# ORCA: A Distributed Serving System for Transformer-Based Generative Models

Presented by Yu et al., at Seoul National University and FriendliAI

at MLSys 2022

Reviewed by Won Jong Jeon

12/8/2022

FUTUREWEI Technologies

# Overview

- Serving for large-scale transformer-based models
  - Request-based scheduling preventing early return of finished requests to clients
    - Solution: iterative scheduling
  - Issue with batching from iterative scheduling
    - Solution: selective batching of requests in the same phase
- Distributed architecture
  - Intra-layer and inter-layer parallelism
  - Engine master and worker processes
  - Minimizing CPU-GPU synchronization (compared to Megatron-LM and FastTransformer) by separate NCCL and gRPC communication for GPU and CPU respectively.
- 36.9x throughput improvement at the same of latency compared to Nvidia's FastTransformer
- Cost of serving with 400 GPT3 175B instances for same target median latency and throughput
  - Baseline 190.6M/year vs. Orca $5.7M/year

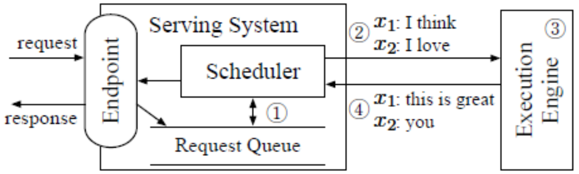



Figure 2: Overall workflow of serving a generative language model with existing systems.

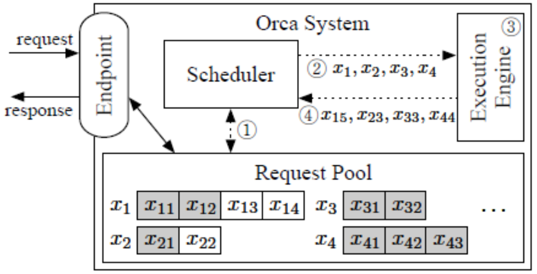



Figure 4: System overview of ORCA. Interactions between components represented as dotted lines indicate that the interaction takes place at every iteration of the execution engine.

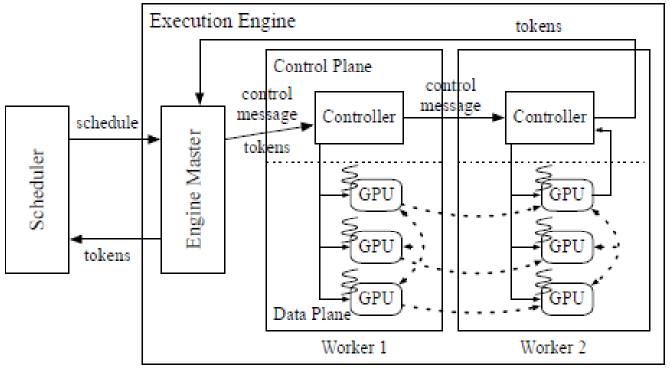



Figure 7: An illustration of the distributed architecture of ORCA's execution engine using the parallelization configura-

# Model serving system

- Main components of model serving system
  - Scheduler: responsible for:
    - Creating a batch of requests by retrieving requests from a queue
    - Scheduling the execution engine
    - Examples: Nvidia Triton Inference Server, TensorFlow Serving
  - Execution engine
    - Processing the received batch by running multiple iterations of the model
    - Returning the generated text back
    - Example: Nvidia FastTransformer
- Prior work
  - Triton: grouping multiple client request into a batch
  - FastTransformer: Conducting the inference procedure in the batched manner

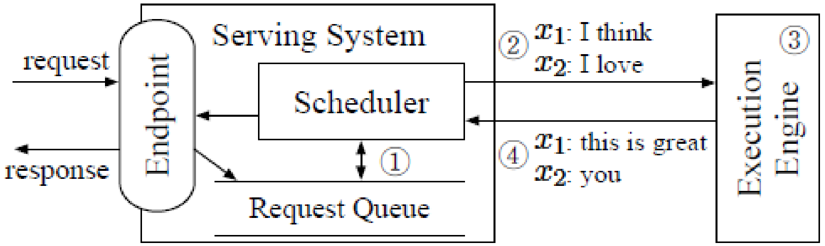
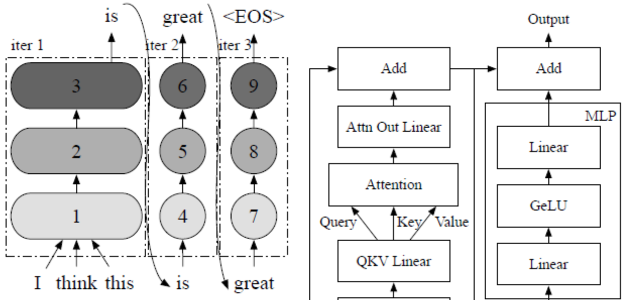



Figure 2: Overall workflow of serving a generative language model with existing systems.

# Serving of transformer-based models

- Example of inference procedure (Figure 1a): 3 iterations
  - 1st iteration: taking all the input tokens (“I think this”) and generating the next token (“is”)
  - 2nd & 3rd iteration (increment phase): taking the output token of 1st iteration and generating the next token.
- Transformer layer used in GPT model (Figure 2)
  - Attention operation computes a weighted average of the tokens, so that each token in the sequence is aware of each other.
- Example of early-finished requests (Figure 3)
  - Two requests batched in one, each has different # of iterations
  - Request $x_2$ finishes earlier than request $x_1$, limiting the efficiency of batched execution




(a) A computation graph representing an inference procedure using a GPT model. The graph does not depict layers other than Transformer layers (e.g., embedding) for simplicity.

(b) A Transformer layer used in GPT.

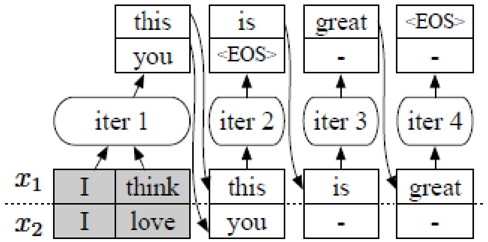



Figure 3: An illustration for a case where the requests have the same input length but some requests finish earlier than others. Shaded tokens represent input tokens. “-” denotes inputs and outputs of extra computation imposed by the scheduling.

# Challenge #1: Early-finished and late-joining requests

- Request-based scheduling (Figure 2)
  - Each request in a batch may require different # of iterations.
  - Preventing an early return of the finished request to the client, causing substantial amount of extra latency
- Solution: Iteration-level scheduling in ORCA (Figure 4)
  - Step 1: scheduler selecting requests from Request Pool to run next
  - Step 2: scheduler invoking execution engine to execute one iteration for the selected requests
  - Step 3: scheduler receiving results for the scheduled iteration
  - How to select the requests at every iteration?

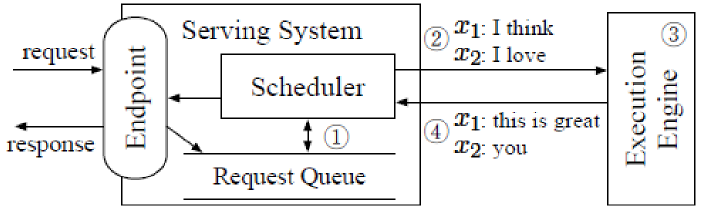



Figure 2: Overall workflow of serving a generative language model with existing systems.

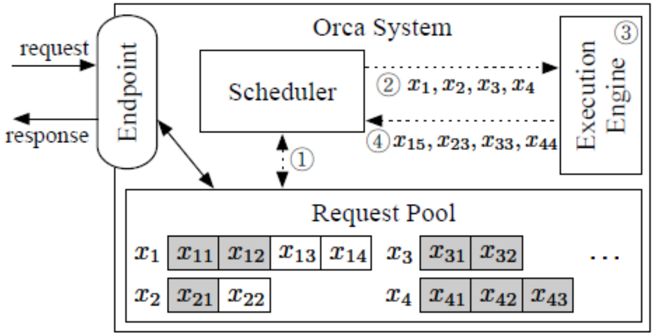



Figure 4: System overview of ORCA. Interactions between

FUTUREWEI Technologies

# Challenge #2: Batching of an arbitrary set of requests

- Naively, batching is only applicable when the two selected requests are in the same phase:
  - With the same # of input tokens (in case of initialization phase)
  - Or with the same token index (in case of increment phase)
- This restriction significantly reduces the likelihood of batching in real-world workloads.
- Solution: selective batching
  - Aware of the different characteristics of each operation
  - Splitting the batch and processing each request individually for the Attention operation, while applying token-wise (instead of request-wise) batching to other operations
  - Additional Split and Merge operation before and after Attention
  - Attention K/V manager: maintaining keys and values separately for each request until the request has finished processing

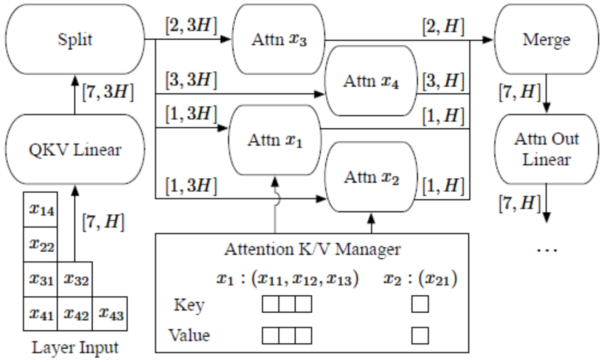



Figure 5: An illustration of ORCA execution engine running a Transformer layer on a batch of requests with selective batching. We only depict the QKV Linear, Attention, and Attention Out Linear operations for simplicity.

# Distributed architecture

- Parallelization techniques for Transformer model (Figure 6)
  - Intra-layer: splitting matrix multiplications over multiple GPUs
  - Inter-layer: splitting Transformer layers over multiple GPUs
  - Also used in FasterTransformer
- Components (Figure 7)
  - Worker process
    - Responsible for an inter-layer partition of the model
    - Each worker manages one or more CPU threads each dedicated for controlling a GPU
    - Controller: Handing over the information received from the engine master to the GPU-controlling threads
  - Engine master
    - Forwarding the received information about the scheduled batch to the first worker process.
- Optimizations
  - Minimizing GPU-GPU synchronization
  - Separate communication channels to NCCL for exchanging intermediate tensor data and gRPC for control messages between the engine master and worker controller

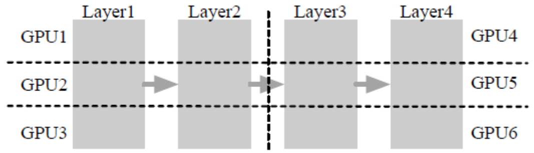



Figure 6: An example of intra- and inter- layer parallelism. A

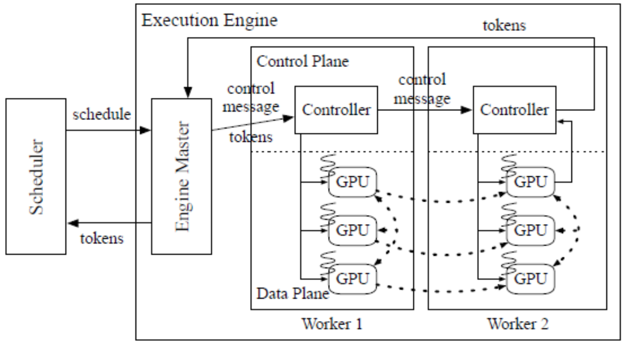
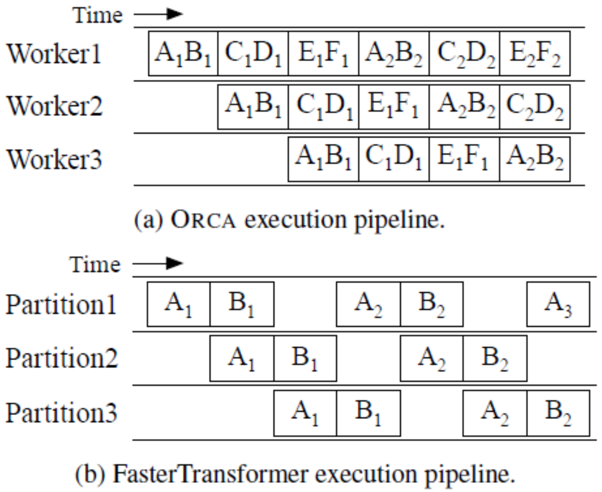



Figure 7: An illustration of the distributed architecture of ORCA's execution engine using the parallelization configura-

# Scheduling algorithm

- How to select the requests at every iteration
  - Ensuring iteration-level first-come first-serve (FCFS) policy
  - Batch size: maximizing throughput while satisfying latency budget
  - GPU memory constraint: reusing intermediate results across multiple operations
- In short, the scheduler selects at most "max batch size" requests based on the arrival time, while reserving enough space for storing keys and values to a request when the request is scheduled for the first time.
- Pipeline parallelism
  - Previous work: Splitting a batch of requests to multiple microbatches for pipeline efficiency (fewer pipeline bubbles)
  - ORCA: No need to divide a batch into microbatches, thanks to iteration-based scheduling



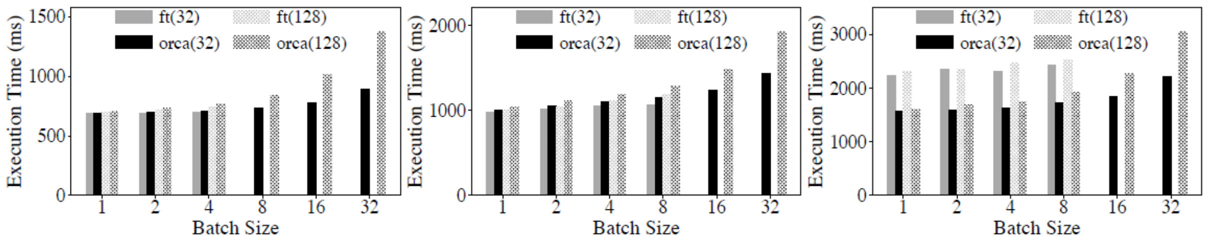
(a) ORCA execution pipeline.

(b) FasterTransformer execution pipeline.

FUTUREWEI Technologies

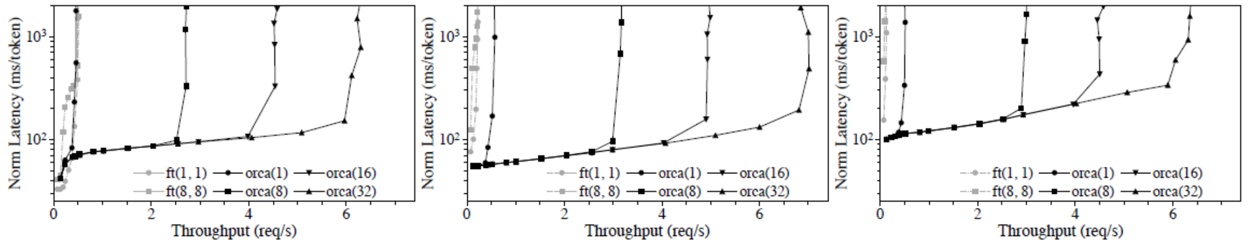
# Evaluation

- Performance of ORCA execution engine without scheduler (Figure 9)
  - ft(*n*) and orca(*n*): processing time for requests with *n* input tokens
  - Similar (or slightly worse) execution time with 13B model
  - 47% faster with 175B model, thanks to control-data plan separation
- End-to-End performance (Figure 10)
  - ft(*max_bs*, *mbs*) with a maximum batch size *max_bs* and a microbatch size of *mbs*.
  - Median end-to-end latency normalized by the # of generated tokens and throughput
  - FastTransformer 0.185 reqs/s, ORCA 6.81 req/s (36.9x speedup)




(a) 13B model, 1 GPU. (b) 101B model, 8 GPUs. (c) 175B model, 16 GPUs.

Figure 9: Execution time of a batch of requests using FasterTransformer and the ORCA engine without the scheduling component.

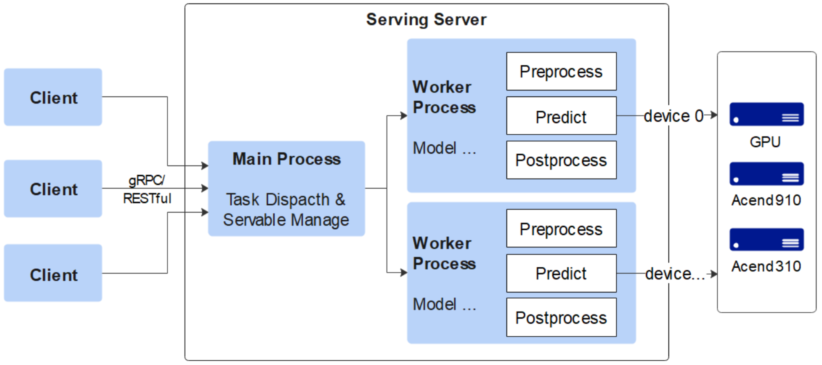


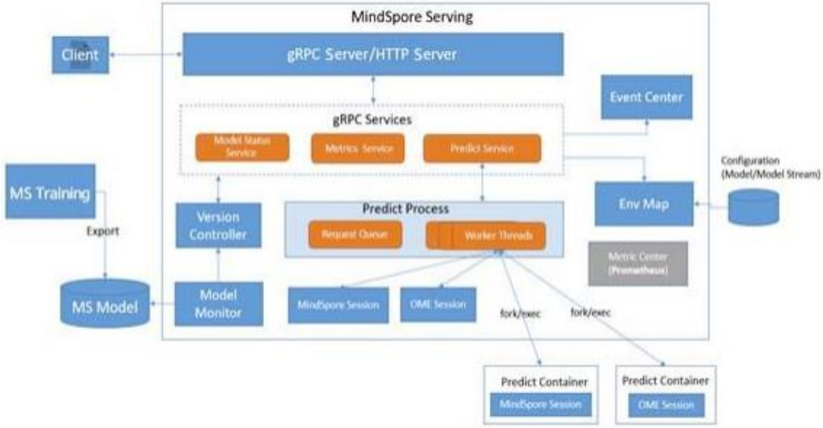

(a) 101B model, 8 GPU. (b) 175B model, 16 GPUs. (c) 341B model, 32 GPUs.

Figure 10: Median end-to-end latency normalized by the number of generated tokens and throughput. Label “orca(*max_bs*)” rep-

# MindSpore Serving perspective

- Distributed inference support
  - Multiple cards are supported in inference phase for large scale neural networks.
  - Only Ascend 910 inference is supported (with HCCL).
  - No GPU with NCCL?
  - No multiple nodes are supported?
- Generic scheduling + model execution engine
  - Request-based FCFS scheduling with batch adjustment
    - Multiple requests are split and combined to meet batch size requirement of the model.
    - No iteration-based scheduling support
- Better Transformer support ? (like in PyTorch 2.0)
  - FlashAttention (Stanford U): optimization for IO access patterns
  - xFormers (FAIR): memory efficient SDPA (Scaled Dot-Product Attention) kernels




FUTUREWEI
Technologies