Whale: Efficient Giant Model Training over Heterogeneous GPUs

Xianyan Jia, Le Jiang, Ang Wang, Wencong Xiao, Ziji Shi, Jie Zhang, Xinyuan Li, Langshi Chen, Yong Li, Zhen Zheng, Xiaoyong Liu, Wei Lin

Alibaba Group
07/12/2022
Model-Size Increasing

AI and Memory Wall

Transformer Size: 240x / 2 yrs
AI HW Memory: 2x / 2 yrs

AI and Memory Wall: https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8
Memory & Bandwidth Wall

- High communication overhead for Data Parallel
- GPU memory limit

AI and Memory Wall: https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8
Distributed Training Strategies

Data Parallelism

Pipeline Parallelism

Tensor Model Parallelism
Data + Pipeline

- Pure pipeline parallelism does not scale well with more GPUs
- Nested DP with pipeline
Apply different strategies to different model parts.
Heterogeneity in GPU Clusters

- Gang Schedule

Heterogeneous GPUs as a resource
- (e.g., Computing Pool with GPUs: P100, V100, A100 and etc)
Challenges: Heterogeneous GPU Training

Inefficiency in utilizing heterogeneous GPUs
- different GPU types: different computing/memory/network capacity
- imbalance in computing time -> low utilization
- gap between model development and the hardware environment

(a) Naïve DP with identical batch size
Gaps and Opportunities

- Lack of unified abstraction to support all of the parallel strategies and the hybrids

- Fully automatic parallel strategy has high cost for giant models

- Inefficiency in utilizing heterogeneous GPUs

- Require significant model code refactoring
Gaps and Opportunities

- Lack unified abstraction to support all of the parallel strategies and the hybrids
  ✓ Unified abstraction for strategy expression
- Fully automatic parallel strategy has high cost for giant models
  ✓ Incorporate user hints
- Inefficiency in utilizing heterogeneous GPUs
  ✓ Parallel strategies should be used adaptively and dynamically
- Require significant model code refactoring
  ✓ Minimize code change, switch among strategies easily
Whale: Efficient Giant Model Training over Heterogeneous GPUs

- Two new high-level primitives for unified expression

- Transform distributed models efficiently and automatically

- Hardware-aware load balancing algorithm

- Train the largest multi-modality model M6 with ten trillion model with only 4-lines of code change
Outline

• Introduction

• Whale: design abstraction

• Whale: parallel planner

• Whale: hardware-aware load balance

• Evaluation

• Conclusion
Abstraction: Internal Key Concepts

TaskGraph

VirtualDevice
Parallel Primitives

Parallel primitive is a Python context manager. Operations defined under form one TaskGraph (TG)

`replicate(device_count)` annotates a TG to be replicated.
- `device_count`: #devices for TG replicas

`split(device_count)` annotates a TG to apply intra-tensor sharding.
- `device_count`: #devices for sharded partitions
Parallel Examples

```
import whale as wh
wh.init(wh.Config({
    "num_micro_batch": 8}))
with wh.replicate(1):
    model_stage1()
with wh.replicate(1):
    model_stage2()
```

Pipeline with 2 TaskGraphs

```
import whale as wh
wh.init()
with wh.replicate(total_gpu):
    features = ResNet50(inputs)
with wh.split(total_gpu):
    logits = FC(features)
    predictions = Softmax(logits)
```

Hybrid of replicate and split
Parallel Planner

(a) Parallel primitive annotation

(b) Virtual device generation

(c) Parallel plan Generation
Hardware-aware Load Balancer

Balance the computing load proportional to the device computing capacity, s.t. memory constraints.

Data parallelism: balance the FLOP by adjusting local batch while keeps the mini-batch unchanged.

Tensor Model Parallelism: balance the FLOP of partitioned operations through uneven sharding.
Memory-Constraint Load Balancing

Algorithm 1: Memory-Constraint Load Balancing

Input: TaskGraph \( TG, VirtualDevice(N) \)

1. \( load\_ratios = \emptyset; \text{mem\_utils} = \emptyset; \text{flop\_utils} = \emptyset \)
2. \( oom\_devices = \emptyset; \text{free\_devices} = \emptyset \)
3. \( \text{foreach } i \in 0...N \text{ do} \)
   4. \( load\_ratios[i] = \frac{DF_i}{\sum_{i=0}^{N} DF_i} \)
   5. \( \text{mem\_utils}[i] = \frac{load\_ratios[i] \times TG_{mem}}{DM_i} \)
   6. \( \text{flop\_utils}[i] = \frac{load\_ratios[i] \times TG_{flop}}{DF_i} \)
   7. \( \text{if } \text{mem\_utils}[i] > 1 \text{ then} \)
      8. \( \text{oom\_devices.append}(i) \)
   9. \( \text{else} \)
      10. \( \text{free\_devices.append}(i) \)
4. \( \text{while } \text{oom\_devices} \neq \emptyset \text{ and } \text{free\_devices} \neq \emptyset \text{ do} \)
   5. \( \text{peak\_device} = \text{argmax}(\text{oom\_devices}, \text{key} = \text{mem\_utils}) \)
   6. \( \text{valley\_device} = \text{argmin}(\text{free\_devices}, \text{key} = (\text{flop\_utils}, \text{mem\_utils})) \)
   7. \( \text{if } \text{shift\_load}(\text{peak\_device}, \text{valley\_device}) == \text{success} \)
      8. \( \text{then} \)
         9. \( \text{update\_profile}((\text{mem\_utils}, \text{flop\_utils})) \)
         10. \( \text{oom\_devices.pop}(\text{peak\_device}) \)
   11. \( \text{else} \)
      12. \( \text{free\_devices.pop}(\text{valley\_device}) \)
Load Balancer Example

- Earlier TaskGraph has higher peak memory than later TaskGraph (e.g. BertLarge)

- Place earlier TaskGraphs on devices with higher memory capacity.

- Partition the model operations to TaskGraphs in a topological sort, balance the TaskGraphs computing FLOP proportional to device capacity.
Outline

• Introduction
• Whale: design abstraction
• Whale: parallel planner
• Whale: hardware-aware load balance
• Evaluation
• Conclusion
(a) Whale DP vs TF DP on ResNet

(b) Whale DP vs TF DP on BertLarge

(c) Whale Pipeline vs Gpipe on BertLarge

(a, b) Whale DP obtained better performance than TF Estimator DP on ResNet and BertLarge

(c) Whale pipeline on BertLarge outperforms Gpipe

* 4-stages 1.45X 🚀, and 8 stages 1.14X 🚀
Micro-benchmark: Hybrid Strategy

(a) BertLarge
Hybrid strategy

(b) ResNet50 #class=100K
DP vs Hybrid

(c) ResNet50 #class=1M
Hybrid strategy

(a) Hybrid pipe+DP (TG=2 and TG=4) got better performance than pure pipe (TG=8) on 8 GPUs
(b) #class=100K, Hybrid split+DP got better performance than pure DP, 1.13~2.43X
(c) #class=1M, DP fails due to OOM. Hybrid achieved 95% scaling from 8~32 GPUs
Micro-benchmark: Hardware-aware

Setup: 8 32GB V100 GPUs and 8 16GB P100 GPUs
(a) Hardware-aware DP got 1.3X to 1.4X
(b) Hardware-aware Pipeline got 1.2X
Industry-Scale Giant Model Training

- M6-10B: 91% throughput scalability from 8 to 256 GPUs
- M6-MoE-10T: A few lines to switch from pipeline to tensor model parallelism (MoE). Train on 512 NVIDIA V100 GPUs.

(a) M6-10B with Pipeline and DP
(b) M6-10B throughput
(c) M6-MoE-10T
Conclusion

Whale: Efficient Giant Model Training over Heterogeneous GPUs

- Efficiency, programmability, and adaptability
- Supports various parallel strategies using a unified abstraction
- Adapts to heterogeneous GPUs with automatic graph optimizations
- Deployed DL infrastructure at Alibaba for real giant model training

[Code] https://github.com/alibaba/EasyParallelLibrary
Thanks

Q&A