Paper Review: DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving

Summary: The paper introduces DistServe, a system that improves large language model (LLM) serving efficiency by separating two very different workloads: the prefill phase (processing the full input to produce the first token) and the decoding phase (generating tokens one by one). Existing systems such as vLLM and DeepSpeed-MII run both phases on the same GPUs and batch them together, which boosts overall throughput but causes interference: long, compute-heavy prefills slow down decoding (increasing TPOT, or time per output token), while many decoding jobs increase TTFT, or time to first token. DistServe disaggregates the two phases onto different GPUs, allowing each to use its own resource allocation and parallelism strategy. Prefill jobs are compute-bound and use small batches with intra-operator parallelism to reduce TTFT, while decoding jobs are bandwidth-bound and use large batches with inter-operator parallelism to improve throughput. To manage communication between GPUs, DistServe adds bandwidth-aware placement algorithms. In high-bandwidth clusters (with Infiniband), prefill and decoding instances can be assigned freely across nodes. In lower-bandwidth settings (25 Gbps cross-node, 600 GB/s NVLINK inside each node), DistServe colocates matching pipeline stages on the same node to reduce data transfer of intermediate KV caches. The system also includes a controller that handles scheduling, batching, and adaptive re-planning when workloads change. DistServe was implemented in about 15K lines of Python and C++/CUDA and tested on a 4-node, 32xA100 80 GB GPU cluster. The experiments used OPT-13B, OPT-66B, and OPT-175B models on three workloads: chatbot (ShareGPT), code completion (HumanEval), and summarization (LongBench), each with specific TTFT and TPOT goals. Under a 90% SLO-attainment target, DistServe achieved up to 7.4x higher request rate or 12.6x tighter latency limits than vLLM and DeepSpeed-MII, with KV-cache transfer overhead below 0.1%. Overall, DistServe provides a clean and effective architecture for latency-aware, cost-efficient LLM serving by separating prefill and decoding computation and optimizing each phase independently.

Evaluation: The paper identifies a critical bottleneck in large language model serving, where prefill and decoding phases interfere with each other, and introduces DistServe as a practical solution. Its impact is significant, as it addresses the core challenge of reducing the cost per query while meeting latency goals that matter for real deployments. DistServe achieves up to 7.4x higher request rates or 12.6x tighter latency targets, leading to clear improvements in efficiency and responsiveness. The idea of separating prefill and decoding is novel in the context of autoregressive inference and is well-supported by theoretical analysis using queuing models and placement algorithms that jointly optimize resource allocation and parallelism. The evaluation is extensive and well-grounded, using OPT-13B, 66B, and 175B models on chatbot, code completion, and summarization workloads, with realistic latency targets. The results are convincing, with the simulator closely matching real measurements and showing less than 2 percent error, and the latency breakdown confirms that communication overhead remains under 0.1 percent. Overall, the paper is clearly written, methodologically strong, and presents an important contribution to efficient and scalable LLM serving.

Main takeaways

1. Separating the prefill and decoding phases of LLM inference removes a key performance bottleneck and allows each to be optimized for its own latency target, improving both efficiency and responsiveness.
2. Bandwidth-aware placement algorithm and simulation-driven parallelism search are crucial for maximizing GPU utilization while keeping communication overhead minimal.

Strengths

1. The system's placement algorithms automatically balance resource allocation and model parallelism based on workload and latency targets, delivering significant gains in cost efficiency.
2. Demonstrates practical implementation details, including bandwidth-aware placement, simulator validation, and runtime scheduling, showing readiness for real-world use.

Weaknesses

1. DistServe's resource and parallelism plan is tuned for specific workload patterns and may become inefficient when workload characteristics change.
2. The system uses a simple FCFS scheduler and lacks advanced runtime features like preemption and fault tolerance, which can cause longer requests to delay shorter ones (convoy effect).

Discussion topic: Can adaptive scheduling or reinforcement learning-based controllers be used to let DistServe adjust resource and parallelism plans in real time?