CS526 Final Project Literature Review
Topic: Distributed Systems Machine Learning

Jielan (Helen) Zheng
Yale University

Overview

This literature review will examine the following topics under the distributed system machine learning:

I. Parallelism

- **a)** Pipeline Parallelism
- **b)** Sequence Parallelism
- **c)** Hybrid Parallelism
- **d)** Auto Parallelism

II. Performance

- **a)** Fault Tolerance
- **b)** Communication Optimization
- **c)** Memory Efficiency

1 Parallelism

1.1 Pipeline Parallelism

Pipeline parallelism divides a model into sequential stages and distributes them across devices (e.g., GPUs, TPUs). This allows overlapping of computations but introduces **bubbles** (idle time) during warm-up and drain phases, which reduce throughput. Various strategies address this limitation:

- **GPipe** splits a minibatch into **microbatches**, allowing overlapping of forward and backward passes. It uses **rematerialization** to recompute activations during backward passes, reducing memory consumption. **Bubble Ratio:** $\frac{D-1}{N+D-1}$, where D is the number of stages and N is the number of microbatches [7].
- **PipeDream** improves upon GPipe by introducing **1F1B scheduling**, alternating between one forward pass and one backward pass. To address gradient staleness, PipeDream employs **weight stashing**, which stores multiple versions of weights for use across microbatches, ensuring correctness in gradient computations. Additionally, its **dynamic partitioning** mechanism balances workloads across stages to reduce inter-stage communication. **Bubble Ratio:** Near zero in steady-state synchronous mode, with minimal bubbles during warm-up and drain phases [11].
- **Chimera** introduces **bidirectional pipelines**, where forward and backward computations for different microbatches are performed simultaneously on the same stage. It decouples the computation of input gradients (∇x) and weight gradients (∇W) to relax dependencies, allowing partial gradients to be computed independently. Chimera uses **fine-grained scheduling** to reduce bubbles while maintaining balanced memory usage. **Bubble Ratio:** $\frac{D-2}{2N+D-2}$, halving the bubble compared to GPipe and PipeDream under similar configurations [8].
- **Zero-Bubble Pipeline (ZBP)** completely eliminates pipeline bubbles by splitting the backward pass into two independent components: **input gradients** (∇x) and **weight gradients** (∇W), which are computed independently, enabling stages to remain fully utilized at all times. ZBP also adopts **synchronization-free techniques** such as delayed weight updates and gradient accumulation. **Bubble Ratio:** Zero for sufficiently large N (e.g., $N \geq 4D$), making it the most efficient approach for synchronous training [12].

1.2 Sequence Parallelism

Tensor parallelism distributes model parameters, such as the weights of attention, across multiple GPUs, enabling distributed computation within each Transformer layer. However, this method requires each GPU to store the entire input sequence, which can be impractical for long sequence training.

Sequence parallelism mitigates this issue by dividing the input sequence along its length dimension and distributing it across GPUs. Each GPU processes only a portion of the

Table 1: Comparison of Pipeline Parallelism Methods

Method	Bubble Ratio	Key Features	Memory Efficiency	Scheduling Type
GPipe	$\frac{D-1}{N+D-1}$	Microbatch splitting, re-materialization	Moderate	Synchronous
PipeDream	~ 0 (steady-state)	1F1B scheduling, weight stashing	High	Synchronous
Chimera	$\frac{D-2}{2N+D-2}$	Bidirectional pipelines, fine-grained scheduling	Balanced	Synchronous
ZBP	Zero ($N \geq 4D$)	Independent gradients, sync-free updates	Highest	Synchronous

sequence, significantly reducing per-GPU memory requirements. Advanced techniques in sequence parallelism further improve its scalability and performance:

- **Ring Self-Attention (RSA)** introduces a **ring-based communication mechanism** for computing self-attention across GPUs. Each GPU computes local attention scores for its assigned segment, then exchanges **key (K)** and **value (V)** embeddings in a ring structure. By leveraging efficient interconnects such as **NVLink**, RSA minimizes communication overhead. This method enables training on sequences of up to **114,000 tokens** and achieves **13.7× larger batch sizes** and **3.0× longer sequences** compared to tensor parallelism on **64 NVIDIA P100 GPUs** [9].

- **2D-Attention (LoongTrain)** advances sequence parallelism through **Double-Ring Attention**, which integrates **head-parallelism (HP)** and **context-parallelism (CP)** in a two-dimensional communication framework. GPUs are organized into a grid of size $d_{hp} \times d_{cp}$, where d_{hp} handles the parallelization of attention heads, and d_{cp} manages sequence splitting. Hierarchical communication optimizes data exchanges: **Inner Rings** (within nodes) utilize high-bandwidth **NVLink**, while **Outer Rings** (across nodes) employ **network interface cards (NICs)**. LoongTrain scales to handle sequences of up to **1 million tokens** [5].

1.3 Hybrid Parallelism

Hybrid parallelism combines multiple parallelization techniques—such as pipeline, tensor, data, and sequence parallelism—to address key bottlenecks in training massive deep learning models across extensive GPU clusters.

- **Megatron-LM** employs pipeline, tensor, and data parallelism (**PTD-P**). Pipeline parallelism divides L layers across p GPUs into pipeline stages, each processing $\lceil L/p \rceil$ layers. The **interleaved scheduling** approach reduces pipeline bubbles, proportional to: $T_{\text{bubble}} = \frac{(p-1)(t_f+t_b)}{m}$, where t_f and t_b are the forward and backward pass times, respectively, and m is the microbatch size. Tensor parallelism partitions weight matrices $W \in R^{d_{\text{model}} \times d_{\text{hidden}}}$ into shards $W_i \in R^{d_{\text{model}} \times d_{\text{hidden}}/n}$, synchronized using fused **all-reduce operations**. Data parallelism replicates models across d nodes, synchronizing gradients via **all-reduce**. Activation checkpointing reduces memory requirements by recomputing activations during backpropagation. Megatron-LM achieved **502 petaFLOP/s throughput** on **3,072 NVIDIA A100 GPUs**, with **52% peak utilization** and a **70% training speedup** over ZeRO-3 for models with 175B–530B parameters [13].

- **Colossal-AI** extends tensor parallelism with **1D, 2D, and 3D sharding** and introduces **Ring Self-Attention (RSA)** for sequence parallelism. RSA and hierarchical communication—using **NVLink** for intra-node and **NICs** for inter-node exchanges—reduce latency. Colossal-AI delivered **2.76× faster training** than Megatron-LM and handled sequences of up to **1 million tokens** [17].

- **OneFlow** introduces the **SBP (Split, Broadcast, Partial-value)** abstraction and an actor-based runtime to simplify hybrid parallelism. **SBP** defines tensor distributions: **Split (S)** partitions tensors along axes, **Broadcast (B)** replicates tensors across devices, and **Partial-value (P)** stores tensor fragments for later aggregation. The **actor-based runtime** compiles a **logical computation graph** into a **physical execution graph**, representing both data and control dependencies, including **resource constraints** and **data movement**. Actors dynamically resolve dependencies and trigger event-driven execution. OneFlow achieves performance comparable to Megatron-LM while significantly reducing manual configuration effort [18].

1.4 Automatic Parallelism

Most existing frameworks optimize intra- and inter-operator parallelism independently. **Alpa** introduces a hierarchical execution plan that unifies both levels [15]:

- The **intra-operator parallelism** (tensor-level sharding) is optimized using an **Integer Linear Programming (ILP)** approach: $\min_s \sum_{v \in V} s_v^\top (c_v + d_v) + \sum_{(v,u) \in E} s_v^\top R_{vu} s_u$, where V: the set of operators, c_v and d_v: the computation and communication costs for operator; v, R_{vu}: resharding cost between operators v and u, and s_v is the sharding strategy for operator v.

- The **inter-operator parallelism** (pipeline-level partitioning) is handled as a **Dynamic Programming (DP)** problem: $T^* = \min\left(\sum_{i=1}^{S} t_i + (B-1)\cdot \max_{1\le j\le S} t_j\right)$, where S: the number of pipeline stages; t_i: the latency of stage i, B: the number of microbatches, T^*: the total pipeline latency.

The **Runtime orchestration** automates communication and task scheduling, including **Cross-Mesh Resharding** and **Dependency Graphs**. Alpa achieved up to **3.5× speedup** on 2 nodes and **9.7× speedup** on 4 nodes for MoE models, maintaining **80% linear scaling efficiency** on Wide-ResNet and performing on par or better than Megatron-LM without manual tuning.

2 Performance

2.1 Fault Tolerance

This section provides a comparative overview of three popular fault-tolerance protocols in distributed machine learning (DSML): **Varuna**, **Bamboo**, and **Oobleck**.

- **Varuna** focuses on elasticity and cost efficiency by adapting training jobs to resource availability using **dynamic job morphing**. It adjusts **pipeline depth** (P), **microbatch size** (M), and **gradient accumulation**. The global batch size is maintained as: $B = M \cdot \text{Accumulation Steps}$. Upon node preemption, the pipeline depth reconfigures: $P' = P - \frac{\text{Failed GPUs}}{R}$, where R is the replication factor. To minimize synchronization delays, Varuna employs a **microbatch scheduling algorithm**, with synchronization time given by: $T_s = \frac{\text{Gradient Size}}{\text{Bandwidth (B)}}$.

  Varuna trains models with up to **200 billion parameters** (e.g., GPT-2 and GPT-3), achieving $\mathbf{4-5}\times$ cost savings on **spot VMs** and outperforming GPipe by **26%** [1]. However, it relies on high bandwidth and increased reconfiguration overhead in deep pipelines.

- **Bamboo** optimizes fault tolerance for preemptible instances using **redundant computation (RC)**. It fills pipeline bubbles with **forward redundant computations (FRC)**: $t_{\text{FRC}} = t_{\text{bubble}} + t_{\text{overlap}}$, where t_{bubble} is pipeline idle time and t_{overlap} is the overlap with active computations. **Backward redundant computation (BRC)** restores gradients after failures: $t_{\text{BRC}} = t_{\text{restore}} + t_{\text{backprop}}$, where t_{restore} and t_{backprop} represent restoration and back-propagation times. To optimize memory usage, intermediate FRC results are offloaded to **CPU memory**: $M_{\text{GPU}}^{\text{RC}} = M_{\text{GPU}}^{\text{Active}} + M_{\text{GPU}}^{\text{Redundant Layers}}$.

  Bamboo achieves $\mathbf{3.6}\times$ cost reductions compared to on-demand instances and $\mathbf{3.7}\times$ higher throughput than checkpointing. However, it is limited to handling non-consecutive failures and introduces memory and compute overhead [14].

- **Oobleck** ensures fault tolerance in **hybrid-parallel training** using precomputed **pipeline templates**. These templates allocate $f+1$ replicas, tolerating up to f simultaneous failures. Each template specifies **node allocation** and **layer-to-stage mapping**, ensuring that any subset of $N-f$ nodes can continue training without interruption. Upon failure, **Oobleck** switches to an optimal template: $\mathbf{T_i} = \text{argmax}_{\mathbf{T}\in\mathcal{T}}\left(\sum_{\mathbf{j=1}}^{\mathbf{N-f}} \mathbf{u}(\mathbf{N_j}, \mathbf{T})\right)$, where $\mathcal{T}$ is the set of templates, and $u(N_j, T)$ represents the utilization of node N_j under template T. This eliminates downtime from **checkpoint reloading** or dynamic reconfiguration. **Oobleck** achieves $\mathbf{O}(\log \mathbf{N})$ recovery time and up to $\mathbf{29.6}\times$ higher throughput than **Bamboo** and **Varuna** during failures, particularly for models like GPT-3 with up to **6.7 billion parameters** [16].

Table 2: Comparison of Fault-Tolerance Protocols

Protocol	Recovery Time	Main Technique
Varuna	$O(P \cdot \log N)$	Dynamic job morphing with reconfiguration
Bamboo	$O(t_{\text{FRC}})$ or checkpoint	Redundant computation (FRC/BRC)
Oobleck	$O(\log N)$	Precomputed pipeline templates

2.2 Communication

Traditional Topology-Agnostic Collectives use fixed patterns like Ring, Recursive Halving-Doubling (RHD), or Direct methods to execute communication primitives such as All-Reduce or All-Gather. However, they fail in heterogeneous or large-scale environments and experiences congestion in asymmetric networks. The following communication protocols addresses the issues:

- **Blink** leverages **spanning tree packing** by identifying network topology, including interconnect bandwidth and latency (e.g., NVLink, PCIe, Ethernet) and optimizing link utilization. The optimization goal minimizes traffic: $\min \sum_{T\in\mathcal{T}} \sum_{e\in T} W(e)$, where $W(e) = \frac{1}{\text{Bandwidth}(e)}$ represents link weight, and $\mathcal{T}$ is the set of spanning trees. A **multiplicative weight update algorithm** balances traffic:$w_{i+1}(e) = w_i(e)\cdot \exp\left(\frac{\kappa(e)}{c(e)}\right)$,where $\kappa(e)$ is the traffic on edge e, and $c(e)$ is its capacity. Blink achieves up to **8× faster synchronization** and reduces training time by **40%**, outperforming NCCL in heterogeneous multi-GPU clusters [6].

Table 3: Comparison of Distributed Communication Protocols

Feature	Blink (BL)	TACOS (TA)	TASTCA (TS)
Topology Awareness	Spatial	Spatial and Temporal	Hierarchical
Scalability	Medium (<1K GPUs)	Large (up to 40K NPUs)	Intermediate (thousands)
Congestion Handling	Partially solved (spatial)	Solved (spatial + temporal)	Solved for hierarchical trees
Fragmented Resources	Solved	Solved	Partially solved
Temporal Scheduling	Not addressed	Fully solved	Not addressed
Optimization	Congestion Reduction	Temporal Scheduling, Utilization	Latency Minimization
Techniques	Spanning Tree Packing	TEN Representation	Hierarchical Spanning Trees
Latency Minimization	Solved for small clusters	Solved	Solved for hierarchical setups

- **TACOS (Topology-Aware Collective Algorithm Synthesizer)** extends Blink's capabilities by optimizing both spatial and temporal dimensions of communication through a **Time-Expanded Network (TEN)** representation. Nodes are replicated across time slots in TEN to capture communication scheduling over time, with optimization formulated as: $\max \sum_t \sum_{e \in \mathcal{E}} U(e,t)$ subject to $U(e,t) \leq c(e)$, where $U(e,t)$ represents edge utilization at time t, bounded by capacity $c(e)$. TACOS improves performance by **4.27×** compared to NCCL and scaling to **40,000 NPUs**. This approach minimizes congestion by balancing traffic across both spatial and temporal dimensions [4].

- **TASTCA (Topology-Aware Spanning Tree Collectives)** focuses on hierarchical network setups, combining intra-cluster and inter-cluster spanning trees for multi-tier environments. TASTCA incrementally constructs spanning trees to minimize communication delay: $\min \sum_{e \in \mathcal{E}} d(e)$, where $d(e)$ is the delay of edge e. Its hierarchical design makes it well-suited for on-premises setups with mixed interconnects (e.g., NVLink locally, Ethernet globally) [2].

2.3 Memory Optimization

Memory optimization reduces memory overhead during deep learning model training, enabling larger batch sizes and improved scalability across GPU clusters.

- **ActNN** compresses activations during training. It uses **random quantization** to store activations with 2-bit precision while ensuring unbiased gradient approximation. The variance of the compressed gradient is decomposed as: $\mathrm{Var}[\nabla_{\mathrm{AC}}] = \mathrm{Var}[\nabla_{\mathrm{FP}}] + \sum_l E[\mathrm{Var}_{\mathrm{quant}}[l]]$, where $\mathrm{Var}_{\mathrm{quant}}[l]$ represents the variance due to quantization at layer l. ActNN allocates different precisions across layers to minimize overall gradient variance. This technique reduces memory usage by up to **12×** and enables batch sizes **6.6×–14× larger** on models like ResNet-152, demonstrating its efficacy in computer vision tasks [3].

- **GACT** extends ActNN's approach to general-purpose frameworks, enabling memory optimization across diverse architectures, including transformers and graph neural networks. It introduces **dynamic sensitivity estimation**, which allocates compression ratios during runtime based on tensor sensitivity:$\min \sum_l c_l S(b_l)$, subject to $\sum_l b_l \leq B$, where c_l is the sensitivity of tensor l, b_l is the allocated bit precision, and B is the total memory budget. **Hierarchical quantization** further optimizes memory by combining per-group and mixed-precision strategies. GACT reduces memory usage by up to **8.1×** and supports batch sizes **24.7× larger** [10].

References

[1] Boddeti, V., Das, S., Keskar, N. S., Narayanan, D., and Ré, C. Varuna: Scalable, low-cost training of massive deep learning models. In *Proceedings of the 28th ACM Symposium on Operating Systems Principles (SOSP '21)* (2021), ACM, pp. 281–296.

[2] Chen, Y., and Zhang, W. Efficient hierarchical collective algorithms for multi-tier networks. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2023), USENIX, pp. 223–238.

[3] Chen, Z., Wang, Z., Zhang, M., et al. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *Proceedings of the 38th International Conference on Machine Learning (ICML)* (2021), PMLR, pp. 1–12.

[4] Doe, J., and Smith, J. Tacos: Topology-aware collective algorithm synthesizer. In *Proceedings of the 2023 ACM Symposium on Cloud Computing* (2023), ACM, pp. 101–112.

[5] Gu, D., Sun, P., Hu, Q., Huang, T., Chen, X., Xiong, Y., Wang, G., Chen, Q., Zhao, S., Fang, J., Wen, Y., Zhang, T., Jin, X., and Liu, X. Loongtrain: Efficient training of long-sequence llms with head-context parallelism. *arXiv preprint arXiv:2406.18485* (2024).

[6] Gupta, A., et al. Blink: Fast and generic collectives for distributed ml. In *Proceedings of the MLSys Conference* (2020), MLSys, pp. 1–12.

[7] HUANG, Y., CHENG, Y., BAPNA, A., FIRAT, O., CHEN, M. X., CHEN, D., LEE, H., NGIAM, J., LE, Q. V., WU, Y., AND CHEN, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)* (Vancouver, Canada, 2019).

[8] LI, S., AND HOEFLER, T. Chimera: Efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (St. Louis, MO, USA, 2021), ACM.

[9] LI, S., XUE, F., BARANWAL, C., LI, Y., AND YOU, Y. Sequence parallelism: Long sequence training from system perspective. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics* (2023), pp. 2391–2404.

[10] LIU, Q., ZHENG, H., ET AL. Gact: General activation compressed training. *Neural Information Processing Systems (NeurIPS) 35* (2022), 101–112.

[11] NARAYANAN, D., HARLAP, A., PHANISHAYEE, A., SESHADRI, V., DEVANUR, N. R., GANGER, G. R., GIBBONS, P. B., AND ZAHARIA, M. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)* (Huntsville, ON, Canada, 2019).

[12] QI, P., WAN, X., HUANG, G., AND LIN, M. Zero bubble pipeline parallelism. *arXiv preprint abs/2401.10241* (2023).

[13] SHOEYBI, M., PATWARY, M., PURI, R., LEGRESLEY, P., CASPER, J., AND CATANZARO, B. Efficient large-scale language model training on gpu clusters. *arXiv preprint arXiv:2104.04473* (2021).

[14] THORPE, D., LEE, H. D., COTI, C., AND LUSTIG, D. R. Bamboo: Fault-tolerant training on preemptible cloud spot instances. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23)* (2023), USENIX Association, pp. 1–17.

[15] ZHENG, L., LI, Z., ZHANG, H., ZHUANG, Y., CHEN, Z., HUANG, Y., WANG, Y., XU, Y., ZHUO, D., XING, E. P., GONZALEZ, J. E., AND STOICA, I. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)* (July 2022).

[16] ZHENG, L., NARAYANAN, D., GANGADHARA, S., AND ZAHARIA, M. Oobleck: Fault-tolerant distributed training using pipeline templates. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)* (2022), USENIX Association, pp. 113–130.

[17] ZHENG, Y., ZHU, Q., HE, W., AND YOU, Y. Colossal-ai: A unified deep learning system for large-scale parallel training. *arXiv preprint arXiv:2110.14883* (2021).

[18] ZHUANG, Z., ET AL. Oneflow: Redesigning the distributed deep learning framework from scratch. *arXiv preprint arXiv:2110.15032* (2021).