Summary of AlpaServe and Speculative Sampling

Zhenning Yang (znyang), Jiaheng Lu (jhlu), Jeremy Flics (jflics)

Problem and Motivation

The increasing popularity of large, pre-trained foundation models presents significant challenges for serving systems. These models require substantial memory to store their parameters and significant computational resources to perform inference. While techniques like model compression and pruning exist, they often result in performance trade-offs and are not always viable as model sizes continue to grow exponentially.

Serving systems must also ensure that they can meet specific latency service level objectives (SLOs), even during periods of peak demand. Traditional approaches to serving models, which typically involve allocating a dedicated GPU to each model, can lead to inefficiencies. This setup minimizes communication overhead but struggles under bursty workloads, as a single GPU can only process a limited amount of input data at a time, leading to increased latency. At the same time, partitioning large models is a must; naive partitioning results in significant cost and serving latency.

AlpaServe introduces an innovative approach by employing model parallelism; statistically multiplexing across multiple deep-learning models. This is achieved through the use of novel model placement algorithms within the serving system. The approach has demonstrated significant improvements in handling request loads, capable of processing requests at rates up to ten times higher all while maintaining latency constraints for over 99% of requests. This strategy not only optimizes resource utilization but also ensures that serving systems can efficiently manage varying workloads without compromising on performance.

Speculative Sampling tackles model serving from a different angle. Instead of improving serving by increasing resource utilization, Chen et al devise a method for accelerating transformer decoding. By using a faster, but less powerful model to generate “draft” tokens that can be verified by a more powerful model in parallel, speculative sampling can generate multiple tokens per transformer call. Ultimately, based on their benchmarking, this technique results in up to a 2.5x speedup, with no difference in quality.

SARATHI tackles the inefficiencies of decoding and pipeline bubbles through chunked-prefills and decode-maximal batching. During the Prefill phase, it processes input prompts with high GPU efficiency, even at small batch sizes. Conversely, in the Decode phase, it outputs tokens auto-regressively, facing reduced GPU utilization with smaller batches. Previous works rely on Tensor (TP) and Pipeline Parallelism (PP) to

boost efficiency; however, TP incurs high communication costs, and PP's inefficiencies are attributed to idle times. Additionally, decoding poses a memory-bound challenge.

Related Works

Model serving systems that dynamically swap large models into accelerator memory can be slow. However, over-provisioning compute resources results in low cluster utilization. Hugging Face serves thousands of models with similar model architectures, and some prior works also explore the idea of weight sharing when serving similar DL models. AlpaServe targets the general case of serving multiple distinct models.

Model parallelism was well studied in the throughput-oriented training setting, and it can be classified into two categories: intra- and inter-operator parallelism. Hence, AlpaServe explores the tradeoff between different types of parallelism.

Other work has extensively studied reducing the sampling latency of auto-regressive models. For example, model quantization and distillation can be used to decrease latency since sampling latency is tied to model size in memory.

Solution Overview

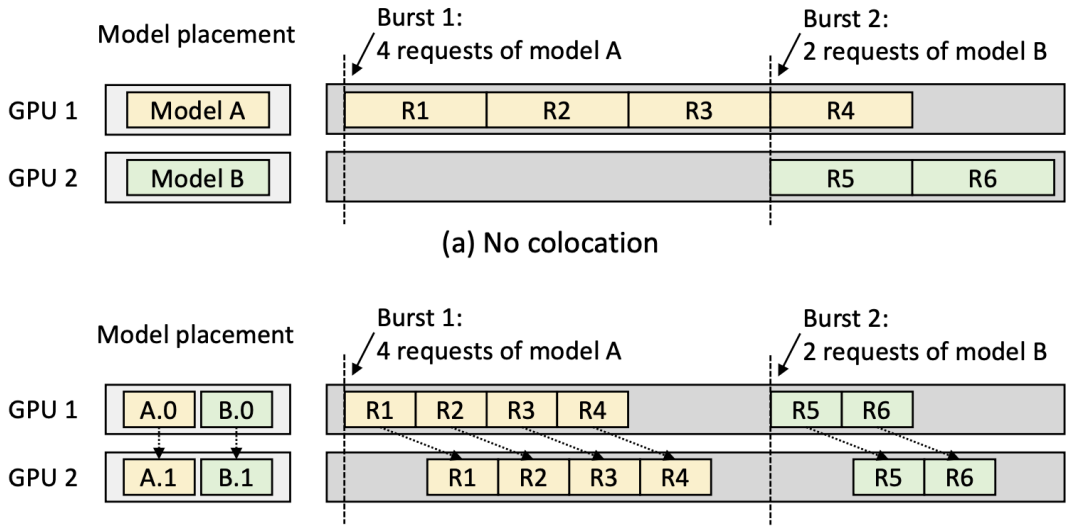

Figure 1: AlpaServe Solution

AlpaServe partition models across multiple GPUs along with pipeline execution, effectively handles the burst of requests for a single model. Fig 1a shows that, without colocation, when burst requests for model A come in, each request has to be processed sequentially, increasing the latency. On the other hand, when colocating model A across GPUs, burst requests can be processed in a pipeline fashion, achieving statistical

multiplexing, and higher resource utilization, hence, softening the negative impact of burst requests. Given a cluster configuration and a list of models, AlpaServe will iterate and profile all partition strategies, trying to find the most optimal parallel strategy that maximizes the SLO attainment.

Given lookahead K and minimum target sequence length T.
Given auto-regressive target model $q(.|.)$, and auto-regressive draft model $p(.|.)$, initial prompt sequence $x_0, \ldots, x_t$.
Initialise $n \leftarrow t$.
while $n < T$ **do**
  for $t = 1 : K$ **do**
    Sample draft auto-regressively $\tilde{x}_t \sim p(x|, x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_{t-1})$
  end for
  In parallel, compute $K + 1$ sets of logits from drafts $\tilde{x}_1, \ldots, \tilde{x}_K$:

$$q(x|, x_1, \ldots, x_n),\ q(x|, x_1, \ldots, x_n, \tilde{x}_1),\ \ldots,\ q(x|, x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_K)$$

  for $t = 1 : K$ **do**
    Sample $r \sim U[0, 1]$ from a uniform distribution.
    if $r < \min\left(1, \frac{q(x|x_1,\ldots,x_{n+t-1})}{p(x|x_1,\ldots,x_{n+t-1})}\right)$, **then**
      Set $x_{n+t} \leftarrow \tilde{x}_t$ and $n \leftarrow n + 1$.
    else
      sample $x_{n+t} \sim (q(x|x_1, \ldots, x_{n+t-1}) - p(x|x_1, \ldots, x_{n+t-1}))_+$ and exit for loop.
    end if
  end for
  If all tokens $x_{n+1}, \ldots, x_{n+K}$ are accepted, sample extra token $x_{n+K+1} \sim q(x|, x_1, \ldots, x_n, x_{n+K})$ and set $n \leftarrow n + 1$.
end while

Figure 2: Speculative Sampling with draft and target model.

In Speculative Sampling, two models are used: one with a relatively small number of parameters to generate a sequence of “draft” tokens, and one large target model that evaluates the drafted tokens. As seen in Fig 2, the draft model creates k tokens auto-regressively. Then, logits are computed in parallel based on the target model for each of these tokens. After this, a modified rejection sampling scheme is used to accept draft tokens until the draft and target models diverge. Finally, one additional token is sampled from a combination of the target and draft model to guarantee that every decoder call returns at least one new token.

This method relies on the observation that drafting many tokens in sequence from a small model and evaluating them in parallel takes roughly the same amount of time as sampling one token from a large model. Therefore even in the worst case, when all draft tokens are rejected, speculative and traditional autoregressive sampling have similar latencies.

SARATHI introduces two key innovations: Chunked-Prefills, dividing a prefill into multiple chunks, and Decode-Maximal Batching, creating a batch from a single prefill chunk with added multiple decode tokens. This approach transitions the decode phase from being memory-bound to compute-bound.

Limitations

AlpaServe

- While model parallelism is advantageous for handling bursts of requests, it is more sensitive to GPU failure. However, AlpaServe does not provide any fault tolerance mechanism.
- AlpaServe assumes all models are already in GPUs, limiting the number of models it can serve. It will be beneficial to include support for swapping models from CPU or disk.
- The models (variants of BERTs and MoEs) they used for evaluation are not diverse enough.
- When generating plans, it omits the impact of other models. (fighting for resources).
- Planning time is not shown in their evaluation.

Speculative Sampling

- Even though latency is decreased, the resource footprint of speculative sampling is larger since it requires deploying two models.
- Drafting tokens has to be done sequentially, and can't be parallelized easily.
- Because of communication complexity, the performance of draft models can be reduced by adding more physical resources. This could lead to resource underutilization.
- Lack of discussion of how to find the draft model.
- Computing draft tokens for many requests at a time is expensive, so this technique only works for small batch sizes.

Future Research Directions

AlpaServe

- Optimizing it for parameter sharing model serving.
- Fault tolerance model serving.
- Effective model weights swapping.

Speculative Sampling

- Finding a single model solution for drafting and evaluating tokens.
- Parallelizing the computation of draft tokens. Maybe by more speculation.

- Improving flexibility of resource allocation for draft models. This could even help with fault tolerance (a la Oobleck).

Summary of Class Discussion

Q: Is it possible to incorporate any fault tolerance mechanism into AlpaServe?
A: In the paper, they didn't mention any mechanism for handling failure. But we could potentially use the Oobleck idea, where we keep the pipeline replicas and use the pre-generated template to quickly recover from failure.

Q: There is no swapping in the AlpaServe paper?
A: AlpaServe assumes all models are in the GPUs and they did not address swapping.

Q: Why did they pick a smaller version of the target model to be a draft? Could they, for example, have used a quantized model instead?
A: In the paper, they did not provide a lot of justification for their draft model. However, it should make sense intuitively that the draft and target models sharing training data would lead to them having closer together distributions.

Q: Is there any analysis on throughput as a function of k?
A: The paper does not include such evaluation, but larger k values should decrease throughput since more resources are used to predict rejected tokens (on average).

Q: How do you determine the right size of the draft model?
A: There is a size/drafting speed tradeoff here. Larger draft models may have higher acceptance rates, but sequential drafting of tokens could take much longer. This is an optimization problem that can be empirically tuned.

Q: Mathematically doing speculative sampling should not degrade quality, but empirically it seems to. Why is this the case?
A: There is inherent randomness in autoregressive sampling, so even if we compared two models with the same distributions, we should expect to see some difference in quality.

Q: Could you do multi-level speculative sampling? That is, could you have an even smaller draft model that the normal draft model verifies?
A: It's possible, but might not be a good idea since each extra layer of drafting adds a factor of k computations.

Q: In the reject case of speculative sampling, why is there a difference between the two models sampled instead of just falling back on the target model?

A: Not entirely sure, but it seems to be a technical detail of the proof.

Q: Since individual requests are parallelized in speculative sampling, does this decrease the number of requests that can be served at a time?
A: Exactly. This is why speculative sampling does not work well for bursty workloads.

Q: How do you combine different percentages of split when prefilled together?
A: Some aggregation work needs to be done.

Q: How to determine the optimal chunk size after cutting?
A: Profiling is recommended. There's a trade-off: smaller prefill chunk sizes allow for more decode tokens to be "piggybacked," though the arithmetic intensity of chunked-prefills computation decreases.

Q: Can SARATHI be integrated with vLLM?
A: Yes, it can.

Q: What is left to do in inference? The papers we have read have a similar flow of improving resource utilization, reducing bubble size, and improving memory management. Are there other interesting perspectives that we can take?

A: Most of the current research is on serving requests as fast as possible, but this is not typically a necessary (or even good thing) to strive for. For example, even if it can, ChatGPT does not need to generate an entire response instantaneously, it just needs to go slightly faster than the user’s reading speed.