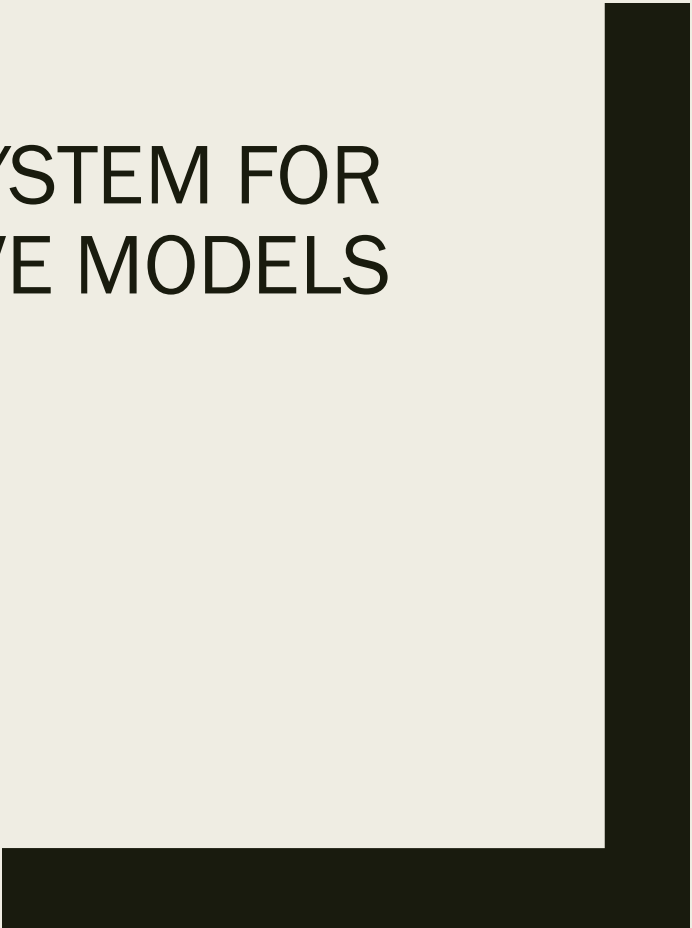




# ORCA: A DISTRIBUTED SERVING SYSTEM FOR TRANSFORMER-BASED GENERATIVE MODELS

세종대학교 전길원



# Abstact

- GPT-3 같은 Large-scale Transformer based model이 큰 관심을 받음
  - 이러한 Model의 Serving System의 필요성이 강조되고 있음
  - inference request 처리를 위해 multiple time 실행되어야 함.
- 문제 : 기존 문제는 Transformer model의 특성을 가진 workload를 잘 수행하지 못함
  - Inflexible scheduling mechanism
- 제안 1 : iteration-level scheduling(new scheduling mechanism)
  - Request 단위가 아닌 iteration 단위의 execution scheduling
- 제안 2 : Selective Batching
  - 연산 중 선택된 set에만 batching을 적용하는 방식
- 성과 : Orca System 개발, GPT-3 model 기준 FastTransformer 대비 36.9배 성능 향상

# Introduction

- Generative model이 급속도로 다양한 분야로 확산되고 중요해지고 있음
  - Generative Model의 핵심 : Transformer and its variation
  - Attention Mechanism : 기존 RNN보다 우수한 representation learning
- ML Inference serving system
  - 실제 Application에서는 inference 절차를 별도의 ML inference service에 위임
  - 기존 서비스 : Triton, TensorFlow Serving
  - 실제 예시 : Triton(Batching) + FasterTransformer(Transformer Model 추론 최적화)
- The Problem of The Existing System
  - Request 단위 Scheduling의 문제
    - 배치 내 일부 요청이 먼저 완료 되어도 전체 배치 완료까지 대기
    - 새로운 요청이 현재 배치 완료까지 큐에서 대기
    - 불필요한 대기로 인한 전체 응답 지연

# Introduction

## ■ Orca

### - Iteration-level Scheduling

- 방식 : Serving System이 Engine에게 Single Iteration만 실행하도록 지시
  - Every iteration 후, 완료된 요청을 즉시 Client에게 반환
- 효과 : New request가 Single Iteration 대기 후 바로 처리 가능

### - Selective Batching

- 문제 : Iteration-level Scheduling 시 각 request가 서로 다른 수의 token을 처리한 상태여서 Batching 어려움
- 방식 : Attention 연산은 개별 처리, 기타 연산은 Batching 적용
  - 이유 : Attention 연산은 model parameter와 무관하여 Batching의 메모리 효율성 이점 X

### - Conclusion

- Distributed serving system : 두 핵심 기술을 기반으로 System구현
- Large-scale model 지원 : hundreds parameters model까지 처리 가능한 확장성

# Background

## ■ Inference Procedure of GPT

- GPT : Autoregressive Language Model
  - 입력으로 주어진 Text Sequence를 기반으로 Next token을 순차적으로 생성
- Iteration : token 1개 생성 -> model 전체 layer 1번 실행
  - Input -> token 1 -> token2 -> ... -> <EOS>
  - Input->token1 : initiation phase , token n -> token n+1 : increment phase
- Attention 연산의 state 유지
  - Attention 연산은 모든 이전 token의 KV 정보가 필요 -> iteration 간에 state 유지 필요
  - fairseq, FasterTransformer 등에서는 incremental decoding을 사용하여 caching함
  - Transformer는 이때문에 iteration이 진행될수록 state가 커지며, LSTM과 state 관리 방식이 다름

# Background

## ■ ML Inference Serving System

- Serving System의 구조
  - 사용자가 inference request 보내면, serving system이 result return
    - Triton inference server, TensorFlow Serving
- Batching
  - Batching은 여러 request를 묶어 engine에 보내는 기술
    - GPU Resource를 효율적으로 사용하고 throughput을 높이기 위한 핵심 기법
  - 큰 입력 텐서일수록 parallel 을 잘 활용할 수 있기에 batch크기가 클수록 성능이 좋아짐
- Problem : Request-level Scheduling
  - 한 배치의 요청들 중 일찍 끝나는 요청도 늦게 끝나는 요청을 기다려야 함 -> Latency
  - 새로운 요청이 와도 현재 배치가 끝날 때까지 대기해야 함 -> Queue wait latency

# Challenge

- C1 : Early-finished and late-joining requests
  - Existing serving system은 request-level scheduling
  - 동일 배치 내 요청이 모두 끝나야 결과 반환 -> 지연 발생
- S1 : Iteration-Level Scheduling
  - 동작 절차
    - 1. Request Pool에서 실행할 Request 선택
      - Request State를 추적하며 실행 가능한 Request 선택
    - 2. 선택된 요청들에 대해 단 한 번의 iteration 실행
      - 각 Request에 대해 현재 Token, Input Sequence 전달
    - 3. Iteration 결과를 수신하고 각 request에 저장
    - 4. 종료된 Request는 Pool에서 제거, Result 반환

# Challenge

## ■ S1 : Iteration-Level Scheduling

### - 효과

#### ■ 빠른 응답

- 요청 끝나는 즉시 응답 반환 : 지연 최소화

#### ■ New request 즉시 처리 가능

- 기존 Batch가 끝날 때까지 기다릴 필요 X

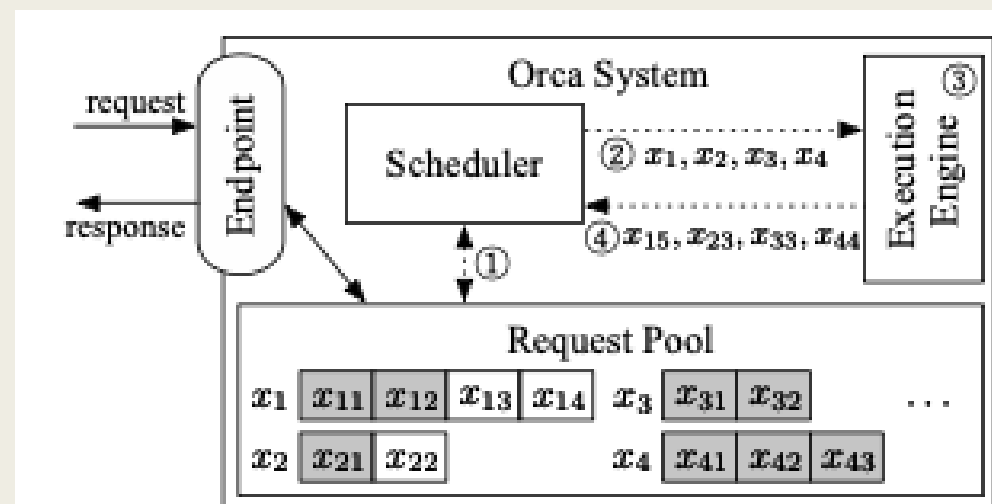


Figure 4: System overview of ORCA. Interactions between components represented as dotted lines indicate that the interaction takes place at every iteration of the execution engine.  $x_{ij}$  is the  $j$ -th token of the  $i$ -th request. Shaded tokens represent input tokens received from the clients, while unshaded tokens are generated by ORCA. For example, request  $x_1$  initially arrived with two input tokens ( $x_{11}, x_{12}$ ) and have run two iterations so far, where the first and second iterations generated  $x_{13}$  and  $x_{14}$ , respectively. On the other hand, request  $x_3$  only contains input tokens ( $x_{31}, x_{32}$ ) because it has not run any iterations yet.



# Challenge

- C2: Batching an arbitrary set of requests
  - Problem :
    - GPU를 효율적으로 사용하려면 Batch 실행 필수
    - Iteration-level scheduling에선 각 Request의 state가 제각각이라 아래의 경우 batch 불가
      - Request들이 initiation phase에 있고 input token수가 다름
      - Request들이 increment phase에 있고 현재 처리 중인 token index가 다름
      - Request들이 서로 다른 phase에 존재 (initiation vs increment)
      - 위의 경우는 input tensor의 shape이 다르기 때문에 GPU에서 병렬 실행이 어려움
- S2 : Selective Batching
  - Core Idea
    - Transformer layer의 연산들 중 일부만 선택적으로 batch 처리
    - Attention 연산은 request단위로 개별 처리, 나머지 연산은 detach하여 token단위로 처리

# Challenge

- S2 : Selective Batching
  - 실행 흐름
    - Request들로부터 input token들을 하나의 tensor로 합치기
      - $[x_1, x_2, x_3, x_4]$  input tensor를  $[7, H]$  형태의 2D Tensor로 병합
    - QKV Linear, LayerNorm, GeLU 등은 전체 token 대상 parallel처리 (입력 shape이 달라도 flatten하여 처리 가능)
    - Attention 연산 직전 Split 하여 개별 수행 (자신의 KV를 이용하여 수행)
    - Attention 결과를 Merge하여 다시 통합 수행
    - Attention KV state 관리 (KV Manager)

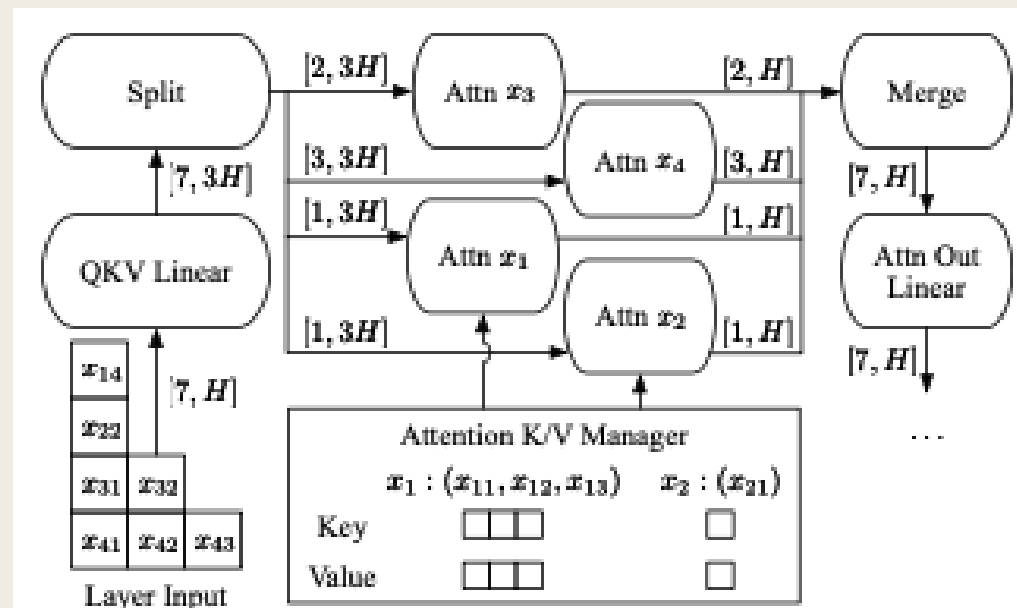


Figure 5: An illustration of ORCA execution engine running a Transformer layer on a batch of requests with selective batching. We only depict the QKV Linear, Attention, and Attention Out Linear operations for simplicity.

# Orca Design

## ■ 주요 3가지 설계 원칙

- Selective Batching : Attention 연산을 제외한 대부분의 연산을 irregular tensor도 batch 처리 가능하게 함
- Distributed parallel : Intra-layer와 inter-layer parallelism을 조합
- Scheduling 최적화 : FCFS를 유지하며 throughput과 memory constraint 동시 고려

# Orca Design

- 4.1 Distributed Architecture
  - 두 가지 parallel 전략을 결합하여 model 분산처리
    - Intra-layer parallelism
    - Inter-layer parallelism
  - Orca Execution Engine Architecture
    - Worker(GPU Controller) : 각각의 Inter-layer partition
    - Engine Master : 전체 Control 및 Request Scheduling
    - Control Message와 Tensor Data 분리 전송
      - Control : gRPC 사용 (CPU<->GPU)
      - Tensor : NCCL 사용 (GPU<->GPU)
      - 기존 시스템들은 NCCL을 Control Message에도 사용해서 Synchronization overhead가 큼.

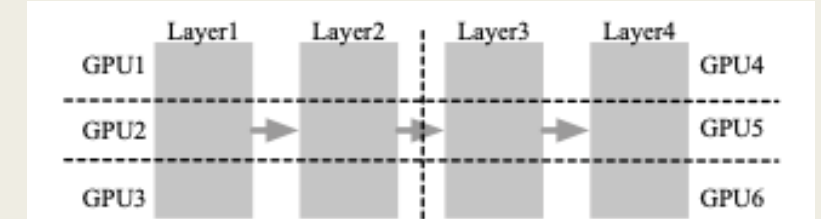


Figure 6: An example of intra- and inter- layer parallelism. A vertical dotted line indicates partitioning between layers and a horizontal line indicates partitioning within a layer.

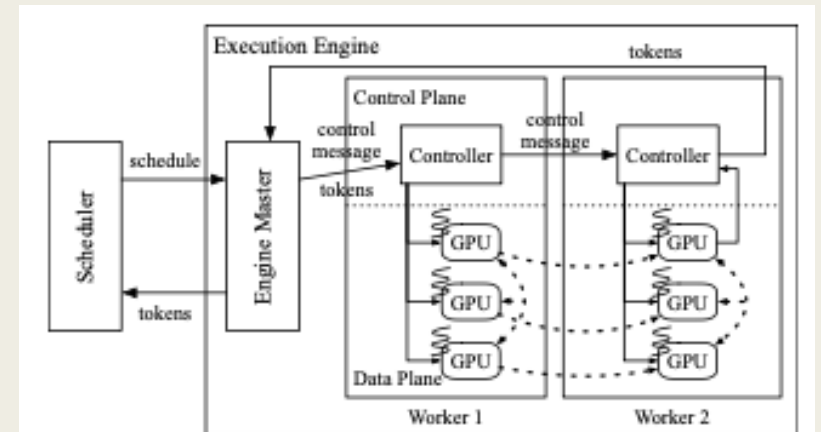


Figure 7: An illustration of the distributed architecture of ORCA's execution engine using the parallelization configuration shown in Figure 6. For example, the first inter-layer partition (Layer1 and Layer2) in Figure 6 is assigned to Worker1, while the second partition is assigned to Worker2.

# Orca Design

## ■ 4.2 Scheduling Algorithm

### - Scheduling 원칙

- Iteration-level FCFS : 먼저 도착한 request가 iteration 수 기준으로 더 먼저 처리
- Selective Batching : request가 다르더라도 연산 종류에 따라 Batch 가능 여부를 구분
- Memory-Aware Scheduling : Attention K/V 저장소 용량을 고려하여 deadlock 방지

### - Scheduler 동작

- 요청마다 max\_token을 기반으로 K/V manager의 memory slot을 예약
  - Slot 부족 시 대기
- 한 번에 max\_batch\_size 만큼의 요청을 선택하여 처리

### Algorithm 1: ORCA scheduling algorithm

**Params:**  $n\_workers$ : number of workers,  $max\_bs$ : max batch size,  $n\_slots$ : number of K/V slots

```
1  $n\_scheduled \leftarrow 0$ 
2  $n\_rsrv \leftarrow 0$ 
3 while true do
4    $batch, n\_rsrv \leftarrow Select(request\_pool, n\_rsrv)$ 
5   schedule engine to run one iteration of
     the model for the batch
6   foreach  $req$  in  $batch$  do
7      $req.state \leftarrow RUNNING$ 
8      $n\_scheduled \leftarrow n\_scheduled + 1$ 
9     if  $n\_scheduled = n\_workers$  then
10      wait for return of a scheduled batch
11      foreach  $req$  in the returned batch do
12         $req.state \leftarrow INCREMENT$ 
13        if  $finished(req)$  then
14           $n\_rsrv \leftarrow n\_rsrv - req.max\_tokens$ 
15           $n\_scheduled \leftarrow n\_scheduled - 1$ 
16
17 def  $Select(pool, n\_rsrv)$ :
18    $batch \leftarrow \{\}$ 
19    $pool \leftarrow \{req \in pool | req.state \neq RUNNING\}$ 
20    $SortByArrivalTime(pool)$ 
21   foreach  $req$  in  $pool$  do
22     if  $batch.size() = max\_bs$  then break
23     if  $req.state = INITIATION$  then
24        $new\_n\_rsrv \leftarrow n\_rsrv + req.max\_tokens$ 
25       if  $new\_n\_rsrv > n\_slots$  then break
26        $n\_rsrv \leftarrow new\_n\_rsrv$ 
27      $batch \leftarrow batch \cup \{req\}$ 
28   return  $batch, n\_rsrv$ 
```

# Evaluation

- ORCA System을  
FasterTransformer와 비교 분석함
  - Microbenchmark
  - End-to-End Performance

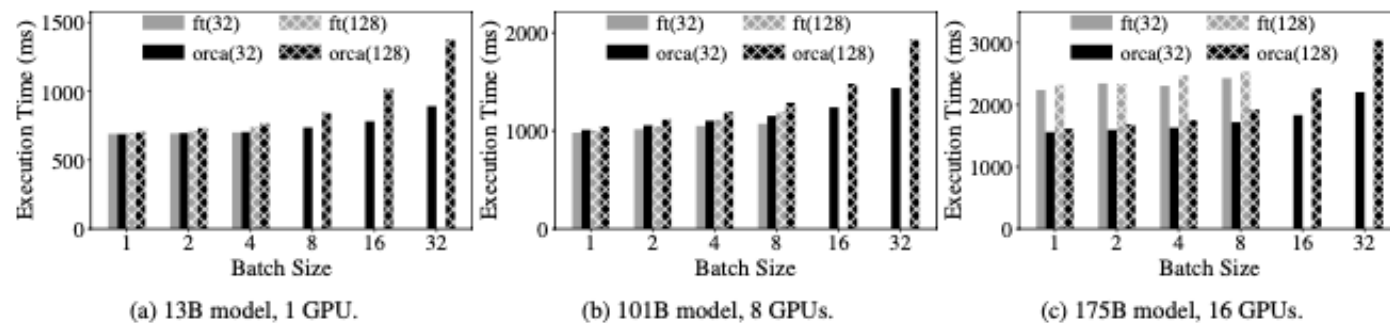


Figure 9: Execution time of a batch of requests using FasterTransformer and the ORCA engine without the scheduling component. Label “ft( $n$ )” represents results from FasterTransformer processing requests with  $n$  input tokens. Configurations that incurs out of memory error are represented as missing entries (e.g., ft(32) for the 101B model with a batch size of 16).

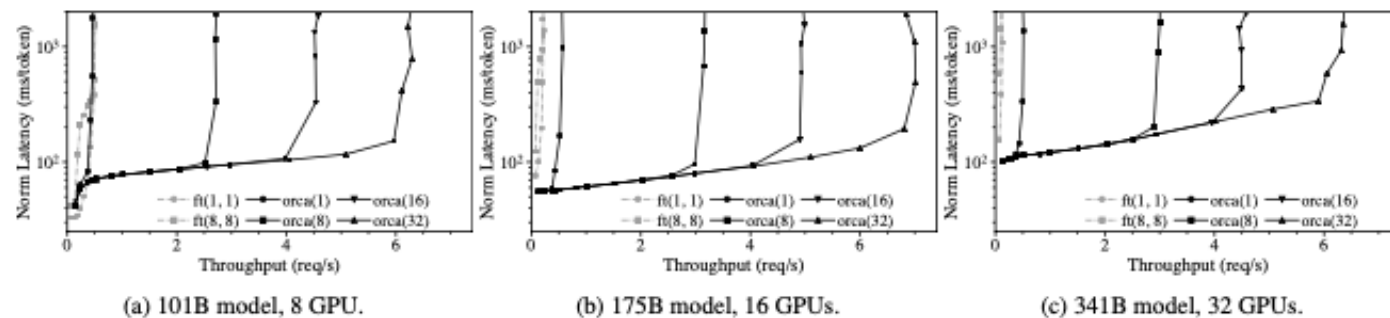


Figure 10: Median end-to-end latency normalized by the number of generated tokens and throughput. Label “orca( $max\_bs$ )” represents results from ORCA with a max batch size of  $max\_bs$ . Label “ft( $max\_bs, mbs$ )” represents results from FasterTransformer with a max batch size of  $max\_bs$  and a microbatch size of  $mbs$ .

# Evaluation

- Engine Microbenchmark (Figure 9)
  - 목적
    - ORCA Engine 자체의 성능을 Scheduler의 영향을 제외하고 평가
  - 실험 방법
    - 같은 requests로 구성된 batch를 iterative하게 ORCA엔진에 주입
    - 각 Request는 동일한 input token(32,128) output token(32)를 가짐
    - 13B, 101B, 175B 실험

# Evaluation

- End-to-End Performance

- 목적
  - 실제 workload를 기반으로 ORCA 전체 시스템의 throughput과 latency 평가
- 실험 방법
  - Input token 수 :  $U(32,512)$ 에서 Sampling
  - Generate token 수 :  $U(1,128)$ 에서 Sampling -> 최대 128 Iteration
  - FasterTransformer는 수동 scheduler를 구현하여 비교



# Evaluation

- Batch Size와 Scheduling의 영향 분석
  - Orca는 max\_batch\_size 증가가 throughput 증가에 도움
  - FasterTransformer는 batch size 증가가 오히려 저하시킴
    - Microbatch 처리 제약
    - Input length / generation token 수 imbalance 시 완료 요청을 기다려야 함
- Homogeneous Request
  - 모든 요청의 input/output 길이가 동일한 request
    - 총 처리되는 iteration 수도 동일
  - FasterTransformer도 max\_batch\_size 증가에 따라 성능 향상
    - 하지만 Orca가 여전히 좋았음
  - max\_batch\_size = 1(요청 하나씩 처리)인 경우, orca가 pipeline일 뿐이어서 batching의 이점이 없어짐