로드 지표를 활용해 특정 전문가가 과부하 상태일 때 토큰을 재라우팅하는 진정한 적응형 추론 시간 게이트 기술을 탐구하고 있습니다. 이는 품질 저하 없이 활용도를 개선합니다.

그러나 대부분의 생산 환경에서는 적당한 용량 계수를 가진 상위 2개 게이트 방식과 가끔씩 부하 기반 전문가 교체 또는 복제를 병행하는 것이 품질과 성능을 동시에 균형 잡는 가장 일반적인 절충 기법으로 남아 있습니다.

많은 MoE LLMs은 품질 향상, 적절한 속도, 균형 잡힌 부하 분산을 위해 최소한 탑-2 게이팅을 사용합니다. 또한 전문가들을 전략적으로 배치하거나 백업 전문가 복제본을 활용하여 전문가 과부하를 방지하는 것이 중요합니다.

전문가들이 과부하 상태( ) 또는 핫 상태(hot)가 되면, 게이트 네트워크가 토큰을 불균형하게 해당 전문가에게 라우팅할 경우 처리량 병목 현상이 발생할 수 있습니다. 이른바 '*스트래글러 효과*(*straggler effect*)'로 불리는 이 현상은 모든 전문가의 계산이 완료되어야만 진행이 가능하게 합니다. 따라서 다른 전문가들보다 더 많은 작업(토큰)을 받는 과부하 상태의 전문가는 추론 파이프라인을 지연시킵니다. 이로 인해 일부 전문가와 그들의 GPU는 유휴 상태로 남아 있는 반면, 다른 전문가들은 이를 따라잡기 위해 노력하게 됩니다.

이를 방지하기 위해 고성능 MoE 서비스 시스템은 용량 계수 매개변수를 노출합니다. 일반적으로 평균 토큰 부하의 1.2~1.5배로 설정되며, 각 전문가가 배치당 처리할 수 있는 토큰 수를 제한합니다. 이를 초과하는 토큰은 차선책인 오버플로 전문가로 라우팅되거나 두 번째 패스를 위해 대기열에 추가됩니다. 이는 훈련 중 MoE가 토큰을 전문가에게 균등하게 할당하도록 유도하기 위해 사용되는 부하 분산 손실이나 게팅 노이즈를 보완합니다.

일부 완전한 기능을 갖춘 추론 서버는 필요 시 핫 전문가들을 여러 GPU에 복제하여 부하를 분산할 수도 있습니다. 이는 추가 GPU 메모리 비용이 발생합니다. 부하 불균형의 예시는 조금 후에 살펴보겠습니다.

추론 시간 스필오버(용량 계수), 훈련 시간 페널티(로드 밸런싱 손실 및 게이트 노이즈), 핫 전문가 복제본의 조합은 용량 한계에 도달했을 때 토큰-전문가 부하 분배를 원활하게 하여 전체 MoE 추론 처리량을 극대화합니다.

데이터 병렬 처리

추론에서 *데이터 병렬 처리* (DP)란 다중 GPU에 전체 모델을 복제하고 각 GPU에 서로 다른 입력 요청(또는 요청 배치의 샤드)을 할당하는 것을 의미합니다( ). 훈련 시 데이터 병렬 처리와 달리, 추론 시 데이터 병렬 처리는 각 복제본에서 독립적인 전방 전파를 실행합니다. 이는 처리량 확장을 위한 가장 간단한 방법입니다.

예를 들어, DP를 사용할 때 하나의 GPU가 초당 10개의 요청을 처리할 수 있다면, 8개의 GPU와 8개의 복제본을 사용하면 요청이 독립적이라고 가정할 때 이상적으로 초당 80개의 요청 처리량을 얻을 수 있습니다. 실제로 추론을 위해 데이터 병렬 처리를 사용하려면 여러 모델 서비스 엔진을 가동하고 이들 간에 요청을 부하 분산해야 합니다.

DP를 사용할 때 각 GPU 또는 GPU 그룹은 시작부터 끝까지 쿼리의 일부를 처리합니다. 장점은 처리량의 선형 확장성과 추론 중 GPU 간 통신이 필요 없다는 점입니다(각 복제본이 독립적으로 실행됨). 그러나 단점도 상당합니다.

DP는 각 복제본이 모델 가중치와 캐시의 전체 사본을 필요로 하기 때문에 메모리 요구량을 배가시킵니다. 따라서 8개의 복제본으로 모델을 서비스하는 것은 단일 GPU에 모델을 호스팅하는 것에 비해 GPU 메모리를 8배 사용하며, 대략 8배의 하드웨어 비용이 듭니다.

실제 환경에서는 데이터 병렬 처리를 각 GPU의 요청 배치 처리 및 다중 스트림 실행과 결합하여 활용도를 극대화하는 경우가 많습니다. 각 복제본은 이상적으로 상태 비저장(stateless) 이어야 하며, 일관성 문제를 피하기 위해 동기화된 캐시를 사용해야 합니다.

DP가 단일 요청의 지연 시간을 줄이지 않는다는 점은 중요합니다. 그러나 메모리 및 컴퓨팅 자원이 확보되어 있고 경합이 상대적으로 적을 경우, 요청을 처리할 수 있는 복제본이 더 많아지기 때문에 처리량은 향상됩니다.

GPU 메모리는 상대적으로 부족하고 비싸기 때문에, DP는 처리량 요구가 매우 높을 때나 TP 및 PP와 같은 다른 병렬화 접근법과 결합할 수 있을 때만 추론에 일반적으로 가치가 있습니다.

추론 시에는 DP의 추가 메모리 및 비용 요구사항으로 인해 TP(타입 병렬화)나 PP(패키지 병렬화) 등 다른 병렬화 기법과 함께 DP를 사용하는 경우가 많습니다. 예를 들어, 메모리 제한으로 인해 모델을 8개의 GPU에 분산해야 하는 경우, DP를 TP, PP 또는 EP와 함께 사용하거나 심지어 이들을 모두 함께 사용하여 4차원(4D) 병렬 처리(컨텍스트 병렬 처리(CP)를 추가하면 5D 병렬 처리!)를 구현해야 합니다.

구체적으로, DP와 TP를 함께 사용하여 두 개의 8-GPU DP 그룹으로 두 개의 대형 8-GPU 모델 복제본을 배포할 수 있습니다. 이렇게 하면 처리량이 두 배로 증가하고, TP를 사용하여 각 대형 모델 복제본을 해당 레이어 내의 8개의 GPU에 분할할 수 있습니다.

대규모 추론 클러스터에서는 다수의 GPU와 노드를 할당하여 대규모 모델의 복수 복사본을 제공함으로써 높은 요청량을 처리할 수 있습니다. DP는 단일 모델 인스턴스가 제공할 수 있는 처리량을 초과하여 확장해야 할 때 특히 유용합니다.

실제 운영 환경에서는 트래픽 수요를 충족시키기 위해 대규모 모델의 다중 DP 복제본을 운영하는 경우가 많습니다.

현대적 추론 서버는 각 복제본을 로드 밸런서 뒤의 별도 모델 인스턴스로 처리합니다. 이는 신중한 요청 라우팅이 필요하지만, 다른 복잡한 모델 샤딩 방식에 비해 상대적으로 간단합니다.

컨텍스트(시퀀스) 병렬 처리

컨텍스트 병렬 처리 (CP)는 단일 토큰 시퀀스를 여러 GPU에 분할하는 보다 전문화된 전략입니다. 이 기술은 수만에서 수백만 토큰 규모의 극히 긴 컨텍스트 입력에 효과적입니다. 이러한 입력은 CP 없이는 너무 느리거나 메모리 부담이 크거나 단순히 단일 GPU에 수용되지 않을 수 있습니다.

CP의 핵심 아이디어는 시퀀스를 여러 청크로 분할하고 각 레이어에서 서로 다른 GPU가 시퀀스의 다른 부분을 병렬로 처리하도록 하는 것입니다. GPU들은 시퀀스 청크 경계에서 필요한 정보만 교환합니다.

CP는 KV 캐시를 여러 GPU에 분산시켜 단일 GPU 메모리보다 큰 컨텍스트를 처리합니다. 또한 사용된 GPU 수에 비례하여 prompt 처리 지연 시간을 대략적으로 감소시킵니다. 따라서 CP는 여러 GPU를 사용하여 긴 컨텍스트의 프리필을 병렬로 수행함으로써 매우 긴 컨텍스트 프리필 실행에 대해 거의 선형적인 속도 향상을 달성할 수 있습니다.

CP의 과제는 트랜스포머가 전역적 자기 주의를 가지므로 모든 토큰이 이전 모든 토큰에 주의를 기울인다는 점입니다. 시퀀스를 단순 분할하면 파티션 경계 간 주의 계산을 위해 GPU 간 대량의 정보 교환이 필요해집니다.

CP 기법은 링 병렬 처리(ring parallelism) 및 블록형 어텐션(blocked attention)과 같은 기발한 방식을 활용하여 이차적 자기 어텐션 통신 시간 복잡도를 줄이고, 각 GPU가 주로 자신의 컨텍스트 분할 영역 내 데이터와 소량의 청크 경계 데이터에만 어텐션하도록 제한합니다.

결과적으로 각 GPU는 레이어별로 입력 시퀀스의 일부 위치만 처리합니다. 중간 결과는 파이프라인 방식으로 전달되며, 종종 링 형태로 배열됩니다. CP는 레이어 차원이 아닌 시퀀스 길이 차원을 따라 진행되는 파이프라인 병렬화와 유사합니다.

컨텍스트 병렬화는 문서를 세그먼트로 분할하고, 여러 GPU에 세그먼트를 할당하며, 각 세그먼트를 동시에 처리함으로써 매우 긴 입력에 대한 사전 채우기(pre-fill)에 탁월합니다. 이는 GPU가 두 배 증가할 때마다 prompt 처리 시간을 대략 절반으로 줄여주며, 경계에서의 세그먼트 간 어텐션에 약간의 오버헤드만 발생합니다.

요약하면, CP는 순차적 토큰별 디코딩 단계( )의 속도를 높이지 않지만, 극도로 긴 prompt의 TTFT를 단축할 수 있습니다. 이는 각 GPU가 해당 세그먼트의 캐시만 저장하도록 장치 간에 어텐션 KV 캐시를 분산함으로써 달성됩니다. 또한 CP를 사용하면 단일 GPU 메모리에 수용 가능한 것보다 더 큰 컨텍스트를 처리할 수 있습니다.

컨텍스트 병렬화(CP)는 파티션 간 통신을 수행하는 어텐션 구현 이 필요합니다. 이는 레이어별 추가 통신을 발생시킵니다. 따라서 짧은 prompt의 경우 CP는 종종 과도한 오버헤드를 유발합니다. 그러나 매우 긴 입력과 엄격한 TTFT 목표의 경우 CP는 프리필 지연 시간과 GPU당 메모리 사용량을 줄일 수 있습니다. 최대 컨텍스트에 대해 프리필 속도와 정확도를 모두 측정하십시오.

하이브리드 병렬 처리

실제 대규모 MoE LLM 서비스에는 기존 병렬화 전략을 조합하여 사용합니다. 오늘날 LLM 모델은 너무 방대하고 복잡하여 단일 병렬화 방법으로는 충분하지 않습니다. 그림 15-7은 4개의 GPU를 활용한 하이브리드 병렬 구성을 보여줍니다.

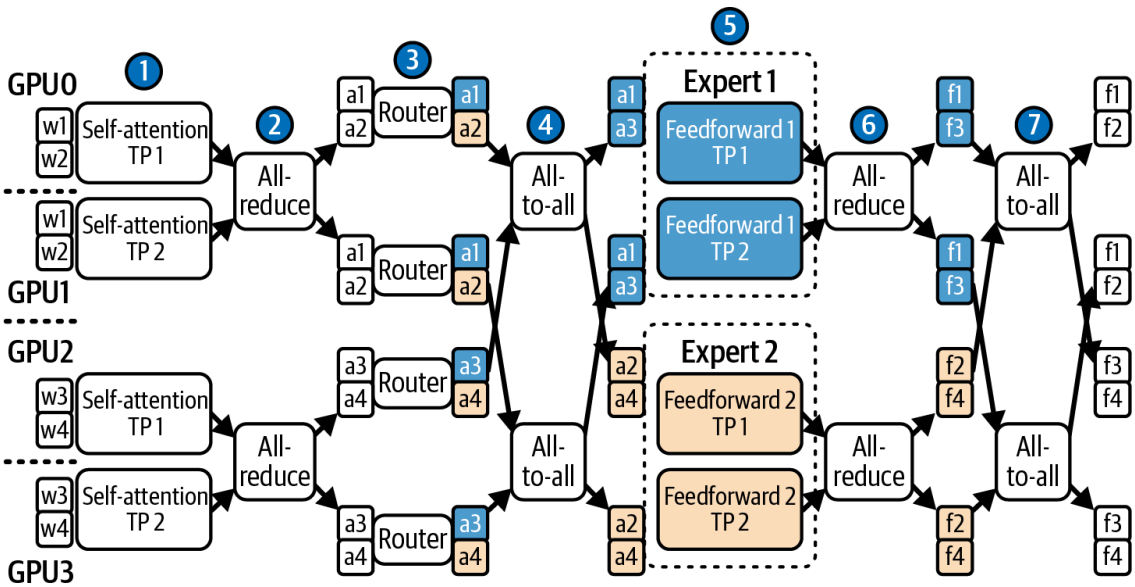

그림 15-7. 4 × 2 × EP 하이브리드 병렬 조합의고수준 다이어그램 (출처: https://oreil.ly/q1AEf)

여기서는 네 개의 파이프라인 단계(GPU당 하나)와 양방향 텐서 병렬화를 사용합니다. 토큰은 전문가 병렬화를 통해 두 명의 전문가로 분배됩니다. 이를 *4×2 EP*

하이브리드 병렬 전략이라 합니다.

더 큰 규모로 확장하여 GPU의 논리적 그룹을 생성해 보겠습니다. 예를 들어 64개의 GPU로 구성된 클러스터를 고려해 보겠습니다. GPU를 각각 4개씩 16개 그룹으로 묶을 수 있습니다. 60개 레이어와 64명의 전문가를 가진 MoE를 사용하면, 각 단계당 15개 레이어로 4-way 파이프라인 병렬화를 적용할 수 있습니다. 이는 모델의 깊이를 분할합니다. 그런 다음 각 단계(15개 레이어) 내에서 2-way TP를 사용하여 각 레이어 내의 무거운 행렬 곱셈을 분할할 수 있습니다. 이는 모델의 너비를 분할합니다.

MoE 레이어의 경우, 상위 2개 게이트 방식을 사용해 그룹당 16명의 전문가를 분산시키는 EP를 적용할 수 있습니다. 이로 인해 토큰은 그룹 내 최대 두 개의 GPU로 전송됩니다. 마지막으로, 데이터 병렬 처리를 통해 이러한 64-GPU 복제본 두 개를 배포하여 시스템 처리량을 두 배로 늘릴 수 있습니다.

이는 하나의 구성 예시일 뿐입니다. 실제 적용 시에는 다양한 병렬화 전략 조합을 실험해야 합니다. 프로파일링 도구를 활용하면 최적의 균형을 찾을 수 있습니다. 특히 Nsight Systems로 엔드투엔드 추적을, Nsight Compute로 커널별 GPU 메트릭을 분석할 수 있습니다.

각 변경 전후로 항상 인터커넥트 트래픽, 텐서 코어 활용도 및 기타 성능 향상 메커니즘을 검증하십시오.

기본 원칙은 TP를 수익 감소 지점까지 활용하는 것입니다—일반적으로 노드 내부 또는 NVL72 섀시 같은 밀접하게 결합된 유닛 내에서 적용합니다. 이후 파이프라인 병렬화는 최소한으로 사용합니다—모델을 메모리에 수용할 수 있을 만큼만 적용합니다. 그런 다음 EP를 극대화하여 MoE 매개변수를 전문가 및 GPU 간에 분산합니다. 마지막으로, 점점 더 많은 최종 사용자와 동시 추론 요청으로 확장함에 따라 처리량을 개선하기 위해 데이터 병렬 복제본을 추가합니다. (매우 긴 입력이 예상되는 경우 선택적으로 CP를 상위에 레이어링할 수 있음).

또한 병렬화를 하드웨어 토폴로지와 일치시켜야 합니다. 예를 들어 NVL72 시스템을 사용할 경우, 72개의 GPU가 모두 NVLink와 NVSwitch를 통해 완전히 연결됩니다. 이 경우 72개 GPU NVLink 도메인 내에서 TP 및 EP 그룹을 구성할 수 있습니다. 그러나 전체 도메인에 근접하는 TP 그룹은 종종 all-reduce 지연 시간으로 인해 반환되는 현상을 보입니다. 따라서 많은 생산 시스템은 NVL72 내부에서도 TP 그룹을 더 작게 유지하고 토폴로지를 고려합니다.

반면, InfiniBand로 연결된 8-GPU 노드 2개로 구성된 소규모 클러스터는 각 노드 내에서만 완전한 NVLink 연결성을 가지며, 노드 간 트래픽은 InfiniBand 패브릭을 통해 이동합니다. 이러한 환경에서는 TP를 각 8-GPU 노드에 국한시키는 것이 가장 좋으며, 가능한 경우 노드 간 병렬 처리를 피하여 노드 간 통신으로 인한 높은 지연 시간과 낮은 대역폭을 방지해야 합니다.

다음 섹션에서는 다중 노드 클러스터에서 더 빠른 추론을 달성하기 위해 이러한 병렬화 전략과 결합할 수 있는 상위 수준의 최적화 기법을 논의합니다. 잘 설계된 서비스 시스템은 이러한 상위 및 하위 수준의 기법들을 다수 결합할 것임을 살펴보게 될 것입니다.

추측적 디코딩 및 병렬 토큰 생성 기법

LLM 추론의 근본적인 성능 과제 중 하나는 디코딩의 순차적 특성입니다. 프리필 단계에서 초기 prompt가 처리된 후 모델은 일반적으로 한 번에 하나의 토큰을 생성한다는 점을 기억하십시오. 각 토큰의 계산은 이전 토큰의 결과에 의존하므로, 이는 본질적으로 직렬 프로세스이기 때문에 병렬화가 어렵습니다. 이는 최신 고속 GPU를 사용하더라도 지연 시간 병목 현상을 유발합니다. 현대 GPU 하드웨어에서 대규모 모델의 경우 수백 개의 토큰을 순차적으로 생성하는 데 몇 초가 소요될 수 있습니다.

이 섹션에서는 디코딩 단계를 가속화하는 여러 기법을 논의합니다. 일부 기법은 추론 단계당 여러 토큰을 생성하는 반면, 다른 기법은 순차적 추론 단계 자체의 수를 줄입니다.

전통적인 추측적 디코딩은 한 배치에 여러 토큰을 제안하는 작고 빠른 "초안" 모델과 각 후보를 검증하는 더 큰 "대상" 모델을 결합합니다. 이는 병렬 생성을 위해 두 번째 추론 패스를 교환하여 더 높은 다중 토큰 처리량을 달성합니다. 그러나 두 개의 별도 모델을 실행하면 배포 복잡성이 증가하며, 검증 패스가 병목 현상이 될 경우 여전히 추론이 지연될 수 있습니다.

메두사(Medusa)는 단일 LLM에 여러 개의 경량 디코딩 헤드 를 직접 연결함으로써 이를 단순화합니다. 각 단계에서 이 헤드들은 트리 기반 어텐션 메커니즘을 사용하여 단일 전방 전달(forward pass) 내에서 여러 토큰 후보를 동시에 생성하고 검증합니다.

메두사의 이 통합 설계는 모델 간 토큰 전송을 피하고 토큰 생성 품질을 저하시키지 않으면서도 상당한 속도 향상을 달성합니다. 초안 생성 및 검증 단계를 단일 모델 패스로 통합함으로써 메두사는 기존의 두 모델을 활용한 추측적 디코딩 기

법보다 개선된 성능을 제공합니다. 이제 이러한 기법들을 자세히 살펴보겠습니다.

두 모델 기반, 초안 기반 추측적 디코딩 및 EAGLE

추측적 디코딩 은 소규모 모델에 추가 작업을 부과하는 대신 비용이 많이 드는 대규모 모델의 처리 시간을 절약하는 기법입니다. 핵심 아이디어는 주 LLM과 병행하여 경량 "초안" LLM을 실행하는 것입니다. 초안 모델은 더 빠르며 현재 컨텍스트를 넘어 *k개* 토큰으로 구성된 배치를 추측적으로 생성합니다.

이후 *대상* 모델이라 불리는 대형 모델은 GPU로 전송된 단일 배치 내 전체 *k-토큰* 시퀀스에 대한 다음 토큰 확률을 예측함으로써 드래프트 토큰을 검증합니다. 대상 모델은 전송된 데이터 바이트당 더 많은 계산을 수행함으로써 산술 집약도를 높여 *k* 토큰 전체를 한 번에 처리합니다. 이로 인해 드래프트 시퀀스 토큰의 효율적이고 빠른 검증이 가능해집니다.

대상 모델의 출력이 초안 모델이 제안한 *k개의* 토큰과 일치한다면, 대규모 모델이 단일 토큰을 생성하는 데 소요되는 동일한 시간 내에 효과적으로 *k개의* 토큰을 생성한 셈입니다. 초안 모델이 생성한 *k개* 토큰 시퀀스 내 어느 토큰에서든 대규모 모델의 검증 결과가 초안 모델의 예측과 일치하지 않을 경우 , 해당 지점 이후의 추측 토큰은 폐기됩니다. 검증 절차는 항상 초안 모델 또는 대상 모델 중 첫 번째 토큰을 수용하므로 각 단계에서 최소한 하나의 생성된 토큰은 유지됩니다.

대상 모델의 수정된 토큰이 사용되고 디코딩이 계속되면, 그 지점에서 새로운 추측적 디코딩 사이클이 시작될 수 있습니다. 그림 15-8은 여러 토큰을 미리 예측하는 소형 초안 모델을 보여줍니다. 이후 대상(대형) 모델이 이 토큰들을 하나씩 검증합니다.

시간이 지남에 따라 추측적 디코딩은 대형 모델이 필요로 하는 일대일 순차 호출의 전체 횟수를 줄입니다. 이론적으로 이는 k× 속도 향상 효과를 제공하는데, 여기서 *k*는 초안 모델이 생성한 토큰 수입니다. 실제로는 오버헤드와 가끔 발생하는 추측적 토큰 거부로 인해 이득은 약 2× 속도 향상 수준입니다.

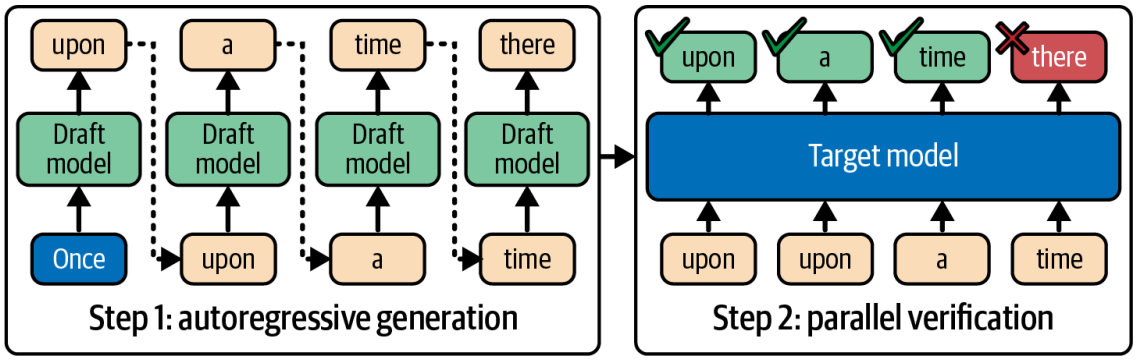

그림 15-8. 디코딩용 초안(소형) 모델과 다중 토큰 검증용 대상(대형) 모델을 활용한추측 디코딩

초안 모델은 대형 모델의 분포에 대해 상당히 높은 정확도를 갖도록 선택되어야 합니다. 즉, 초안 모델의 예측은 대형 모델의 예상 출력과 높은 중첩성을 가져야 합니다. 초안 모델이 자주 잘못 예측하면 추측적 디코딩은 거의 이점을 제공하지 못하고 컴퓨팅 자원을 낭비합니다. 대형 대상 모델이 예측하지 않을 토큰을 자주 예측하면 많은 추측적 토큰이 거부됩니다. 이는 컴퓨팅 시간을 낭비하게 됩니다.

추측적 디코딩이 실질적인 성능 향상을 제공하려면 드래프트 모델이 대형 모델보다 훨씬 빠르며(일반적으로 4배 이상), 주요 모델의 증류 버전이나 동일한 데이터로 미세 조정되어 출력 간 상관관계가 우수한 소형 모델을 사용하는 것이 일반적인 전략입니다.

초안 모델은 대형 모델과 동일한 토큰화기 및 어휘를 사용해야 합니다. 이 점이 간과되는 경우가 있으며, 이는 저조한 결과로 이어집니다.

초안 모델은 동일한 prompt와 더 높은 온도(temperature)를 사용하여 토큰을 생성함으로써 대형 모델의 가능한 연속어 중 하나와 일치할 확률을 높입니다. 한편, 대상 모델은 앞으로 건너뛰어 모든 초안 토큰을 단일 배치로 처리합니다.

표준 추측적 디코딩 수용 절차 하에서는 대상 모델의 출력 분포가 보존됩니다. 따라서 샘플링 설정이 일치할 경우 최종 샘플은 대형 모델의 분포와 일치합니다. 초안이 분기될 경우 대상 모델의 검증 단계에서 해당 토큰을 거부하여 정확성이 유지됩니다. 유일한 차이는 소형 모델이 이러한 추측적 토큰에 대해 정확히 예측할 때 *최대 k회까지* 대상 모델 호출이 생략된다는 점입니다.

추측적 디코딩은 다양한 방식으로 구현될 수 있습니다. vLLM, SGLang, TensorRT-LLM과 같은 대부분의 현대적 LLM 추론 엔진은 초안 및 대상 모델 생성을 조정하는 내장 지원을 갖추고 있습니다. 경험적으로 약 1.5~2.5배의 속도 향상이 일반적이며, 신중한 배치 및 초안 모델 설계를 통해 더 높은 이득이 가능합니다.

PyTorch에서 `torch.nn.functional.scaled_dot_product_attention` 연산은 장치 세대, 형상, 마스크, dtype에 따라 최적의 백엔드(FlashAttention 또는 cuDNN 커널)를 자동 선택합니다. `torch.nn.attendion.sdpa_kernel()`를 사용해 명시적인 `SDPABackend` 유형으로 백엔드를 고정할 수 있습니다. 예를 들어 LLM 디코딩 벤치마킹 시 융합된 백엔드가 활성화되었는지 확인하는 것이 중요합니다.

예를 들어, EAGLE(Extrapolation Algorithm for Greater LLM Efficiency) 알고리즘 은 토큰 수준이 아닌 특징 수준에서 작동함으로써 추측적 디코딩을 재고합니다. EAGLE은 대규모 모델 자체의 중간 표현을 한 단계 추론하여 다음 토큰의 특징을 예측합니다. 이는 불확실성을 해소하고 더 높은 수용률을 달성합니다.

EAGLE은 4토큰 초안에서 기본 디코딩 대비 최대 약 3.5배의 속도 향상을 보였으며, 대규모 모델의 출력 분포를 유지했습니다. EAGLE은 유리한 환경에서 초안 깊이의 이론적 한계에 근접합니다. 이는 혁신적인 기법을 통해 추측 디코딩이 디코딩 시간을 더욱 단축하면서도 출력 품질을 동시에 유지할 수 있음을 보여줍니다.

EAGLE-2는 가능성들을 나타내는 컨텍스트 인식 동적 *드래프트 트리*를 도입하여 EAGLE을 확장합니다. 일부 평가에서 EAGLE이 기본 디코딩 대비 최대 3.5배의 속도 향상을 달성한 반면, EAGLE-2의 접근법은 작업과 모델에 따라 EAGLE보다 약 20~40% 빠른 속도 향상을 보고했습니다. EAGLE-2에서는 초안 모델이 분기되는 토큰 시퀀스 집합을 생성하여 검증 단계의 병렬성을 더욱 높입니다. 이는 그림 15-9에 표시되어 있습니다.

EAGLE-3은 직접 토큰 예측을 선호하고 다층 특징을 융합함으로써 이전 버전인 EAGLE-1 및 EAGLE-2를 계속 개선합니다. EAGLE-3은 특정 작업에서 EAGLE-2 대비 최대 1.4배 개선을 보고하며, 최적화되지 않은 기준 변형 대비 최대 6.5배의 속도 향상을 보입니다. EAGLE-1과 EAGLE-2에서는 초안 모델이 내부 특징 벡터를 예측한 후 이를 토큰으로 디코딩합니다. 이러한 초기 EAGLE 방법은 본질적으로 내부 특징을 추측한 후 이를 토큰에 매핑하는 방식으로 작동했습니다.

EAGLE-3은 특징 수준 예측 단계를 생략하고 초안 토큰을 보다 직접적으로 예측합니다. 다만 여전히 내부 특징을 사용하지만, 다중 레이어 표현(하위, 중간, 상위)으로 융합하여 초안 예측을 안내합니다. 이는 최상위 레이어만 사용하는 방식과 대비됩니다. 이로 인해 EAGLE-3은 더 간소화되고 제약이 적어 확장성이 향상되었습니다.

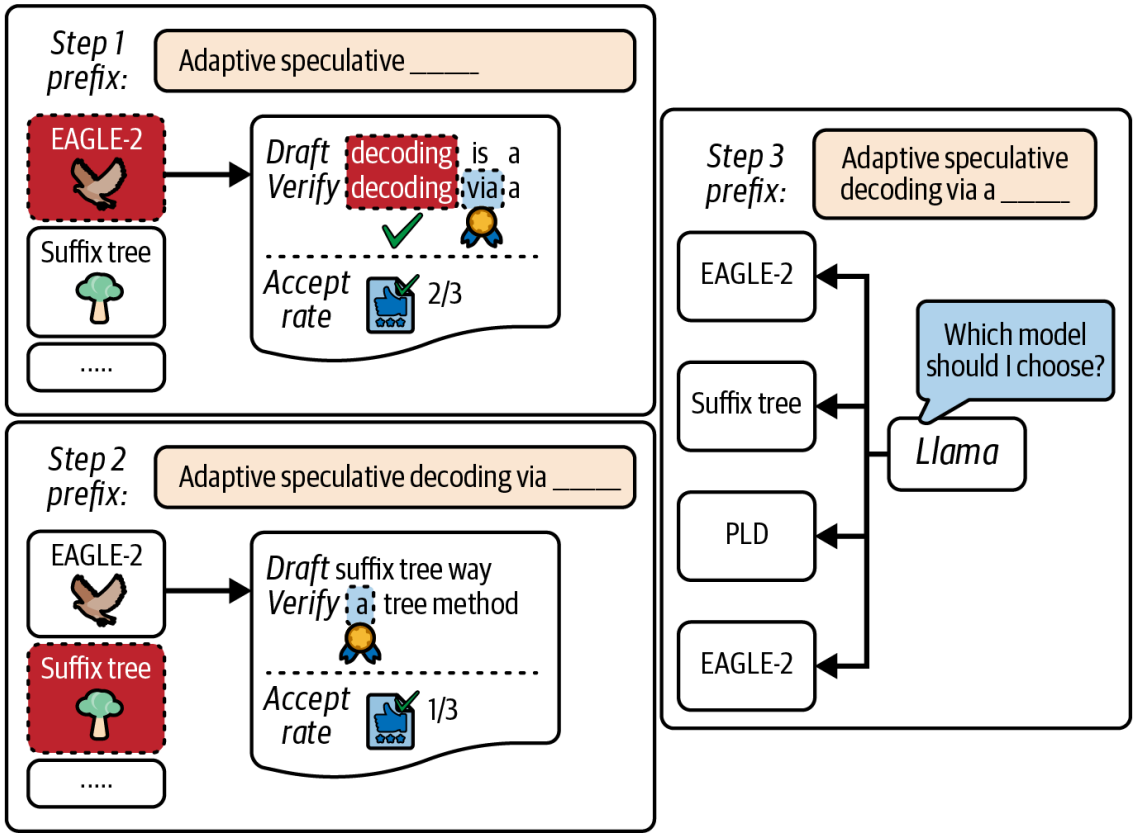

그림 15-9. EAGLE-2를 이용한추측적 디코딩 (출처: https://oreil.ly/uG07b)

또 다른 기법은 *동적 깊이 디코딩으로*, 출력 품질에 미치는 영향을 최소화하면서 적응적으로 레이어를 건너뛸 수 있습니다. 계산량을 줄이는 다른 기법으로는 *N번째*트랜스포머 레이어를 건너뛰기, 초안 모델에 낮은 정밀도(예: FFP8 및 NVFP4) 사용, 초안 단계에서 일시적으로 더 작은 히든 크기를 사용하는 것 등이 있습니다.

일부 연구용 프로토타입은 정확도를 속도와 교환하기 위해 디코딩 중 네트워크 일부를 건너뛰거나 정밀도를 낮추는 모드를 제공합니다. 현재 시점에서 이러한 모드는 생산 모델에 보편적으로 적용되지 않으므로, 해당 모드를 활성화하기 전에 항상 대상 작업에서 품질과 속도를 검증해야 합니다.

향후 LLMs에는 최적화된 '빠른 생성(fast-generation)' 모드가 포함될 수 있습니다. 예를 들어, 모델에 대체 경량 레이어나 정밀도 감소 디코딩 구성을 내장할 수 있습니다. 이러한 내장 최적화를 통해 모델은 계산을 건너뛸 수 있습니다.

단일 모델 자기 추측적 디코딩

추측적 디코딩 에 대한 또 다른 접근법은 외부 초안 모델 사용을 완전히 배제하고, 더 큰 대상 모델이 일부 계산을 선택적으로 생략함으로써 자체 출력을 초안화하고 검증하는 것입니다. 이러한 방법 중 하나가 자체 추측적 디코딩(self-specu-

lative decoding)으로, 초안화-검증 방식(draft-and-verify scheme)이라고도 합니다.

자기 추측적 디코딩에서는 대규모 대상 모델이 빠른 근사 패스를 통해 *k개의* 토큰을 생성합니다. 예를 들어, 새로운 토큰마다 레이어의 절반만 실행하도록 선택할 수 있으며(정밀도 감소 가능), 나머지 레이어는 건너뜁니다. 이는 소형 초안 모델과 유사한 초안 출력을 생성합니다. 수용률과 초안 깊이가 유리할 경우 유사한 속도 향상을 달성할 수 있습니다. 이는 소형 초안 모델과 유사한 초안 출력을 생성하며, 비슷한 2배 속도 향상도 달성합니다.

그런 다음, 자기 투기적 디코딩의 두 번째 단계에서 대상 모델은 한 번에 *k개의* 토큰을 검증하기 위해 완전한 전방 패스를 수행합니다. 모두 일치하면 해당 토큰들에 대해 절반의 레이어 실행을 절약합니다. 일치하지 않으면 단순히 기존 투기적 접근 방식으로 되돌아가 불일치 지점까지 토큰을 수락하고 정상적으로 진행합니다.

자체 추측적 디코딩에서는 동일한 모델이 초안 생성 및 검증 작업을 모두 수행하므로 별도의 모델을 훈련하거나 유지 관리하거나 런타임에 메모리에 로드할 필요가 없습니다. 핵심 과제는 정확도를 지나치게 저하시키지 않으면서 초안 단계에서 필요한 연산량을 줄이는(레이어 제거, 정밀도 감소 등) 효과적인 방법을 찾는 것입니다.

관련 기법으로 *일관성 있는 디코딩이* 있는데, 이는 하나의 LLM을 훈련시켜 여러 토큰을 생성하고 검증하도록 하는 방식입니다. 이 단일 모델 접근법은 별도의 초안 모델 없이 약 3배의 속도 향상을 가져옵니다. 이는 추측적 디코딩을 모델 자체의 가중치에 내재화하는 추세를 보여줍니다.

이러한 방법들은 매우 활발한 연구 분야를 대표합니다. 특히 모델이 내재된 내부 중복성을 활용해 스스로를 가속화할 수 있게 한다는 점에서 흥미롭고 유망합니다. 또한 최적화가 모델 내부에서 이루어지기 때문에 추론 엔진 구현을 단순화할 수 있습니다.

메두사의 다중 헤드를 활용한 멀티토큰 디코딩

추측적 디코딩은 여전히 궁극적으로 초안 모델을 사용해 토큰을 하나씩 생성합니다—단, 초안 모델이 더 작아져 속도가 빨라질 뿐입니다. 그러나 메두사 프레임워크는 더 급진적인 접근법을 취합니다. 모델 아키텍처 자체를 수정하여 각 디코딩 단계마다 여러 개의 새 토큰을 병렬로 예측하도록 합니다.

두 모델을 사용하는 추측적 디코딩이 여전히 토큰을 하나씩 생성하는 것(비록 더 빠르긴 하지만)과 달리, 메두사의 아키텍처는 단일 모델에서 반복당 여러 토큰을 진정으로 생성합니다. 따라서 메두사의 다중 헤드 접근법은 메두사-1과 메두사-2 모두에 걸쳐 발표된 실험에서 약 2.2~3.6배의 속도 향상을 보고했습니다.

그러나 메두사는 특정 레이어에서 분기되는 추가 디코더 헤드로 트랜스포머 기반 LLM을 수정하기 때문에 맞춤형 모델 훈련이 필요합니다. 이 때문에 *메두사라는* 이름이 붙었습니다. 이를 통해 모델은 여러 다음 토큰을 동시에 제안할 수 있습니다.

서로 다른 메두사 헤드가 생성한 다중 토큰 후보는 트리 구조로 구성됩니다. 예를 들어 메두사는 한 번의 패스로 깊이 2의 이진 트리를 생성하여 최대 4개의 토큰을 생산할 수 있습니다. 이후 그림 15-10과 같이 다중 토큰 시퀀스를 병렬로 검증합니다.

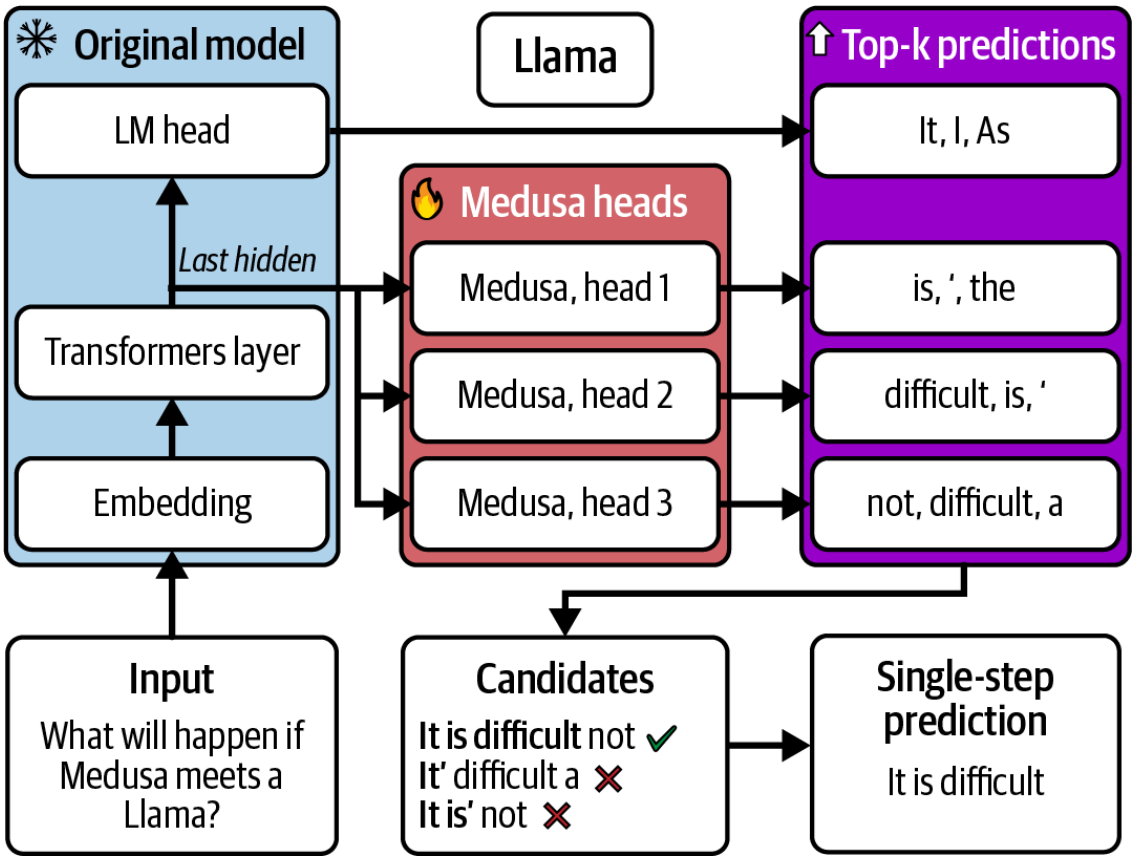

그림 15-10. Medusa를 이용한다중 토큰 디코딩 (출처: *https://oreil.ly/MJMOQ*)

내부적으로 메두사는 병렬 토큰 예측 간의 일관성을 높이기 위해 특수한 트리 기반 어텐션 패턴을 사용합니다. 모델은 다중 토큰 시퀀스를 동시에 확장하고 검증하는 법을 학습합니다. 메두사를 통해 LLM은 본질적으로 몇 개의 토큰을 '미리 예측'하는 법을 학습하고, 이를 한 번에 모두 출력합니다.

추론 과정에서 메두사는 병렬 예측 횟수에 비례하여 순차적 디코딩 반복 횟수를 줄일 수 있습니다. 예를 들어, 메두사가 한 번의 패스로 4개의 토큰을 예측하면,

깊이 2의 이진 트리 출력 시 필요한 전방 패스 횟수를 최대 4배까지 줄일 수 있습니다.

그러나 실제로는 예측 분기가 검증에 실패하여 부분 재실행이 필요한 경우 가끔 백트래킹해야 하는 오버헤드로 인해 메두사는 일반적으로 약 2~3배의 속도 향상을 달성합니다. 이는 추측적 디코딩 검증 및 거부 시 발생하는 오버헤드와 유사합니다. 예를 들어, 메두사 LLM이 반복당 4개의 토큰을 생성한다면, 100개 토큰의 응답은 100번이 아닌 25번의 모델 반복만으로 처리됩니다.

이러한 추가 헤드를 추가하려면 모델을 수정하고 재훈련해야 합니다. 또한 추가 헤드 매개변수로 인해 메두사 모델의 크기가 약간 더 큽니다. 이는 개발 복잡성과 훈련 비용을 다소 증가시킵니다. 그러나 일단 훈련되면 메두사 모델은 추론 시간을 크게 단축할 수 있습니다.

사용 사례에 맞게 훈련된 경우, 메두사(Medusa)를 지원하는 LLM은 우수한 저지연 성능을 제공할 수 있습니다. 그러나 이 기술은 추가 헤드를 추가하기 위해 모델 아키텍처를 수정하고 미세 조정해야 합니다.

여러 요청의 디코딩 단계 병렬 처리

또 다른 병렬 디코딩 기법 은 여러 최종 사용자의 동시 요청에서 디코딩 단계를 교차 배치하는 것입니다. 이 병렬 처리는 알고리즘적 기법이나 모델 아키텍처 수정이라기보다 추론 엔진의 기능에 가깝습니다. 하지만 토큰 단위 배치와 최종 사용자 요청 간 배치를 통해 GPU를 지속적으로 활용하는 데 도움이 됩니다.

vLLM과 같은 프레임워크는 요청 라우팅, 연속 배치, 토큰 스케줄링 형태로 이를 구현합니다. 핵심은 순차적 단계 사이의 공백을 채워 GPU를 지속적으로 활용하는 것입니다. 구체적으로, I/O 대기나 데이터 종속성으로 인해 GPU가 정지된 경우, 스케줄러는 해당 GPU에서 다른 시퀀스의 다음 토큰 예측을 동시에 실행할 수 있습니다.

추론 엔진의 고급 라우팅, 배치 처리, 스케줄링 기능을 CUDA 스트림과 결합하세요. 그런 다음 Nsight Systems로 토큰 단계 커널이 단일 스트림에서 직렬화되지 않고 NIC/NVLink 전송과 중첩되는지 확인하세요. 그러면 애플리케이션, 시스템, 하드웨어 수준에서 대규모 병렬 처리를 달성할 수 있습니다.

중요한 점은 디코딩 단계를 인터리빙해도 단일 시퀀스의 지연 시간은 단축되지 않는다는 것입니다. 오히려 추가적인 컨텍스트 전환으로 인해 토큰당 아주 작은

오버헤드가 발생할 수도 있습니다. 그러나 다수의 사용자를 동시에 서비스할 때 전체 처리량과 GPU 활용도를 크게 향상시킬 수 있습니다. 이는 고부하 상태에서 최종 사용자의 평균 요청 지연 시간을 줄여줍니다.

디코딩 기법 결합 및 복잡도 평가

이러한 디코딩 최적화 기법 은 결합할 수 있다는 점이 주목할 만합니다. 예를 들어, 스펙큘러 디코딩을 메두사 지원 대상 모델과 함께 사용할 수 있습니다. 이때 대형 대상 모델은 소형 모델이 예측한 여러 토큰을 동시에 검증합니다.

이러한 기법은 추가적인 복잡성을 수반합니다. 별도의 모델을 유지하거나, 주 모델을 수정하거나, 더 정교한 제어 로직을 관리해야 합니다. 실제 운영 환경에서는 성능 향상 대비 이 복잡성을 평가해야 합니다. 대화형 애플리케이션의 경우, 특히 대규모 환경에서 응답 시간을 단 몇 밀리초라도 줄이는 것은 복잡성을 감수할 만한 가치가 있습니다.

요약하면, 스펙츄레이티브 디코딩(초안 모델 유무에 관계없이) 및 메두사 스타일 다중 토큰 예측과 같은 고급 디코딩 기법은 전통적으로 한 번에 하나의 토큰을 생성하는 현대적 자동 회귀 LLMs의 전체 추론 시간을 단축할 수 있습니다. 한 번에 더 많은 토큰을 생성하고, 전체 연산 집약도를 높이며, 디코딩 단계를 컴퓨팅 병목 영역으로 이동시킴으로써 GPU의 극한 컴퓨팅 성능을 활용할 수 있습니다.

디코딩 최적화 환경은 지속적으로 진화하며 복잡성이 증가하고 있습니다. 추세는 명확합니다: LLM 디코딩을 더 병렬적이고 효율적으로 만드는 것입니다.

제약 조건 하의 디코딩 성능 영향

의 LLM 디코딩 과정에서 별개이지만 중요한 측면은 특정 제약 조건 하에서 텍스트를 생성하는 것입니다. 예를 들어, 모델은 출력이 사전 정의된 형식(JSON)을 따르도록 강제하거나, 특정 문법을 사용하도록 하거나, 안전을 위해 특정 시퀀스를 허용하지 않도록 할 수 있습니다. 따라서 *구조화된 출력이라고도 불리는* 제약 조건 하 디코딩은 생산 환경에서 종종 필요합니다.

예를 들어 OpenAI의 함수 호출 API는 호출할 함수와 해당 함수의 입력 및 출력을 결정하기 위해 잘 구조화된 출력 형식에 의존합니다. 이러한 제약 조건은 함수의 API 시그니처와 일치하도록 설계되었으며 애플리케이션 수준에서는 단순히 타협할 수 없는 요소입니다.

제약된 디코딩은 모델의 토큰 선택 과정을 변경하여 각 단계에서 유효한 토큰만 허용합니다. 이는 허용된 토큰 목록을 제공하는 것만큼 간단할 수도 있고, 공식적인 문법을 내장하고 상태 머신을 사용하여 무효 토큰을 걸러내는 것만큼 정교할 수도 있습니다.

제약 디코딩의 단점은 이러한 제약을 적용하는 데 추가 지연 시간이 필요하다는 점입니다. 일반적으로 토큰당 수 밀리초가 소요됩니다. 이는 모델이 반복적으로 백트래킹을 수행하고, 정제된 어휘를 탐색하며, 최종적으로 유효한 토큰을 생성해야 하기 때문입니다. 백트래킹은 LLM의 생성 루프 내부에서 발생합니다. 따라서 모델 자체에 세분화된 텔레메트리 데이터가 없으면 제약 디코딩의 디버깅, 프로파일링 및 튜닝이 어려울 수 있습니다.

또 다른 성능 고려 사항은 캐시 효율성입니다. 디코딩 단계마다 전체 확률 분포를 전수 검색하면 캐시가 과부하되어 처리량이 더욱 감소할 수 있습니다. 허용 토큰 집합이 크게 제한될 때 특히 두드러집니다.

이러한 비용을 완화하기 위해 많은 프레임워크는 JSON 문법을 컴파일하고 각 상태에 대해 유효한 토큰을 사전 계산합니다. 이를 통해 추론 엔진은 런타임에 무효한 소프트맥스 출력을 가릴 수 있습니다. 이 접근법은 백트래킹을 줄이고 캐싱을 개선하며 수용률을 높입니다.

단순한 JSON 스키마와 같은 중간 수준의 제약 조건에서는, 컴파일된 문법을 사용하고 GPU 실행과 마스크 계산을 중첩하는 현대적 엔진이 대규모 환경에서 오버헤드를 한 자릿수 초반 범위로 낮출 수 있습니다. 그러나 복잡한 문법이나 소규모 배치에서는 여전히 두 자릿수 오버헤드가 발생할 수 있습니다. 정확한 스키마 하에서 TTFT(첫 번째 응답 시간)와 초당 토큰 수를 항상 측정하십시오.

이러한 기법들은 Hugging Face Transformers, NVIDIA NeMo, vLLM 등 인기 라이브러리 및 추론 엔진에 구현되었습니다( ). 예를 들어 NVIDIA의 TensorRT-LLM과 Hugging Face Transformers는 모두 사용자가 정의한 어휘 마스크를 통해 속도 저하를 거의 유발하지 않으면서 제약 조건을 적용할 수 있습니다. 특히 TensorRT-LLM은 XGrammar 백엔드를 통해 JSON 스키마 및 문맥 자유 문법(CFG)을 지원하는 가이드 디코딩 기능을 제공합니다. 또한 vLLM과 Transformers는 구조화된 출력 API를 제공합니다.

TensorRT-LLM의 가이드 디코딩을 사용할 경우, XGrammar는 GPU 상에서 JSON 스키마 및 CFG(문법) 지원을 제공합니다. XGrammar는 제약 조건을 컴파일하여 Python 측의 대규모 토큰 마스크 오버헤드를 방지합니다. 그러나 특정 구성에서는 느린 백엔드로 전환되어 첫 요청 지연이 과도하게 발생할 수 있음을 유의하십시오. 따라서 캐시 로컬리티와 토큰 마스크 대역폭을 유지하기 위해 문법을 간결하게 유지하는 것이 중요합니다.

또 다른 인기 있는 전략은 제약 디코딩을 커널 자체에 통합하는 것입니다. 이 경우 최종 소프트맥스 단계에서 토큰 마스크를 주입하여 유사한 성능 향상을 달성합니다.

효율적으로 구현될 경우, 제약 디코딩은 종종 제약 없는 디코딩만큼 빠르게 실행될 수 있습니다—특히 제약 규칙 세트가 너무 크거나 지나치게 제한적이지 않은 경우에 그렇습니다. 디코딩 제약이 지나치게 제한적이라면, 디코딩은 사실상 작은 토큰 공간을 탐색하는 작업이 됩니다. 이는 더 많은 백트래킹과 느린 토큰 생성을 초래합니다.

가능한 경우 문법, 형식, 어휘집과 같은 크거나 지나치게 제한적인 제약 조건은 피하십시오. 한 가지 방법은 LLM이 정상적으로 디코딩하도록 한 후 출력을 후처리하거나 필터링하는 것입니다. 그러나 이는 항상 성능이 우수하거나 실행 가능한 옵션은 아닙니다. 다양한 옵션을 자질하여 사용 사례에 가장 적합한 것을 선택하십시오.

MoE 추론을 위한 동적 라우팅 전략

MoE 모델을 효율적으로 서비스하려면 전문가 병렬 처리(expert parallelism)를 활용해 GPU 간 전문가 분할을 신중하게 수행해야 합니다( ). 또한 런타임에 토큰을 해당 전문가들에게 동적으로 라우팅하기 위한 지능적이고 동적인 메커니즘, 즉 게이트 네트워크가 필요합니다.

MoE 추론 중 각 토큰의 전방 전달은 게이팅 네트워크에 의해 하나 이상의 전문가 GPU로 라우팅되어야 합니다. 시스템은 GPU를 바쁘게 유지하고 과부하 GPU(핫스팟)를 피하기 위해 이 통신을 균형 있게 처리해야 합니다. 대규모 다중 노드 추론 환경에서 이를 해결하는 방법을 살펴보겠습니다.

전문가 간 통신 최적화

전문가 간 토큰 활성화 의 전수 교환 과정에서 MoE는 GPU 간 토큰 배치를 셔플합니다. 각 GPU는 호스팅하는 전문가에 필요한 토큰만 수신하며, 이는 그림 15-

11에 표시되어 있습니다. 이 과정은 MoE의 모든 레이어에서 발생하며 비용이 많이 드는 작업으로, 효율적으로 처리되지 않으면 추론 시간을 지배할 수 있습니다.

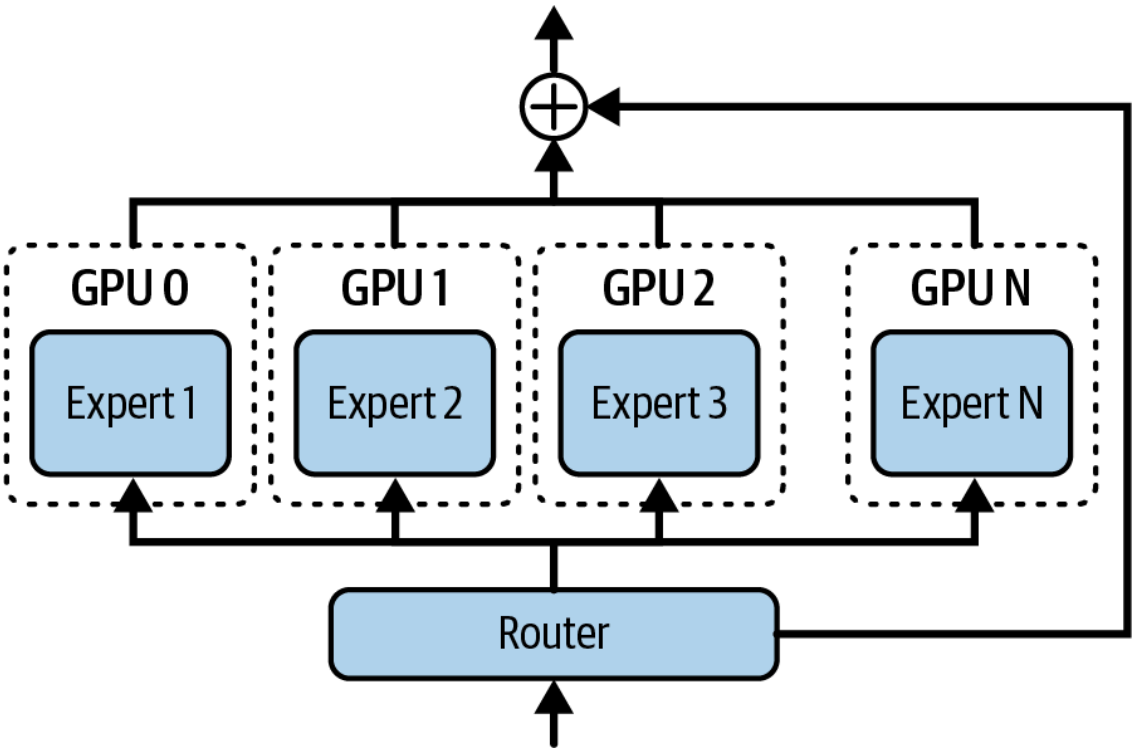

그림 15-11. 각 GPU는 호스팅하는 전문가에게 필요한 토큰만 수신

통신 오버헤드를 줄이는 한 가지 전략은 GPU 클러스터에 계층적 라우팅 전략을 적용하는 것입니다. 먼저 NVSwitch/NVLink(고속)를 사용하여 동일 노드 내 GPU 간 토큰을 라우팅하고, 비로컬 전문가가 필요한 잔여 토큰에 대해서만 노드 간 라우팅을 수행합니다. 이 2단계 전-전(all-to-all) 방식은 노드 간 트래픽 양을 줄일 수 있습니다.

또한, 통신과 계산을 중첩시켜 비동기 통신을 사용할 수 있습니다. 고성능 MoE 추론 서버는 이중 버퍼 통신을 사용하여 한 배치의 토큰이 전송되는 동안 이전 배치의 전문가 계산이 병렬로 수행되도록 합니다. 이러한 파이프라인화는 통신 지연 시간의 상당 부분을 숨깁니다.

노드 간 패브릭을 완전히 활용하면서 노드 내 셔플을 백그라운드에서 실행할 경우, 거의 최적에 가까운 전수 완료 시간을 달성할 수 있습니다. 노드 간 NIC 경로와 노드 내 NVLink 경로 모두에서 링크 활용도를 반드시 측정하십시오.

단일 글로벌 전역적 전수 통신 장벽을 사용하는 단순한 전문가 라우팅 구현은 GPU가 동기화 오버헤드로 대기하게 할 수 있습니다. 이는 모든 링크가 최적화되지 않을 경우 대역폭을 낭비합니다.

버터플라이 스케줄(일명 *시프트된 전수대응 스케줄*)과 같은 기법은 통신을 단계별 라운드로 분할할 수 있습니다. 이렇게 하면 모든 NVLink/NIC가 하나의 대규모 동기화가 아닌 부분적 교환으로 바쁘게 작동합니다. 이 단계적 접근 방식은 링크 활용도를 높이고 유휴 시간을 줄입니다.

전수간 교환은 내장된 전수수신(`ncclAllToAll`) 집단 전송 또는 그룹화된 송수신 호출을 사용할 수 있습니다. 교환을 분할 및 파이프라인 처리하거나 버터플라이와 같은 계층적 스케줄을 노드 간에 적용하면 처리량이 향상되는 경우가 많습니다. 토폴로지에 맞는 알고리즘을 검증하고 선택하십시오.

1단계 전-대-전(all-to-all) 통신은 랙 내(NVLink)로 유지하고 잔여 토큰에 대해서만 랙 간(inter-rack) 스필을 사용하십시오. NVLink 셔플이 백그라운드에서 실행되는 동안 NIC가 포화 상태인지 확인하기 위해 링크 자질을 자질하십시오. 또한 노드 간 링크가 병목 현상인 경우, 전문가 계산과 함께 MoE 전-대-전 통신을 더블 버퍼링하는 것이 권장됩니다. 챕킹/파이프라인 교환 및 버터플라이/시프트 스케줄을 활용하면 글로벌 배리어 지연을 피하고 부동소수점 글로벌 콜렉티브보다 우수한 성능을 낼 수 있습니다.

통신 트래픽을 줄이는 또 다른 해결책은 *전문가 콜로케이션(expert collocation)* 입니다 . 특정 전문가들을 동일한 GPU 또는 노드에 배치하여 불필요한 통신을 피하는 *개념입니다* . 토큰 라우터에 의해 동일한 토큰에 대해 자주 활성화되는 전문가 5와 7을 예로 들어보겠습니다. 전문가 5와 7을 동일한 GPU에 배치하면 추가적인 전수 통신 홉을 제거할 수 있습니다. 자질 도구와 게이트 주파수 분석을 통해 작업 부하에 적합한 이러한 페어링을 식별할 수 있습니다.

또 다른 해결책은 전문가 간 통신(전문가 교환 활성화 포함)을 압축하는 것입니다. 예를 들어, 전량 통신을 수행하기 전에 NVIDIA Transformer Engine을 사용해 텐서 코어에서 FP8 또는 NVFP4로 캐스팅할 수 있습니다. 이는 압축 계산 비용을 상쇄하는 NIC 부하를 줄여줍니다. 이는 GPU 간 활성화 전송 속도 향상을 위해 수치 정밀도를 아주 약간 희생하는 방식입니다. 캐스팅 및 팩/언팩 오버헤드는 일반적으로 네트워크 및 메모리 전송 비용에 비해 작습니다.

요약하면, MoE 통신 최적화는 하드웨어 및 네트워크 토폴로지를 분석하여 GPU에 전문가 배치 위치를 최적화하는 것을 포함합니다. 배포 시 효율적인 전수 통신을 위해 클러스터의 상호 연결을 구성하는 것이 중요합니다. 예를 들어, NVL72 랙에서 NVLink 스위치 메시를 사용하면 단일 도메인 내 최대 72개 GPU 간 전체 대역폭 통신을 확보할 수 있습니다. 단순한 전수 통신 선택은 레이어 시간을 지배하고 매우 낮은 SM 효율을 초래할 수 있습니다. 가능한 경우 전문가 트래픽을 우

선순위화하고 중첩시키며, 서로 다른 상호 연결 및 통신 알고리즘 선택 시 자질과 검증을 반드시 수행하십시오.

MoE의 경우, 클러스터 를 올투올 통신에 최적화하도록 구성하십시오. 이는 토폴로지에 적합한 NCCL 올투올 알고리즘 또는 그룹화된 송수신 구현을 선택한 후, 노드 간 경로에 GPUDirect RDMA가 활성화되었는지 확인하는 것을 의미합니다. 또한 InfiniBand 링크가 여러 물리적 링크(포트)를 단일 논리 채널로 구성하도록 적절히 결합되었는지 확인하십시오. 대역폭이 결합되고 장애 조치 시 원활하게 전환되어야 합니다. 즉, 네트워크 토폴로지가 MoE에 최적화되었는지 확인하십시오. 여기에는 스택의 하드웨어 및 소프트웨어 계층 모두 포함됩니다.

부하 분산, 용량 계수 및 전문가 복제

"핫스팟"을 방지하려면 각 전문가 와 GPU가 작업을 균등하게 분배받는 것이 좋습니다. 그렇지 않으면 MoE의 게팅 네트워크가 특정 전문가에게 불균형적으로 많은 토큰을 할당할 경우 해당 GPU들이 과부하 상태가 됩니다. 이는 지연 시간을 증가시키고 전체 시스템 처리량을 저하시킵니다.

다른 전문가 GPU들의 평균 활용률이 60% 정도인 반면, 단일 전문가 GPU가 활용률 99%에 달하는 핫스팟이 되는 상황을 고려해 보십시오. 이 문제는 적절히 처리되지 않으면 전체 훈련 또는 추론 클러스터의 병목 현상을 유발할 수 있습니다.

모델 훈련 중 핫스팟은 일부 전문가를 과도하게 사용하고 다른 전문가를 부족하게 사용할 경우 게이트에 페널티를 부과하는 부하 분산 손실 항을 추가하여 해결할 수 있습니다. 그 결과 훈련된 MoE 모델은 토큰을 전문가들 사이에 상당히 균등하게 분배하는 경향이 있습니다.

그러나 추론 시 특정 입력 prompt나 주제가 일부 '핫' 전문가에 집중되어 불균형을 유발할 수 있습니다. 추론 핫스팟을 방지하는 한 가지 전략은 오버플로우 메커니즘을 트리거하는 *용량 계수*를 사용하는 것입니다.

용량 계수를 지정함으로써 모델은 각 전문가가 주어진 시간에 최대 토큰 수(예: 32개 토큰)만 처리하도록 구성될 수 있습니다. 전문가가 이 용량 이상의 토큰을 수신하면 초과 토큰은 다음으로 높은 라우팅 점수를 가진 대체 전문가로 전달되거나, 토큰이 직렬화되어 두 번째 패스에서 처리됩니다. 그림 15-12는 용량 계수 1.0과 1.5를 비교합니다.

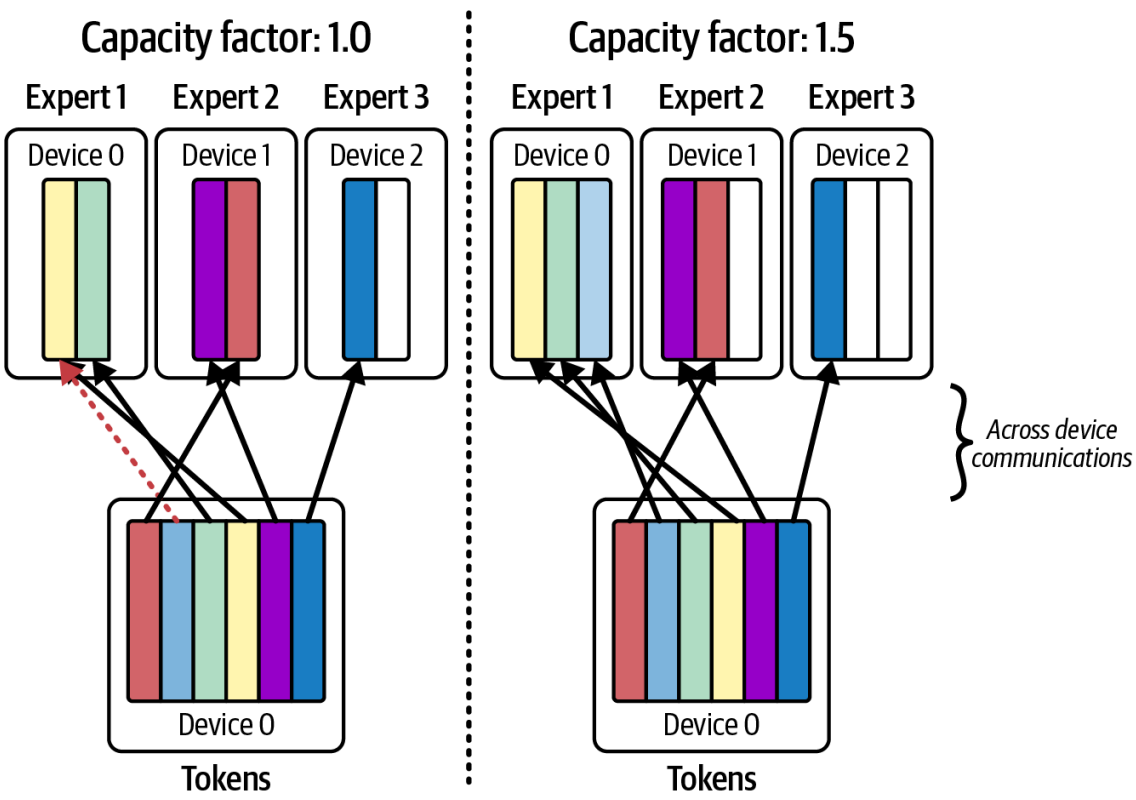

그림 15-12. 전문가 용량 계수 1.0 대 1.5비교

실제 적용 시 상위 2개 전문가 선택 방식과 함께 용량 계수 1.2(20% 오버플로 허용)가 흔히 사용됩니다. 이는 각 전문가가 평균 부하의 최대 120%까지 처리한다는 의미입니다. 이후 초과 토큰은 다음 전문가로 전송됩니다. 이를 통해 시스템 내 전문가 간 부하가 효과적으로 분산됩니다.

핫스팟 을 방지하는 또 다른 전략은 전문가 복제입니다. 특정 전문가가 지속적으로 핫스팟이 되면, 해당 전문가를 다른 GPU에 복제할 수 있습니다. 이렇게 하면 게팅 함수가 토큰의 일부를 다른 GPU에서 실행 중인 전문가 복제본으로 보낼 수 있습니다.

복제본은 추론 엔진에서 구현하는 순수한 애플리케이션 수준 최적화입니다. 모델 자체는 복제본을 인식하지 못합니다. 복제본은 별도의 인덱스를 가진 독립적인 전문가로 등록되며(종종 다른 GPU에 배치됨), 엔진은 상대적 라우팅 점수에 따라 원본 전문가와 복제본 모두에 토큰을 분배할 수 있습니다.

전문가 복제는 각 복제된 전문가의 메모리 및 비용을 증가시킵니다. 그러나 일반적으로 소수의 전문가만 과부하되는 경향이 있으므로, 소수의 핫 전문가만 복제하는 것은 표적화된 해결책입니다. 전체 모델과 모든 전문가를 복제하여 비용을 두 배로 늘리는 것과는 다릅니다.

또한 모델이 업데이트될 경우 복제본을 동기화 상태로 유지하려면 복제 처리에 세심한 주의가 필요합니다. 게이트 라우터는 단일 전문가만 인식하고 복제본 자

체는 알지 못하므로 모든 복제 전문가들이 동일하게 유지되는 것(예: 동일한 가중치)이 중요합니다.

일반적으로 복제본은 원본 모델과 동일한 체크포인트에서 로드되며 독립적으로 업데이트되지 않습니다. 이는 원본 전문가와 복제본 간의 편차를 방지합니다.

적응형 전문가 라우팅 및 실시간 모니터링

기존의 훈련 후 고정되는 모어(MoE) 전문가 게이트( )와 달리, 적응형 라우팅은 추론 중 실시간으로 게이트의 결정을 조정하여 현재 상태와 전문가 부하에 대응할 수 있습니다. 예를 들어, 시스템이 한 전문가 GPU가 뒤처지고 있음을 감지하면 게이트 기능에 일부 토큰을 다른 전문가로 전환하도록 지시할 수 있습니다. 다른 전문가는 라우팅 점수가 약간 낮을 수 있지만, 사용 가능한 용량이 있으므로 요청을 수신합니다.

전문가별 활용률 및 응답 지연 시간 지표를 지속적으로 모니터링해야 합니다. 현대적인 MoE 시스템은 텔레메트리 프레임워크와 통합되어 각 전문가가 활용률 지표를 Prometheus/Grafana로 전송합니다. 이를 통해 시스템은 가동률 계수나 게이트 알고리즘을 실시간으로 동적으로 조정할 수 있습니다.

대부분의 LLM 전문가 게이트 기능은 훈련 시점에 결정된 라우팅 점수만 고려합니다. 그러나 진정한 적응형 시스템은 추론 엔진에서 처리되어 추론 시점에 동적으로 수행되어야 합니다.

적응형 라우팅 구현을 위해 추론 엔진은 모델의 포워드 패스를 사용자 정의 로직으로 감싸야 합니다. 예를 들어, 게팅 소프트맥스를 가로채 현재 부하 지표에 따라 일부 토큰을 다른 전문가에게 재할당할 수 있습니다.

추론 엔진은 GPU별 활용률 및 전문가별 토큰 수와 같은 실시간 지표를 활용하여 전문가 부하를 지속적으로 측정합니다. 한 전문가의 GPU 활용률이 99%인 반면 다른 전문가들의 GPU 활용률이 60%인 경우, 시스템은 일부 토큰을 해당 전문가의 복제본이나 전문가 선호도 점수가 약간 낮은 다른 전문가로 라우팅하여 일시적으로 부하를 낮출 수 있습니다.

그림 15-13은 편향 게이트 점수 방식을 활용한 적응형 MoE 라우팅 전략을 보여줍니다. 이 접근법은 원래 훈련 환경에서 사용되었으나, 추론에는 더 단순한 방식이 적용됩니다. 이 경우 수정된 전문가 편향 알고리즘을 사용하여 주 전문가들이 과부하 상태일 때 토큰을 대체 전문가로 전환합니다.

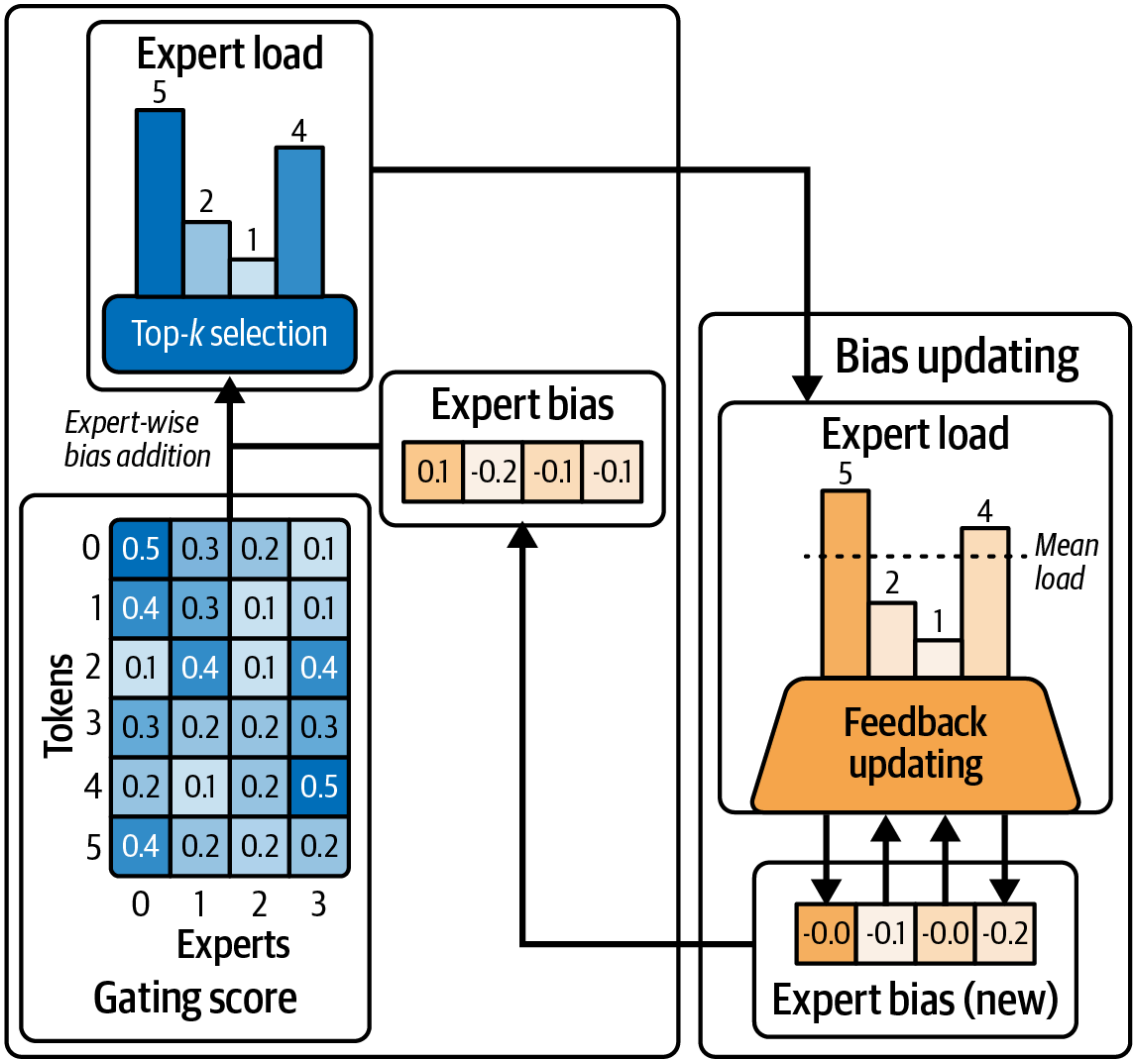

그림 15-13. 작동 중인적응형 MoE 라우팅

이 접근법은 불필요한 통신을 줄이고 부하를 더 균일하게 분산시킬 수 있습니다. 그러나 추가 모니터링, 의사 결정, 복잡성, 구성 관리, 로깅 등의 형태로 일부 추가 비용이 발생합니다. 이점이 비용을 상쇄할 수도 있고 그렇지 않을 수도 있습니다. 모든 시나리오와 워크로드는 고유하지만, 분명히 탐구할 가치가 있는 사항입니다.

Nsight Systems 같은 도구로 자질할 때는 전문가 GPU 간 전수 통신(all-to-all communication)의 타임라인 추적을 모니터링해야 합니다. 예를 들어 타임라인에서 특정 GPU의 세그먼트가 훨씬 길다면, 해당 GPU가 더 많은 토큰을 처리하고 있을 가능성이 높습니다.

추론 시스템은 이러한 통찰력을 활용하여 전문가 게이트 확률을 조정하고, 전문가들을 다른 GPU로 동적으로 재배정할 수 있습니다. 또한 추가 전문가 복제본 인스턴스를 생성하는 등의 조치를 취할 수 있습니다. 이는 전문가 게이트 알고리즘 조정, 전문가 배치 변경, 전문가 복제본 생성/제거 등을 통해 부하 재분배에 도움이 됩니다.

추론 시점에 새로운 전문가 복제본을 동적으로 생성하는 것은 쉽지 않습니다. 이 접근법은 해당 전문가들을 위해 사전 할당된 용량이나 빠른 모델 로딩이 필요합니다. 이는 고급 최적화 기법입니다."

GPU 간 특정 전문가 그룹을 다르게 구성하면 병렬화 효율이 향상되어 토큰 라우팅이 더 균일해지고 전체 처리량이 증가할 수 있습니다. 이는 GPU들이 각 레이어 작업을 더 동기화하여 완료할 수 있기 때문입니다. 지속적인 불균형이 감지되면 최신 MoE 스케줄러는 용량 계수를 조정하여 추가 전문가 복제본을 실시간으로 배포할 수 있습니다. 이는 불균형한 게팅으로 인한 핫스팟을 완화할 수 있습니다.

시스템이 수행하는 모든 동적 변경 사항을 기록하십시오. 또한 특정 전문가 활용도가 80%와 같은 임계값을 초과할 때 경보를 설정하는 것이 좋습니다.

동적 라우팅 전략은 두 가지 핵심 목표를 추구합니다: 라우팅 오버헤드 감소와 전문가 GPU 간 작업 균등 분배입니다. 낮은 오버헤드 달성은 고대역폭 상호 연결 활용, 데이터 전송과 연산 중첩, 공동 배치 및 지능형 스케줄링을 통한 중복 데이터 이동 최소화에 달려 있습니다.

로드 밸런싱은 간단한 상위 1개 또는 상위 2개 게이트 방식이나 고급 용량 인식 게이트를 통해 달성됩니다. 동적 복제 및 전문가 재할당을 통해서도 가능합니다. 이러한 기법들을 조합하여 GPU를 최대 계산량과 최소 통신 지연으로 유지하는 것이 일반적입니다.

라우팅 오버헤드가 최소화되고 부하가 균형 잡힌다면, 적응형 MoE 추론 시스템은 전문가 수를 늘릴 때 거의 선형적인 처리량 확장을 달성할 수 있습니다. 예를 들어, GPU를 두 배로 늘리면 추론 처리량을 거의 두 배로 증가시킬 수 있습니다.

요약하면, 이러한 적응형 내보내기 라우팅 및 부하 분산 최적화는 앞서 다룬 병렬 처리 및 디코딩 기법과 통합될 수 있습니다. 이를 통해 초대규모 환경에서 고성능을 발휘하도록 MoE 추론 시스템을 튜닝할 수 있습니다. 지속적인 프로파일링과 적응형 알고리즘은 GPU가 통신 지연으로 유휴 상태에 빠지지 않고 계산 작업에 집중하도록 유지합니다.

고급 추론 엔진은 특정 전문가 계산을 동적으로 우회하거나, 활용도가 낮은 전문가를 비활성화하거나, 캐싱을 통해 특정 토큰에 대한 전문가 계산을 건너뛸 수 있습니다. 이는 지연 시간을 더욱 줄여줍니다. 이러한 기법들은 본 장에서 설명한 개념을 기반으로 합니다.

핵심 요약

수십억 명의 최종 사용자에게 대규모 LLMs을 서비스하려면 추론 파이프라인의 모든 단계에서 최적화가 필요합니다. 본 장의 주요 내용은 다음과 같습니다:

지연 시간과 처리량 모두 최적화를 위한 분산 처리

prompt 사전 채우기 단계와 디코딩 단계를 별도의 GPU 풀로 분할하면 간섭을 제거할 수 있습니다. 이를 통해 첫 번째 토큰까지의 시간(TTFT) 단축과 초당 토큰 처리량(TPS) 향상을 동시에 달성할 수 있으며, 둘 중 하나를 희생할 필요가 없습니다. 이는 대규모 LLM 서비스의 핵심 기술입니다.

대규모 모델에는 하이브리드 병렬 처리 활용

수조 매개변수 규모의 모델에는 단일 병렬화 전략만으로는 부족합니다. 텐서 병렬화, 파이프라인 병렬화, 전문가 병렬화, 데이터 병렬화를 필요에 따라 조합하세요. 예를 들어, 모델을 메모리에 수용하기 위해 GPU 간 레이어 분할(TP/PP)을 수행하고, 모델 용량 확장을 위해 MoE 레이어에 전문가 병렬화를 적용하며, 처리량 요구를 충족시키기 위해 데이터 병렬 복제본을 추가합니다. 최적의 조합은 하드웨어에 따라 달라집니다. 항상 워크로드에 맞게 구성을 자질하고 조정하세요.

순차적 디코딩 병목 현상 완화

고급 디코딩 기법은 생성 속도를 크게 가속화할 수 있습니다. 빠른 초안 모델을 활용한 이중 모델 추측 디코딩은 작업에 맞게 수용률을 조정할 경우 약 2~3배의 속도 향상을 제공합니다. EAGLE-2는 대상 분포를 유지하면서 일부 작업에서 최대 3.5배(EAGLE-1 대비 20~40% 향상)의 속도 향상을 보고합니다. 메두사 구현체는 대상 워크로드에 맞게 훈련 및 검증되었을 때 비추측적 디코딩 대비 최대 3.6배 가속화를 보고합니다. 이러한 기법은 표준 검증 하에서 대규모 모델의 출력 분포를 유지하면서 토큰 수준 병렬성과 산술 집약도를 높입니다. 결과적으로 메인 모델 출력을 재훈련하지 않고도 더 빠른 응답이 가능합니다. 이는 추측적 디코딩이 품질을 보존하는 기법임을 보여줍니다.

제약 조건 하에서도 출력 품질 및 형식 유지

실제 운영 환경에서는 LLM이 엄격한 형식을 따르거나 특정 토큰을 피해야 하는 경우가 많습니다. 제약 디코딩 기법은 생성 과정에서 JSON 스키마나 금지 단어 같은 규칙을 강제 적용할 수 있게 합니다. 토큰당 일부 오버헤드가 발생하지만, 컴파일된 문법과 최적화된 마스크 경로를 사용하면 대규모 환경에서도 제약 디코딩이 일반 디코딩 대비 10% 미만의 낮은 오

버헤드로 실행될 수 있습니다. 다만 복잡한 문법이나 소규모 배치에서는 더 높은 오버헤드가 발생할 수 있습니다. 항상 제약 조건이 성능에 미치는 영향을 테스트하십시오. 가능하다면 지나치게 엄격한 규칙은 피하십시오. 과도한 백트래킹을 유발하여 생성을 느리게 할 수 있습니다.

효과적인 확장을 위한 MoE 워크로드 균형 조정

전문가 혼합 모델은 모델 크기와 GPU/전문가 수 간의 거의 선형적인 확장성을 제공합니다. 단, 라우팅을 효율적으로 처리할 경우에만 해당됩니다. 네트워크 병목 현상을 줄이기 위해 고대역폭 상호 연결과 계층적 전수 통신을 사용하십시오. 용량 제한과 상위 2개 게이트 적용을 통해 각 전문가가 비슷한 양의 작업을 받도록 하여 뒤처지는 전문가가 발생하지 않도록 하십시오. 지속적으로 과부하가 걸리는 전문가는 복제하여 부하를 분산하십시오. 잘 조정된 MoE 추론 시스템은 GPU를 추가할수록 거의 선형적인 처리량 확장성을 달성할 수 있습니다.

하드웨어-소프트웨어 공동 설계를 활용

현대 GPU 하드웨어는 이러한 병렬 및 분산 추론 방식을 지원하도록 설계되었습니다. vLLM, SGLang, NVIDIA Dynamo와 같은 추론 엔진을 포함하여 하드웨어와 토폴로지를 최대한 활용하는 소프트웨어를 사용하십시오. 이들은 최소한의 오버헤드로 다중 GPU 및 다중 노드 추론을 조정할 수 있습니다. 인노드 통신은 NVSwitch에서 유지하고, 필요한 경우에만 InfiniBand를 사용하며, 통신과 계산을 중첩시켜 하드웨어의 강점에 전략을 맞추십시오. 이러한 조화는 최상의 지연 시간과 비용 효율성을 달성하는 핵심입니다.

복잡성과 투자 대비 수익률(ROI)을 이해하라

각 최적화는 시스템 복잡성을 증가시킵니다. 추측적 디코딩 및 적응형 MoE 라우팅 같은 기법은 성능을 크게 향상시킬 수 있지만, 추가 모델이나 복잡한 로직이 필요합니다. 항상 비용을 고려하세요. 대화형 애플리케이션의 경우, 일반적으로 2~3배의 지연 시간 개선은 그만한 가치가 있습니다. 더 단순한 사용 사례의 경우, 직관적인 접근법으로도 충분할 수 있습니다. 프리필/디코딩 간섭 제거와 같은 가장 큰 병목 현상부터 해결하세요. 이후 필요에 따라 점진적으로 복잡성을 추가하십시오. 모니터링과 자질을 통해 어떤 최적화가 최고의 반환 효과를 제공하는지 판단하십시오.

결론

분리된 프리필/디코딩 파이프라인, 멀티토큰 추측 디코딩, 동적 전문가 라우팅, 적응형 오케스트레이션을 통합함으로써 최소한의 리소스 경합과 초저지연으로 LLMs을 실시간 서비스할 수 있습니다.

vLLM, SGLang, NVIDIA Dynamo와 같은 현대적 추론 서비스 플랫폼은 이러한 최적화 기법 다수를 채택합니다. 이들은 클러스터 리소스를 효율적으로 할당하고, 노드 간 KV 캐시를 조정하며, 추측적/제약적 디코딩을 구현하고, 프리필/디코드 작업을 스케줄링하는 등 다양한 기능을 수행합니다.

핵심은 알고리즘 혁신과 고성능 하드웨어 역량을 결합한 종단 간 적응형 아키텍처입니다. 향후 몇 장에 걸쳐 동적·적응형·다중 노드 서비스 전략을 포함한 모델 추론 성능 최적화를 심층적으로 살펴보겠습니다.

애플리케이션 수준의 프리픽스 캐싱, 지연 시간 인식 요청 라우팅, 강화 학습(RL) 기반 클러스터 튜닝부터 시스템 수준의 적응형 메모리 할당, 정밀도 전환, 혼잡 인식 리소스 스케줄링에 이르기까지 모든 내용을 다룰 것입니다.