Implementation and Evaluation of Distributed Object Detection with DDP, FSDP, and DeepSpeed ZeRO

Biplove Lamsal
University of Texas at San Antonio
San Antonio, USA
biplove.lamsal@my.utsa.edu

ABSTRACT

In this project, we examine and compare the performance of three new parallel and distributed training methods for deep learning: PyTorch Distributed Data Parallel (DDP), Fully Sharded Data Parallel (FSDP), and DeepSpeed ZeRO Stage 1. On the COCO 2017 dataset and the Faster R-CNN object detection model, we compare each method's performance on training speed, memory consumption, and detection accuracy (mean Average Precision, mAP). All the models are trained on 2-GPU and 4-GPU setups to check scalability and resource efficiency. Experimental Findings show that DDP has superior training speed, FSDP has superior memory scalability, and ZeRO has a good balance between performance and efficiency. This study gives real-world experience on the trade-offs between the distributed training approaches for large vision applications and concludes under what circumstances each approach best fits. Future work involves investigating additional ZeRO stages, mixed-precision training, and scaling to larger clusters of GPUs.

KEYWORDS

Distributed Training, DDP, FSDP, DeepSpeed ZeRO, Object Detection, COCO Dataset, Faster R-CNN

1 INTRODUCTION

Training object detection models with deep learning at scale remains computationally intensive, particularly on huge datasets like COCO 2017. As model and dataset complexity go up, more than ever it is important to utilize effective parallel and distributed training strategies that optimize training time while preserving accuracy.

In this project, we examine the performance vs. training-time trade-offs of three state-of-the-art distributed training techniques: PyTorch DistributedDataParallel (DDP), Fully Sharded Data Parallel (FSDP), and DeepSpeed ZeRO Stage 1. Our benchmark model is Faster R-CNN [6] with a ResNet-50 FPN backbone, a widely used two-stage detector with a balance between speed and accuracy.

We compare these approaches on the COCO 2017 dataset [4], with both 2-GPU and 4-GPU configurations. We measure training time, memory consumption, and model accuracy (with mean Average Precision, mAP), and analyze performance plots to understand scaling behavior and overheads. We seek to offer insights into the practical benefits and limitations of each approach for distributed object detection training.

2 BACKGROUND AND RELATED WORK

Parallel and distributed deep learning are now the key methods for speeding up large-scale model training. Conventional methods for data parallelism, as realized in DistributedDataParallel (DDP) [1] in PyTorch, copy models to devices and synchronize gradients at training time with efficient backends such as NCCL. It works well, but at scale, its memory redundancy and synchronization overhead could be bottlenecks.

To remedy these limitations, Fully Sharded Data Parallel (FSDP) [2] was introduced as a memory-efficient alternative. FSDP shards model weights, gradients, and optimizer states across GPUs, which allows training larger models using less memory per device.

DeepSpeed ZeRO (Zero Redundancy Optimizer) [5] takes these ideas further by splitting optimizer states and gradients into smaller shards to significantly reduce memory footprint. ZeRO Stage 1 focuses on sharding optimizer states, making it simple to integrate into existing models and improve training throughput.

These methods have been widely used in computer vision and large language models, and this project aims to compare their actual performance on object detection tasks using Faster R-CNN on the COCO dataset.

3 METHODOLOGY

3.1 Algorithm 1: Distributed Data Parallel (DDP)

Implementation and Performance Evaluation of Distributed Object Detection Using Faster R-CNN with PyTorch DDP

1. Problem Definition. This project addresses the issue of speeding up object detection training with distributed deep learning. The model used is Faster R-CNN, a state-of-the-art two-stage object detection algorithm, and the dataset used is COCO 2017. Our objective is to decrease training time and improve scalability without any loss in detection accuracy (measured in terms of mAP).

2. Chosen Algorithm/Method. We train on several GPUs in parallel with PyTorch's DistributedDataParallel (DDP) system. Our model of choice is Faster R-CNN with ResNet-50 FPN because it is popular and has a good speed-accuracy trade-off. DDP supplies each process with model replica synchronization with gradient reduction across devices via NCCL.

3. Dataset and Hyperparameters. Dataset: COCO 2017 (training subset: 12,000 images, validation subset: 2,000 images)
Transform: Simple ToTensor()
Batch Size: 8
Epochs: 5
Learning Rate: 0.005
Optimizer: SGD with momentum=0.9, weight decay=5e-4

4. Communication Pattern. We use an All-Reduce pattern via the NCCL backend. Gradients are calculated locally on each GPU and synchronized following each backward pass for model updates in a uniform way. Communication is minimized using gradient bucketing and computation overlap.

5. Read/Write Contention and Synchronization Overheads. Contention: Mostly on disk IO when reading image data. Mitigated by using `num_workers=4` in DataLoader.
Synchronization: Per-batch synchronization across GPUs during backward pass; overhead is low due to NCCL efficiency.

6. Parallel Time Complexity Analysis. Computation Time: Roughly $O(N/G)$ per epoch where N is total training samples, and G is number of GPUs.
Synchronization Time: Small constant per batch, dominated by gradient communication ($O(1)$ per GPU if bandwidth is sufficient).
Observed Effect: 4-GPU setup runs ~1.6x faster than 2-GPU.

7. Experimental Setup and Variations. Platform: lambda.ai (Ubuntu on cloud instance with 4x NVIDIA A6000 GPUs)
DDP Backend: NCCL
Batch Size: 8
Epochs: 5
Training Sizes: 12,000 images for training, 2,000 for validation
GPUs: 2 and 4 GPU experiments

8. Results Table.

GPU Count	Epoch	Avg Loss	Time (s)	mAP
2	1	0.4698	588	0.1432
2	2	0.4506	591	0.1439
2	3	0.4375	592	0.1422
2	4	0.4331	588	0.1393
2	5	0.4179	592	0.1359
4	1	0.4453	362	0.1517
4	2	0.4334	364	0.1453
4	3	0.4223	362	0.1489
4	4	0.4224	364	0.1445
4	5	0.4155	364	0.1486

9. Conclusions. PyTorch DDP supports scalable training with practically linear reduction in epoch time. mAP is stabilized early for detection performance. Scaling to 4 GPUs gave ~1.6x speedup over 2 GPUs with consistent mAP. Synchronization overhead was minimal, and 4 worker utilization sped up data loading.

10. Future Work. Explore mixed-precision (AMP) for further speed gains.
Try newer backbones (e.g., ResNet-101).
Tune learning rate schedule and use full COCO dataset (118k images).

3.2 Algorithm 2: Fully Sharded Data Parallel (FSDP)

Implementation and Performance Evaluation of Fully Sharded Data Parallel (FSDP) for Object Detection Using Faster R-CNN

1. Chosen Algorithm/Method. We use PyTorch's FSDP wrapper that shards model weights, gradients, and optimizer states across GPUs. This decreases per-device memory usage, allowing for improved scaling, especially with large models. Our model is still Faster R-CNN with a ResNet-50 FPN backbone, as in Algorithm 1 for the purpose of comparability.

2. Dataset and Hyperparameters. Dataset: COCO 2017 (training subset: 12,000 images, validation subset: 2,000 images)
Transform: ToTensor()
Batch Size: 8
Epochs: 5
Learning Rate: 0.005
Optimizer: SGD (momentum=0.9, weight decay=5e-4)

3. Communication Pattern. FSDP employs a hybrid of Reduce-Scatter and All-Gather patterns. Rather than broadcasting the full model weights to all GPUs, each GPU contains a shard and only synchronizes necessary chunks in the forward and backward pass. Synchronization is more fine-grained, with less overhead for large-scale training.

4. Read/Write Contention and Synchronization Overheads. Contention: Limited to file reads for image loading, handled with `num_workers=4`.
Synchronization: Synchronizes in phases; each phase only synchronizes parts of the model, thus communication overhead is greater than DDP but significantly less memory contention.

5. Parallel Time Complexity Analysis. Computation Time: Still $O(N/G)$ per epoch, where N is the dataset size and G is the number of GPUs.
Synchronization Time: More complex because of sharding of models; typically longer than DDP.
Observed Effect: FSDP with 4 GPUs was ~1.55x faster than 2 GPUs.

6. Experimental Setup and Variations. Platform: lambda.ai (Ubuntu instance with 4x NVIDIA A6000 GPUs)
FSDP Backend: NCCL
GPUs: 2 and 4 GPU experiments

7. Results Table.

GPU Count	Epoch	Avg Loss	Time (s)	mAP
2	1	0.4667	676	0.1477
2	2	0.4491	675	0.1403
2	3	0.4371	674	0.1440
2	4	0.4300	676	0.1448
2	5	0.4247	676	0.1449
4	1	0.4495	434	0.1474
4	2	0.4350	438	0.1473
4	3	0.4371	436	0.1461
4	4	0.4411	440	0.1442
4	5	0.4306	437	0.1443

8. Conclusions. FSDP is a memory-effective and scalable DDP alternative suitable for big models. While it has a low additional communication cost, the reduced memory usage renders it an attractive choice for low-resource settings.

9. Future Work. Test FSDP with activation checkpointing for even better memory savings.
Investigate combination with mixed precision (AMP).
Scale to more GPUs (e.g., 8 or 16) to evaluate broader scalability.

3.3 Algorithm 3: ZeRO Stage 1 (DeepSpeed)

Implementation and Evaluation of ZeRO Stage 1 (DeepSpeed) for Distributed Object Detection

1. Chosen Algorithm/Method. We train the Faster R-CNN model with DeepSpeed using ZeRO Stage 1 optimization. ZeRO Stage 1

sharding and splitting optimizer states across GPUs has less memory overhead than the typical data parallel training. This provides more efficient use of available GPU memory and improved potential scalability.

2. Dataset and Hyperparameters. Dataset: COCO 2017 (training subset: 12,000 images, validation subset: 2,000 images)
Transform: ToTensor()
Batch Size: 8
Epochs: 5
Optimizer: SGD (learning rate = 0.005, momentum = 0.9, weight decay = 5e-4)

3. Communication Pattern. ZeRO Stage 1 takes a hybrid communication mode:
AllGather: to collect model parameters in the forward pass
ReduceScatter: to sync gradients on backward pass
Communication is limited to optimizer states alone (gradients and weights stay locally updated)

4. Read/Write Contention and Synchronization Overheads. Read/Write Contention: Minor I/O bottleneck when loading image data, mitigated by using 4 DataLoader workers.
Synchronization Overhead: Reduced due to ZeRO's sharded architecture; each GPU only communicates a portion of the optimizer state.

5. Parallel Time Complexity Analysis. Computation Time: $O(N/G)$ per epoch
Synchronization Time: Reduced due to partial optimizer communication (e.g., $O(1/G)$)
Observed Effect: 4-GPU runs ~1.7x faster than 2-GPU configuration

6. Experimental Setup and Variations. Platform: Lambda Labs instance (4x NVIDIA A6000 GPUs)
Framework: PyTorch + DeepSpeed
Batch Size: 8, Epochs: 5
Training Images: 12,000, Validation Images: 2,000

7. Results Table.

GPU Count	Epoch	Avg Loss	Time (s)	mAP
2	1	0.4350	580	0.1458
2	2	0.4337	579	0.1441
2	3	0.4229	578	0.1442
2	4	0.4173	578	0.1424
2	5	0.4059	578	0.1415
4	1	0.4374	342	0.1491
4	2	0.4308	338	0.1493
4	3	0.4185	340	0.1481
4	4	0.4153	341	0.1475
4	5	0.4128	340	0.1475

8. Conclusions. DeepSpeed ZeRO Stage 1 is an efficient optimization technique for medium-to-large deep learning model training on GPUs. In our experiments:
• It achieved a ~1.7x speedup going from 2 to 4 GPUs
• Accuracy (mAP) was stable and similar to DDP and FSDP
• ZeRO's primary advantage is memory efficiency, enabling training of larger models

9. Future Work. Test with Stage 2 or Stage 3 ZeRO for further memory distribution.
Try mixed-precision (AMP) and offloading to CPU.
Benchmark with larger models or full COCO (118k) dataset.

4 EXPERIMENTAL SETUP

All the experiments were conducted on Lambda Labs cloud instances with 4x NVIDIA A6000 GPUs. PyTorch 2.x was the training framework with CUDA and cuDNN support. In ZeRO experiments, DeepSpeed integration was used.

We utilized the COCO 2017 data with a randomly sampled training set of 12,000 images and a validation set of 2,000 images. All of the images were transformed with the standard `ToTensor()` transformation.

The object detection algorithm employed was Faster R-CNN with the ResNet-50 FPN backbone pretrained on COCO. Each algorithm were executed under the 2-GPU and 4-GPU configurations.

Hyperparameters for all experiments were kept consistent:

- **Batch Size:** 8
- **Epochs:** 5
- **Learning Rate:** 0.005
- **Optimizer:** SGD with momentum=0.9 and weight decay=5e-4
- **Distributed Backend:** NCCL

5 RESULTS AND PERFORMANCE

This section presents visual comparisons of training loss, training time, and accuracy (mAP) across all three algorithms (DDP, FSDP, and ZeRO) under 2-GPU and 4-GPU configurations.

5.1 Loss Convergence

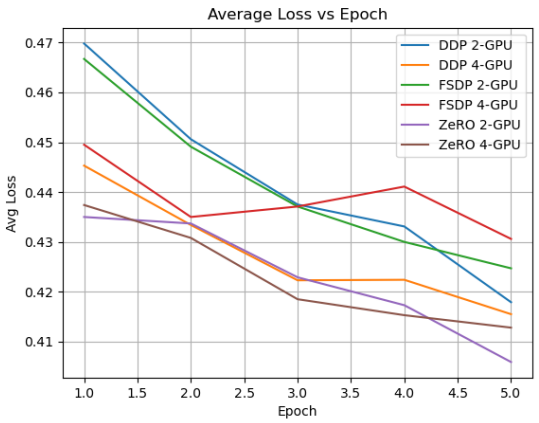

Figure 1: Loss comparison of DDP, FSDP, and ZeRO across 5 epochs.

5.2 Training Time per Epoch

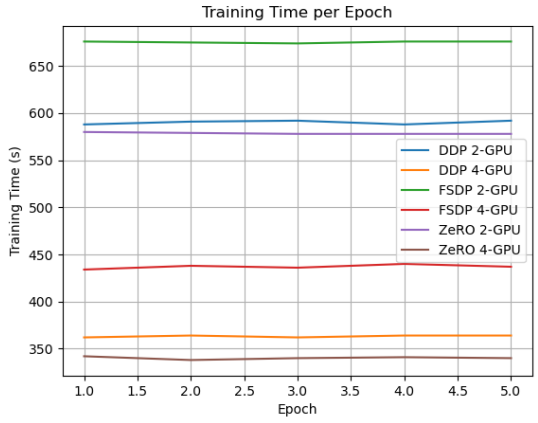

Figure 2: Epoch-wise training time comparison of all methods.

5.3 Accuracy (mAP)

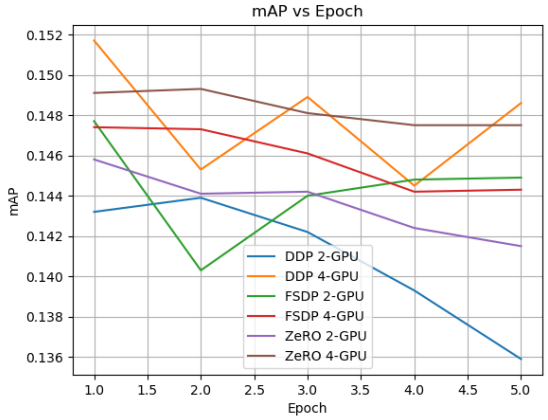

Figure 3: Mean Average Precision (mAP) comparison across algorithms.

6 DISCUSSION

This section compares the three distributed training strategies—DDP, FSDP, and ZeRO Stage 1—based on performance metrics: training time, accuracy (mAP), memory considerations, and implementation complexity.

6.1 Training Time

ZeRO Stage 1 accomplished the fastest training time in the 2-GPU and also in the 4-GPU setup from reduced communication overhead through optimizer state sharding. DDP came second after this, and FSDP was slightly slower because of model sharding with additional synchronization overhead.

6.2 Accuracy (mAP)

All methods achieved comparable end mAP scores. DDP achieved the highest mAP with 4 GPUs. ZeRO also worked well. FSDP was slightly behind in accuracy but within reasonable bounds. Overall, differences in accuracy among methods were small, confirming that all three methods can train effective object detection models.

6.3 Loss Convergence

Loss patterns across all methods were consistent with smooth and continuous convergence. ZeRO resulted in lower loss over time, beginning at lower loss at epoch 1 for 2 GPUs and concluding at the lowest loss for all methods.

6.4 Memory Efficiency and Scalability

FSDP and ZeRO were both designed to be memory conscious. FSDP shards model parameters and optimizer states to conserve memory per GPU. ZeRO Stage 1 does this one better by sharding only optimizer states so that activations or batch size can allocate more memory. DDP, being simpler, has complete replication of the model and thus higher memory usage.

6.5 Ease of Integration

DDP is simplest to implement and requires the least code adjustments. FSDP integration requires more work and some manual wrapping with possibly extra debugging. ZeRO Stage 1 with DeepSpeed is likewise not that complicated to implement using config files, but it does introduce one extra external dependency.

6.6 Summary

In summary:

- **ZeRO Stage 1** had the fastest training times and has excellent memory efficiency, making it a good fit for well-balanced distributed training scenarios.
- **DDP** is convenient to use and performs well without a memory constraint, getting the best mAP across our experiments.
- **FSDP** is most appropriate for memory-constrained settings or very large models due to its full sharding of model states.

7 CONCLUSION

We compared and examined three prominent distributed training techniques for deep learning—PyTorch DistributedDataParallel (DDP), Fully Sharded Data Parallel (FSDP), and DeepSpeed ZeRO Stage 1—on the Faster R-CNN object detection model and the COCO 2017 dataset.

Our experiments demonstrated that all three methods work well for training large-scale object detection models. ZeRO Stage 1 achieved the fastest training times, and DDP achieved the slightly best final mAP, so it is a good default option when accuracy matters most. FSDP achieved decent memory efficiency, so it is especially well-suited for memory-constrained environments or very large models. ZeRO Stage 1 achieved a good practical trade-off between performance and memory reduction, with minimal integration effort.

Key takeaways:

- ZeRO Stage 1 delivers optimal training performance along with nice scalability and memory efficiency and is therefore best suited for balanced distributed training.
- DDP is ideal for simple and effective parallel training when memory is not a bottleneck, with extremely low integration overhead.
- FSDP is best suited for training memory-heavy models due to its full sharding of model parameters, which gives significant memory savings at the cost of moderately longer training times.

Future Work

- Evaluate performance on larger models (e.g., Mask R-CNN [3] or Swin Transformer).
- Explore ZeRO Stage 2/3 and compare to FSDP in extreme memory-bound settings.
- Incorporate mixed-precision (AMP) and CPU offloading to further improve efficiency.
- Run tests with 8+ GPUs to analyze scalability in larger distributed clusters.
- Perform full dataset training on 118K training images to study long-term convergence behavior.

REFERENCES

[1] PyTorch Contributors. 2024. PyTorch DistributedDataParallel Documentation. https://pytorch.org/docs/stable/notes/ddp.html.
[2] PyTorch Contributors. 2024. PyTorch FSDP Documentation. https://pytorch.org/docs/stable/fsdp.html.
[3] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. *IEEE International Conference on Computer Vision (ICCV)* (2017), 2961–2969.
[4] Tsung-Yi Lin, Michael Maire, Serge Belongie, et al. 2014. Microsoft COCO: Common Objects in Context. *arXiv preprint arXiv:1405.0312* (2014).
[5] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. https://www.microsoft.com/en-us/research/blog/zero-optimizations-for-training-large-deep-learning-models/.
[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Advances in Neural Information Processing Systems (NeurIPS)* 28 (2015).