Review and Critical Analysis

Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters

Peng, Y., Bao, Y., Chen, Y., Wu, C., & Guo, C. (EuroSys, 2018)

https://dl.acm.org/doi/10.1145/3190508.3190517

1 Summary

This paper introduces **Optimus**, a custom cluster scheduler designed to minimize job completion time (JCT) for Deep Learning (DL) training workloads. The authors identify that existing cluster schedulers (e.g., YARN, Mesos, Kubernetes) allocate static resources, failing to exploit the specific characteristics of DL jobs: their iterative nature, predictable convergence patterns, and the non-linear relationship between resource scaling and training speed.

Optimus implements a dynamic resource allocation mechanism driven by online performance modeling. By predicting both the remaining training steps (convergence) and the processing speed (throughput) for various configurations, Optimus dynamically rightsizes active jobs—adjusting the number of workers and parameter servers at runtime—to maximize the marginal gain in completion time.

2 Methodological Framework

The system relies on fitting two predictive models *online* during job execution to drive a greedy scheduling algorithm.

2.1 Performance Modeling

2.1.1 Convergence Estimation (Steps Remaining)

Unlike batch processing where the total work is known, DL jobs run until a loss metric converges. Optimus models the loss curve $l(k)$ at step k using a non-linear inverse time decay function:

$$l(k) = \frac{1}{\beta_0 \cdot k + \beta_1} + \beta_2 \tag{1}$$

Using a Non-Negative Least Squares (NNLS) solver, the system continuously updates parameters β to predict the total epochs required to reach a target loss threshold.

2.1.2 Throughput Estimation (Speed)

The paper models training speed $f(p, w)$ as a function of parameter servers (p) and workers (w). For **synchronous training**, the model (Eq. 4) accounts for computation (scaled by batch size M/w) and communication overheads:

$$f(p, w) = \left(\theta_0 \cdot \frac{M}{w} + \theta_1 + \theta_2 \cdot \frac{w}{p} + \theta_3 \cdot w + \theta_4 \cdot p\right)^{-1} \tag{2}$$

Terms involving w/p capture the contention at parameter servers, while linear terms capture synchronization barriers. The coefficients θ are learned via regression by sampling different resource configurations during the initial training steps.

2.2 Dynamic Scheduling Algorithm

Optimus formulates resource allocation as a combinatorial optimization problem. Because solving this optimally is NP-hard, they employ a heuristic based on **Marginal Gain**.

At each scheduling interval, the scheduler calculates the "Marginal Gain" for each job: the reduction in remaining JCT normalized by the amount of dominant resource (CPU/GPU) consumed.

$$\text{Gain}_j = \frac{\Delta\text{Time}_j}{\Delta\text{Resource}_j} = \frac{\frac{Q_j}{f(p,w)} - \frac{Q_j}{f(p',w')}}{\text{Resource}(p', w') - \text{Resource}(p, w)} \tag{3}$$

where Q_j is the remaining steps. Resources are iteratively assigned to the job offering the highest marginal gain.

2.3 System Optimizations

- **Task Placement:** Optimus prioritizes placing workers and parameter servers of the same job on the same physical server to minimize cross-rack traffic.
- **Parameter Assignment Algorithm (PAA):** To address load imbalance in MXNet (where parameters are sharded randomly), Optimus introduces a bin-packing strategy that distributes parameters evenly based on tensor size and update frequency.

3 Experimental Evaluation

3.1 Setup

- **Testbed:** Kubernetes cluster with 7 CPU servers and 6 GPU servers (GTX 1080Ti).
- **Workloads:** 9 DL models including ResNet-50, VGG, and DeepSpeech2 using MXNet.
- **Baselines:**
  - **DRF:** Dominant Resource Fairness (default in Mesos/YARN).
  - **Tetris:** A packing-aware scheduler (adapted to use Optimus's speed estimates).

3.2 Key Results

- **Performance:** Optimus reduced the average Job Completion Time (JCT) by **2.39**$\times$ compared to DRF (a 139% performance improvement).
- **Efficiency:** The Makespan (time to finish all jobs) was reduced by **1.63**$\times$ (63% improvement).
- **Accuracy:** The online fitting model achieved prediction errors of $< 10\%$ for training speed and $< 20\%$ for convergence epochs.
- **Overhead:** The cost of dynamic resizing (checkpointing and restarting containers) was measured at $\approx 2.5\%$ of the total makespan, validating the feasibility of dynamic scaling.

4 Critical Analysis

4.1 Strengths

- **White-Box Scheduling:** Unlike "black-box" schedulers that treat jobs as opaque resource consumers, Optimus inspects the training loop (loss curves). This allows it to distinguish between a job that is just starting and one that is nearing convergence, allocating resources where they yield the highest utility.
- **Unified Modeling:** The combination of convergence modeling (how much work is left) and throughput modeling (how fast work is done) provides a rigorous definition of "utility" that surpasses simple fair sharing.
- **Practical Engineering:** The integration with Kubernetes and the handling of practical issues—like stragglers and parameter skew in MXNet—demonstrates high applicability to production environments.

4.2 Limitations and Modern Context

- **Parameter Server Dependency:** The throughput model (Eq. 4) relies heavily on the ratio w/p. Modern deep learning has largely shifted to **AllReduce** architectures (Horovod, NCCL, PyTorch DDP) which do not use dedicated parameter servers. While the core "marginal gain" logic remains valid, the specific throughput equations would need complete reformulation for Ring-AllReduce topologies.
- **Convergence Assumptions:** The inverse time decay model assumes a relatively smooth, monotonic convergence. Modern training often involves learning rate warmups, cosine annealing, or restarts, which produce non-monotonic loss curves that might confound the simple regression model used here.
- **Checkpointing at Scale:** The evaluation used relatively small models on a small cluster. For modern Large Language Models (LLMs) with terabytes of state, "dynamic resizing" via checkpoint-restart is an extremely heavy operation (taking minutes to hours), potentially negating the benefits of fine-grained dynamic scheduling.

5 Conclusion

Optimus is a seminal paper in the field of ML Systems, demonstrating that significant efficiency gains are possible by tailoring cluster management to the mathematical properties of deep learning. It established the standard for *utility-based scheduling* in AI clusters. While the specific focus on Parameter Server tuning reflects the era of 2018, the principle of using online performance modeling to drive resource allocation remains a cornerstone of modern AI infrastructure.