Data Parallelism

FID3024 Systems for Scalable Machine Learning

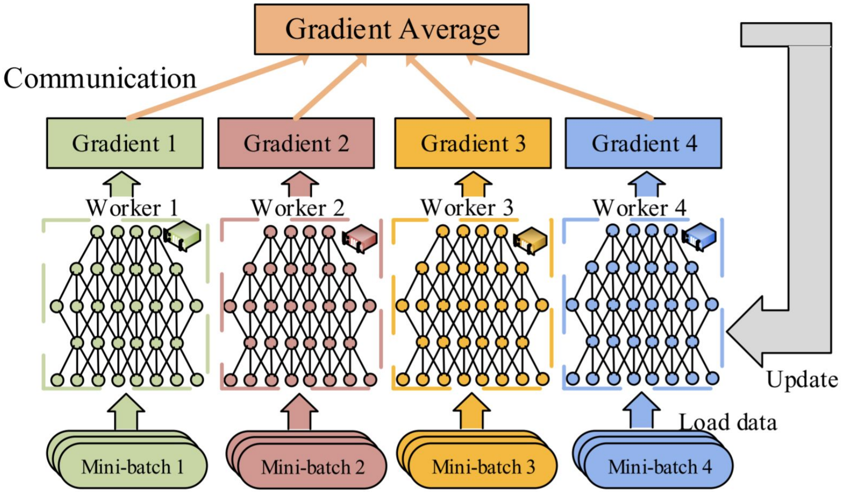
Sina Sheikholeslami, Dominik Fay, Federico Baldassarre, Matteo Gamba
19 October 2020

Communication-Efficient Distributed Deep Learning
A Comprehensive Survey

Zhenheng Tang, Shaohuai Shi, Xiaowen Chu, Wei Wang, Bo Li

2020

Data Parallelism

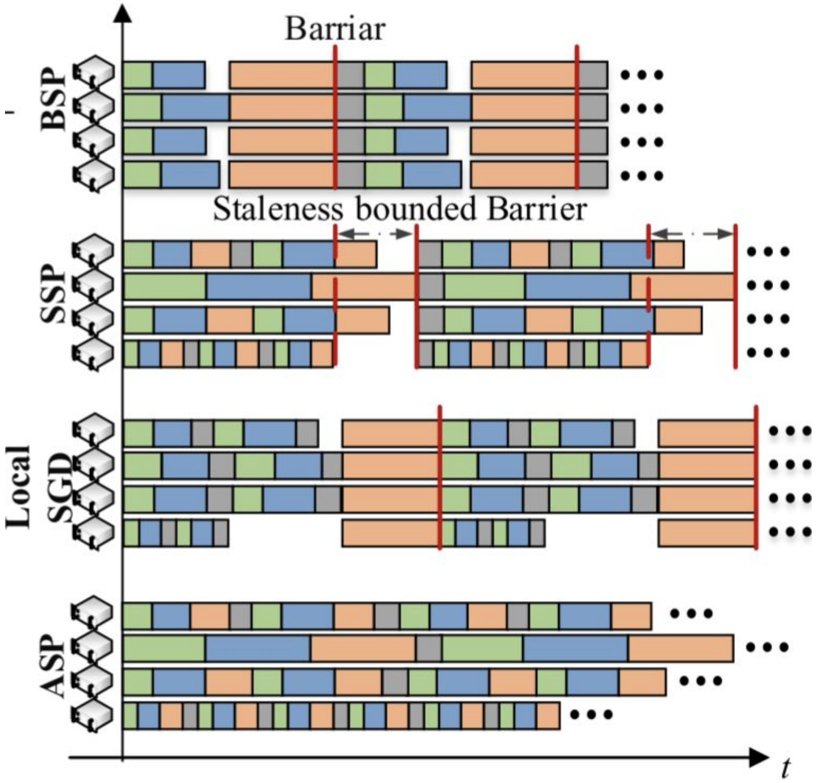


(Tang et al., 2020)

Four Dimensions of Data Parallelism

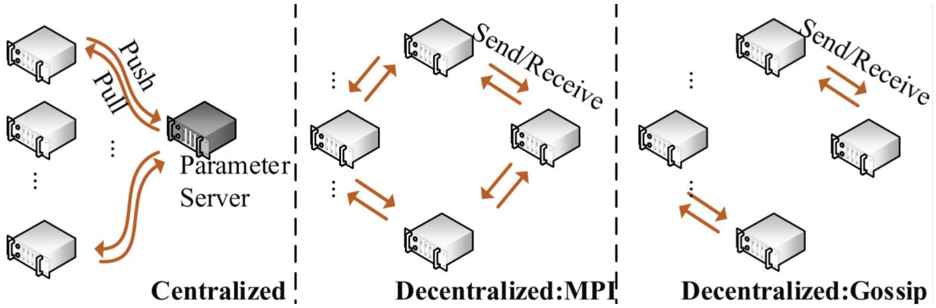
- **When?:** Communication Synchronization and Frequency
  - Synchronous, Stale-Synchronous, Asynchronous, Local SGD
- **Who?:** Aggregation Algorithm (System Architecture)
  - Parameter Server, All-Reduce, Gossip
- **What?:** Communication Compression
  - Quantization, Coding, Sparsification
- **How?:** Parallelism / Scheduling of Computations and Communications
  - Pipelining, Scheduling

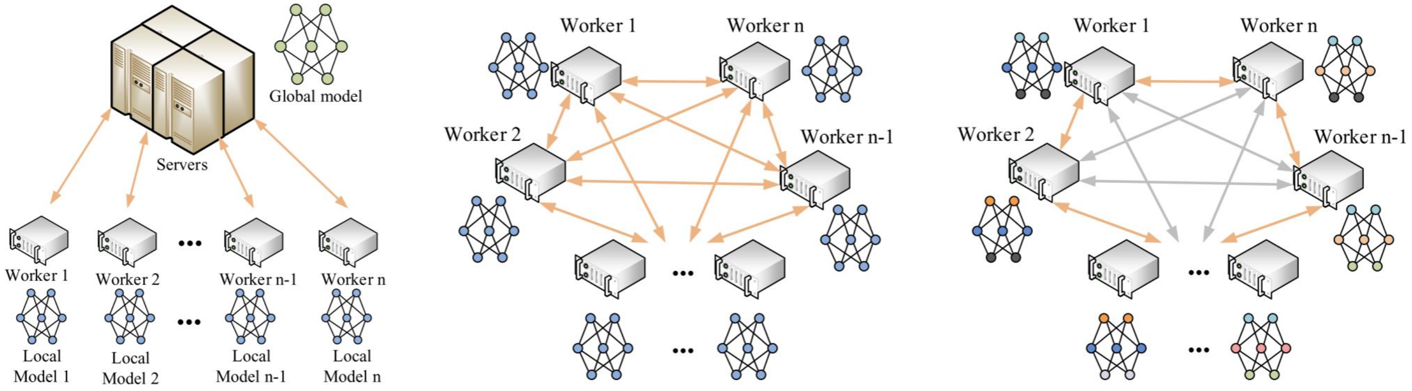
WHEN: Communication Synchronization & Frequency



(Tang et al., 2020)

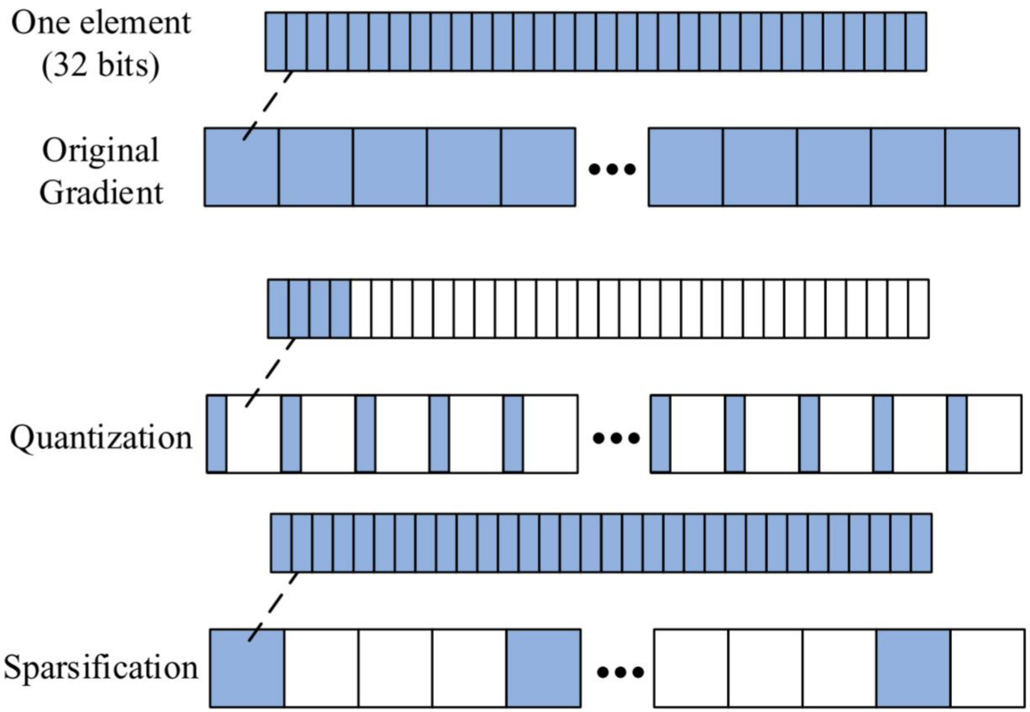
WHO: Aggregation Algorithm & System Architecture





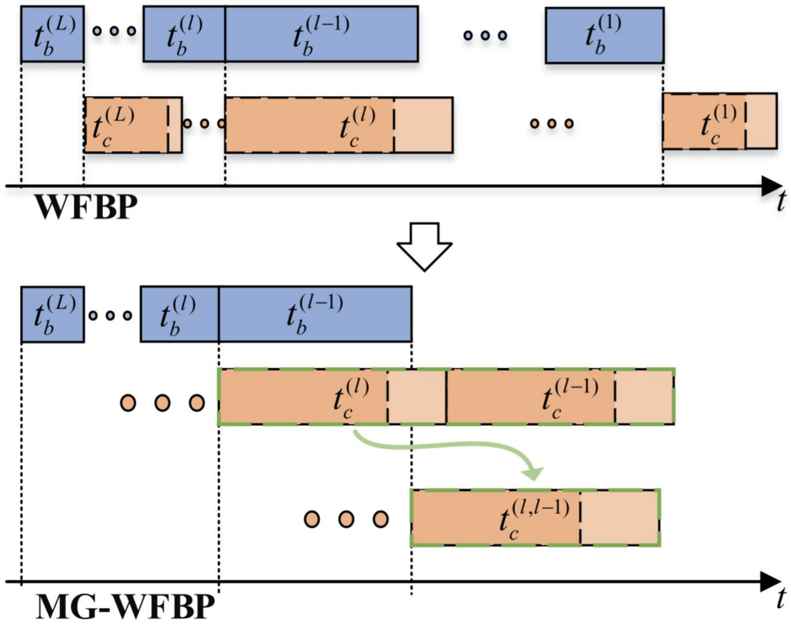
(Tang et al., 2020)

WHAT: Communication Compression



(Tang et al., 2020)

HOW: Parallelism & Scheduling of Comm. & Comp.



(Tang et al., 2020)

Auxiliary Techniques

- Error Accumulation
- Momentum Correction
- Local Gradient Clipping
- Warm-up Training

CodedReduce: a Fast and Robust Framework for Gradient Aggregation in Distributed Learning

Amirhossein Reisizadeh, Saurav Prakash, Ramtin Pedarsani, Amir Salman Avestimehr

2020

Introduction

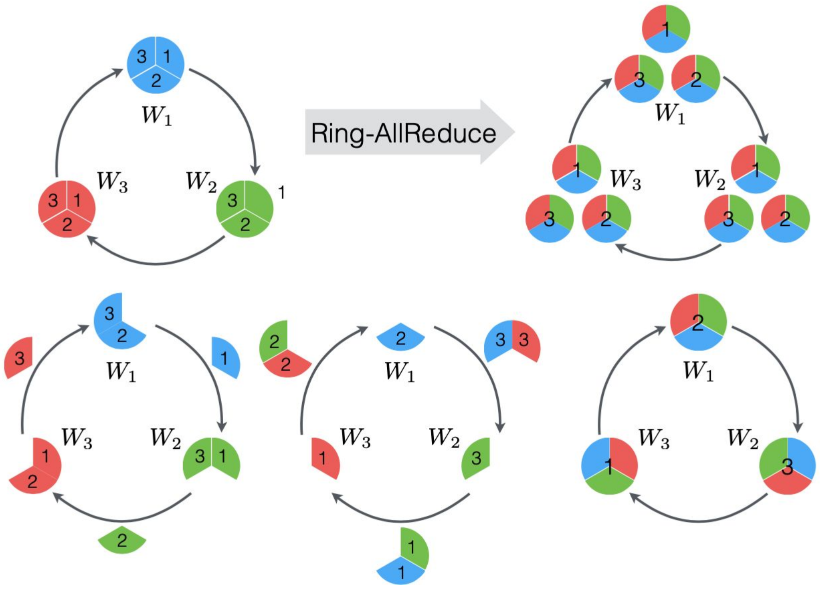
Two bottlenecks in synchronous SGD:

- Communication bandwidth
- Stragglers' delays

The former can be addressed with Ring-AllReduce (RAR) and the latter with Gradient Coding (GC).

But can we have both at once?

Background - Ring-AllReduce

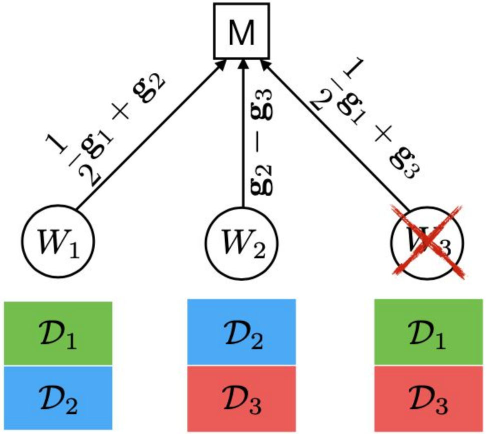


Dataset is uniformly partitioned among N workers.

In each communication round, they send 1/N fraction of the gradient to their neighbor.

No straggling resilience.

Background - Gradient Coding



For robustness against S stragglers, each worker receives (1+S)/N fraction of the data set.

Master can recover the full gradient from N-S workers due to redundancy.

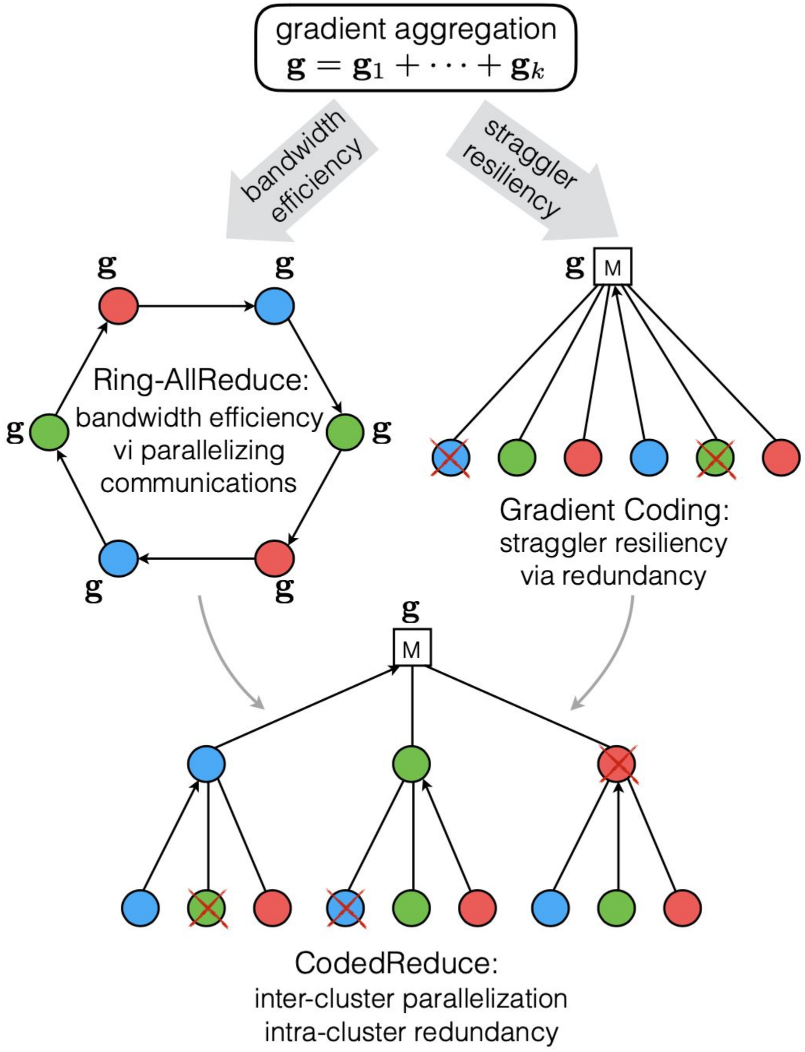
O(1) parallelization gain for fixed straggler ratio.

Method - CodedReduce

Combine redundancy and parallelization via a tree structure

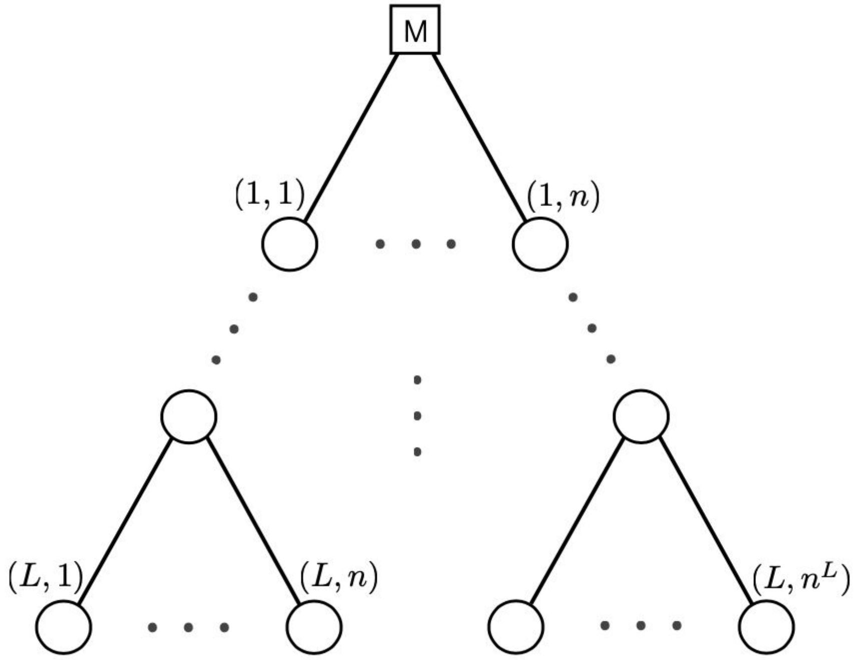
- L layers, n children per parent
- N=n^L + n^(L-1) + … n workers in total

Essentially, this is hierarchical Gradient Coding.



Method - CodedReduce

1. **Allocation**
   Recursively, every node takes its fraction of the data and passes the rest on to its children.
2. **Execution**
   After computing the partial gradient, each node passes it on to its parent, starting at the leaves. Upon receiving n-s messages, the parent passes its aggregated gradient on.



Theoretical analysis

- For the same straggler resilience, CodedReduce has a lower computation load per node (fraction of the dataset), compared to Gradient Coding:
  - GC: $\frac{\alpha N+1}{N} \approx \alpha$ CR: $1/\sum_{l=1}^{L}\left(\frac{n}{\alpha n+1}\right)^{l} \approx \alpha^{L}$.

- Assuming exponentially distributed computation times, the expected run time scales as
  - GC: $\Theta(1) + \Theta(N)$ CR: $\Theta(1) + \Theta(n)$

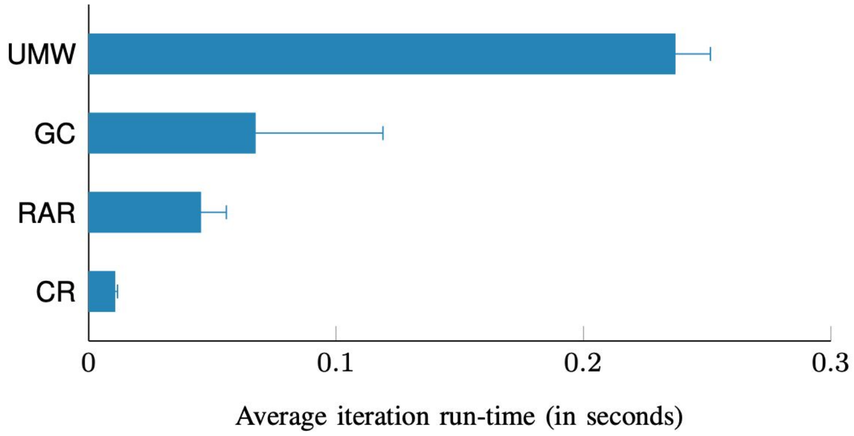
Theoretical analysis

Scheme	Straggler Resiliency (α)	Communication Parallelization Gain (β)
RAR	0	$\Theta(N)$
GC	r	$\Theta(1)$
CR	$r^{1/L}$	$\Theta\left(N^{1-1/L}\right)$

Empirical evaluation

Training a linear model on N=84 workers

UMW = Uncoded Master-Worker

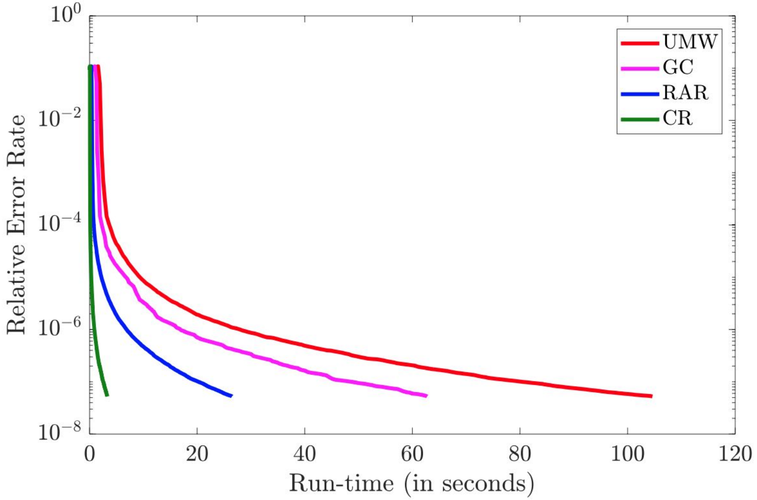


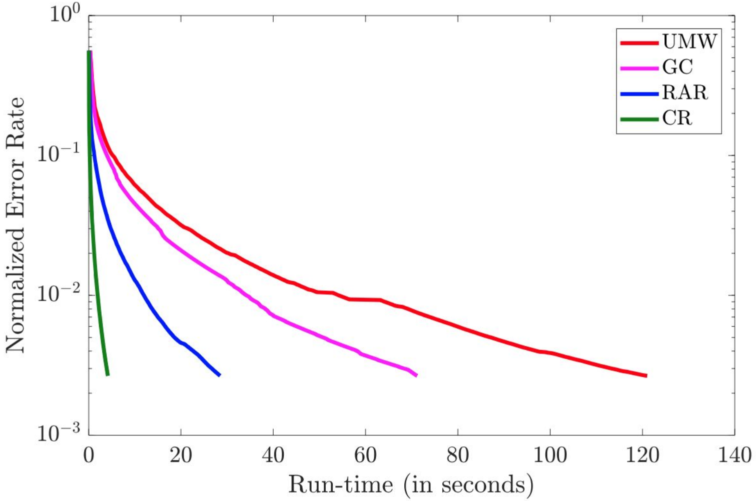
Empirical evaluation

N=156 workers

Top: Logistic regression (real data)

Bottom: Linear regression (synthetic)





Discussion

Experiments: Very small models only (~5000 parameters)

- How does the efficiency depend on model size?
- Overhead cost of data distribution?

How many actual stragglers were there? Was the exponential model accurate?

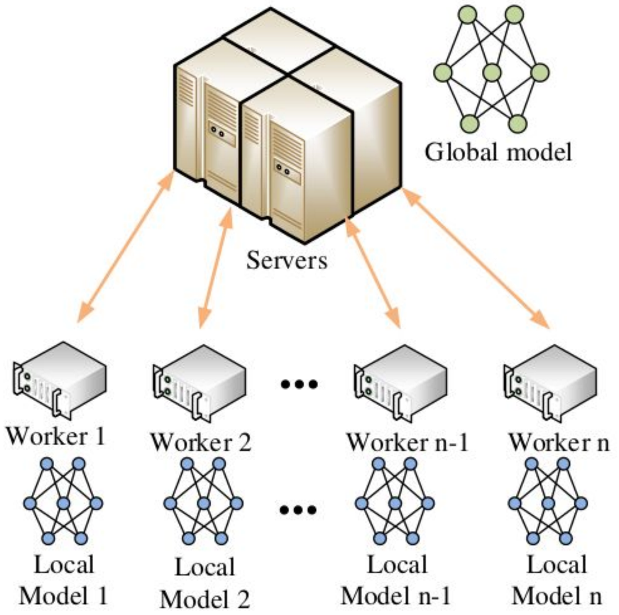
TicTac: Accelerating Distributed Deep Learning with Communication Scheduling

Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, Roy H. Campbell

SysML 2019

Context

- Parallel scheduling of communication and computation
- Distributed SGD with Parameter Server architecture

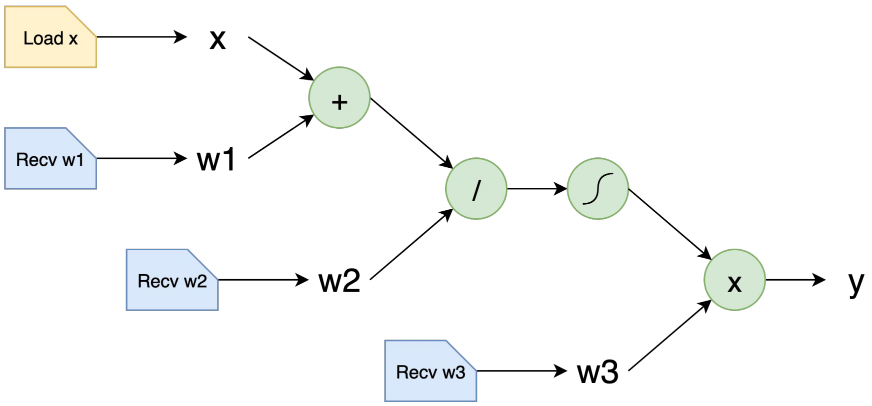


~~Problems~~ Opportunities

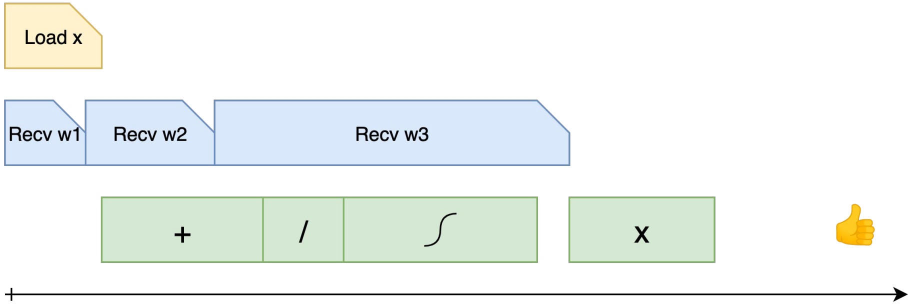
- Common DL frameworks model operation as a DAG
- Computation and communication can overlap
  - Computation happens on CPU/GPU
  - Communication happens on NIC
- DAG execution order is not optimized for network communication
  - PS sends params to workers in random order
  - Each worker executes DAG ops in random order
- Suboptimal overlap ➔ suboptimal GPU utilization

Example: forward pass

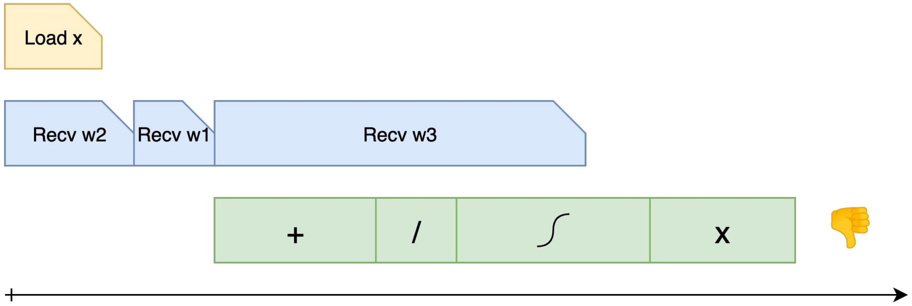
Ops dependencies:



Valid scheduling 1:

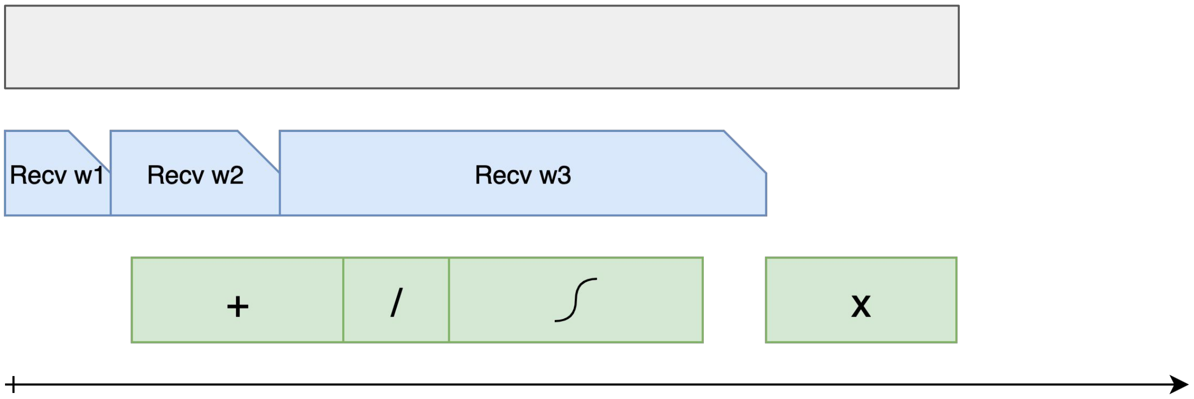


Valid scheduling 2:



Metrics

- N Network communication time
- C Computation time
- T Total iteration time

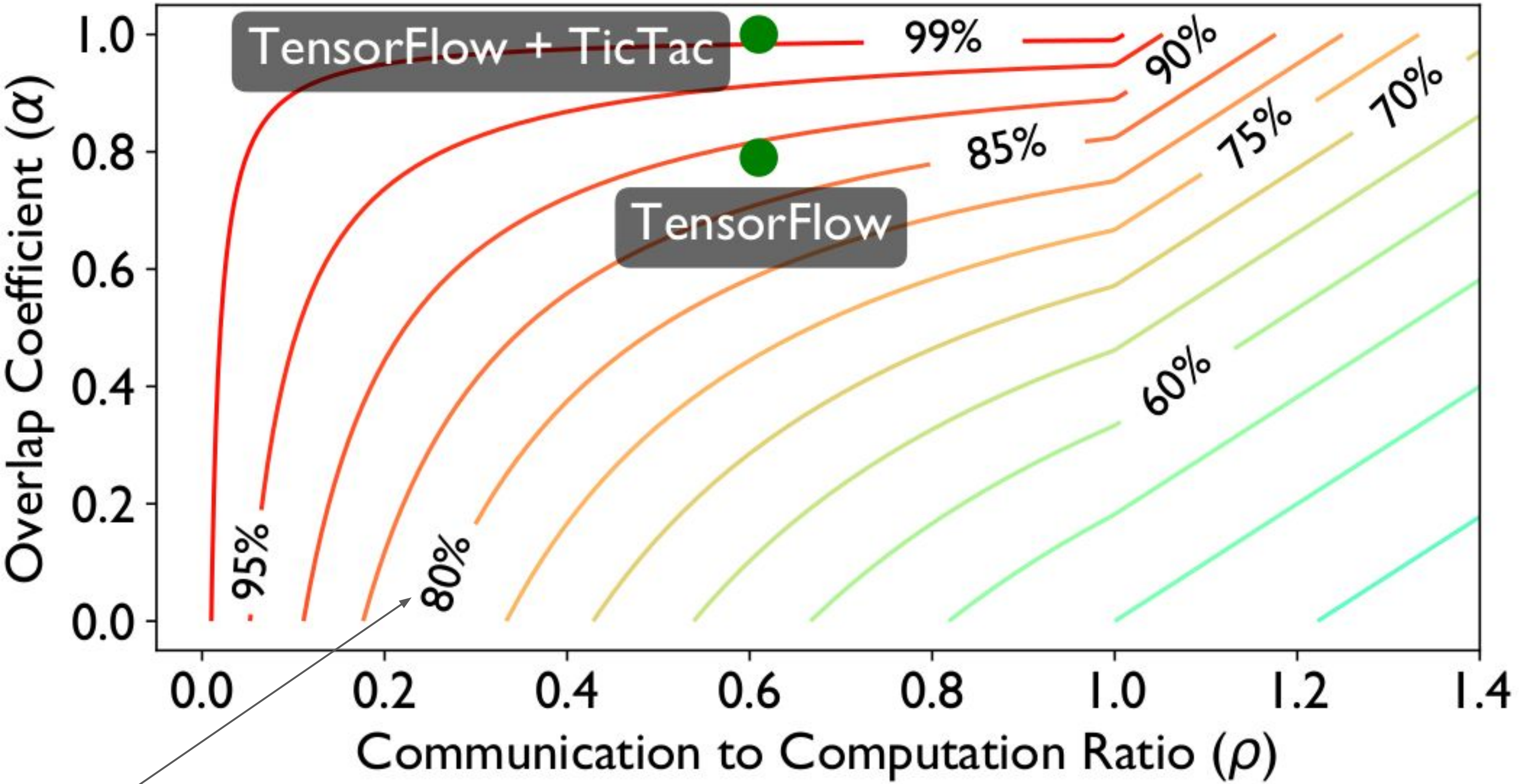


Metrics

- $\boxed{N}$ Network communication time
- $\boxed{C}$ Computation time
- $\boxed{T}$ Total iteration time

- Comm/comp ratio $\rho = \frac{N}{C}$
- Overlap coefficient $\alpha = \frac{(N+C)-T}{\min(N,C)}$
- GPU utilization $U = \frac{C}{T} = \frac{C}{N+C-\alpha \min(N,C)} = \frac{1}{1+\rho-\alpha \min(\rho,1)}$

$$\alpha = \frac{(N+C)-T}{\min(N,C)}$$



$$\rho = \frac{N}{C}$$

GPU utilization

$$U = \frac{C}{T} = \frac{C}{N+C-\alpha\min(N,C)} = \frac{1}{1+\rho-\alpha\min(\rho,1)}$$

Proposed solution

- Heuristic scheduling algorithm to increase GPU utilization
  - Forward pass: PS should send params to workers so that pending operations can be executed as soon as possible
  - Backward pass: workers should prioritize computing gradients that can be sent to the PS as soon as possible

- Strategies
  - TIC: assume every computation op takes the same time
  - TAC: include execution time of computation ops in the scheduling heuristic

Implementation

- Small modifications to TensorFlow scheduler

Algorithm 1: Property Update Algorithm

```
// Update properties for the given the set of
   outstanding read ops R
Function UpdateProperties(G, Time, R):
    foreach op ∈ G do
        op.M ← Σ_{∀r∈op.dep∩R} Time(r);
    end
    foreach op ∈ R do
        op.P ← 0;
        op.M⁺ ← +∞;
    end
    foreach op ∈ G − R do
        D ← op.dep ∩ R;
        if |D| = 1 then
            ∀r ∈ D : r.P ← r.P + Time(op);
        end
        if |D| > 1 then
            ∀r ∈ D : r.M⁺ ← min{r.M⁺, op.M};
        end
    end
end
```

Algorithm 3: Timing-Aware Communication Scheduling (TAC)

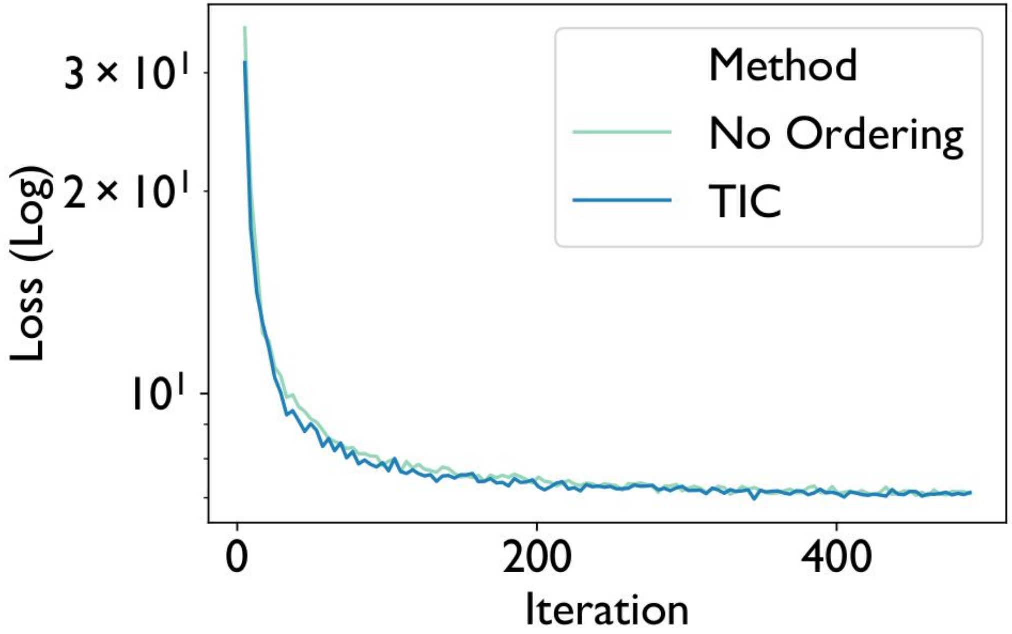
```
// Compare two given recv ops
Function Comparator(Op_A, Op_B): Bool
    A ← min(P_A, M_B);
    B ← min(P_B, M_A);
    if A ≠ B then
        return A < B            # Main comparison
    else
        return M⁺_A < M⁺_B      # Tie breaker
    end
end
Function TAC(G, Time)
    FindDependencies(G) ;
    R ← {op|∀op in G, op is recv};
    count ← 0;
    while R is not empty do
        UpdateProperties(G,R,Time);
        Find the minimum op from R wrt Comparator;
        Remove op from R;
        op.priority ← count;
        count ← count + 1;
    end
end
```

Experiments

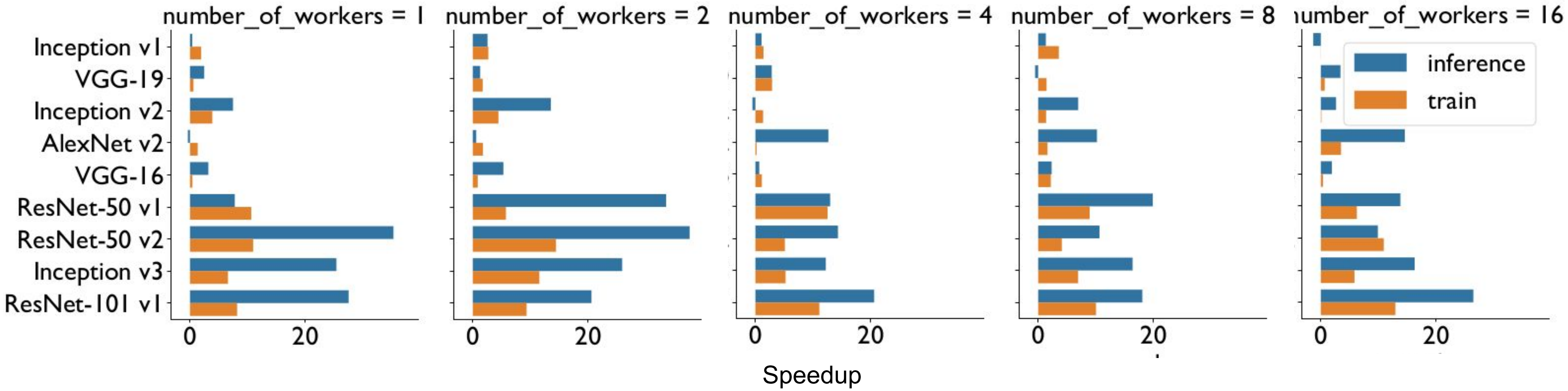
- Hardware setups
  - GPU cluster (reasonably expensive)
  - CPU cluster
- Workers: 2-16
- Parameter servers: 1-4
- Variable batch size (inference only)
- 10 architectures for computer vision

Experiments: training dynamics

- Convergence, generalization, etc. are not affected



Experiments: scaling up workers and PSs



Discussion: VGG vs. ResNet vs. Inception

- VGG: pretty much linear DAG, not many optimization opportunities
- ResNet: several skip connections, arbitrary op order can lead to very bad performances
- Inception: parallel ops give even more speedup opportunity

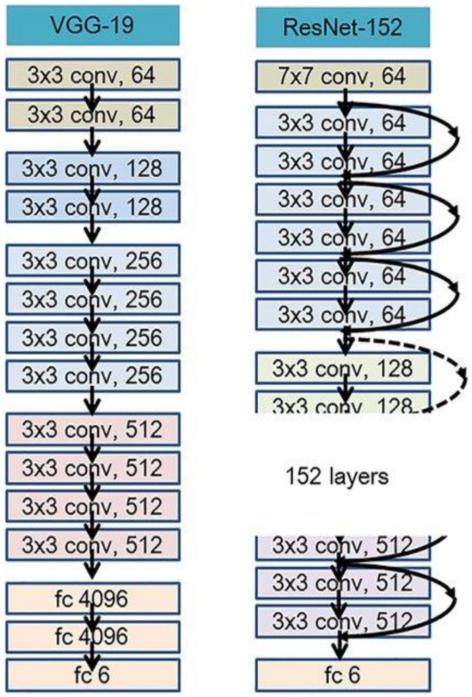


Figure: researchgate.net

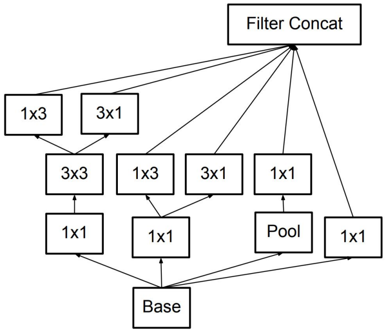
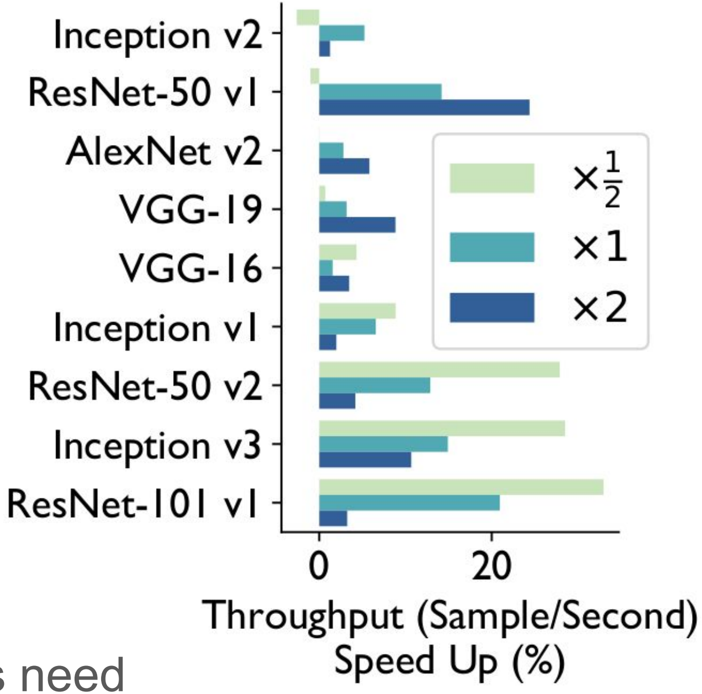


Figure: Inception v3

Experiments: variable batch size at inference time

- Bigger batches require longer computational time
- Network transfer time remains the same, but there is more room for overlap (VGG-19)
- When computation becomes predominant, speedup is less pronounced (ResNet)

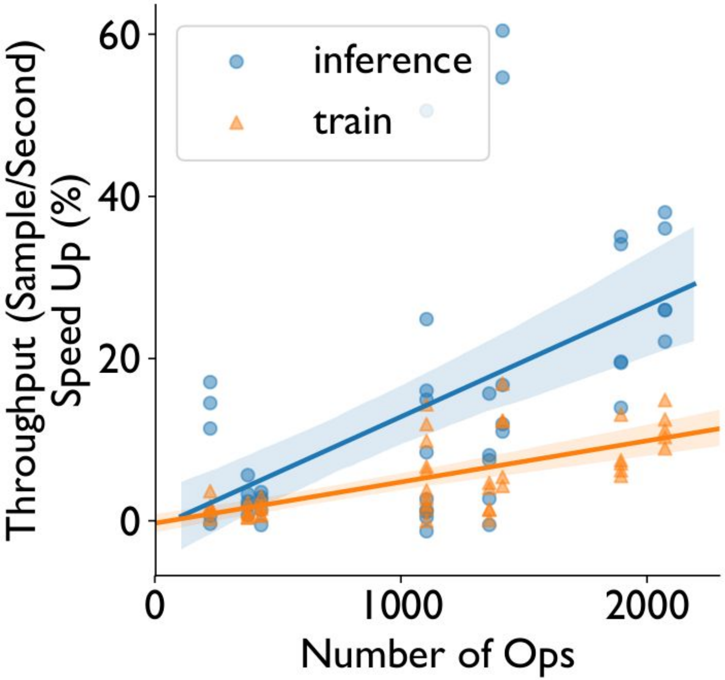


- **Discussion:** At inference time, network transfers need to happen only once, is it so important to optimize them?

Experiment: Speedup vs. DAG size

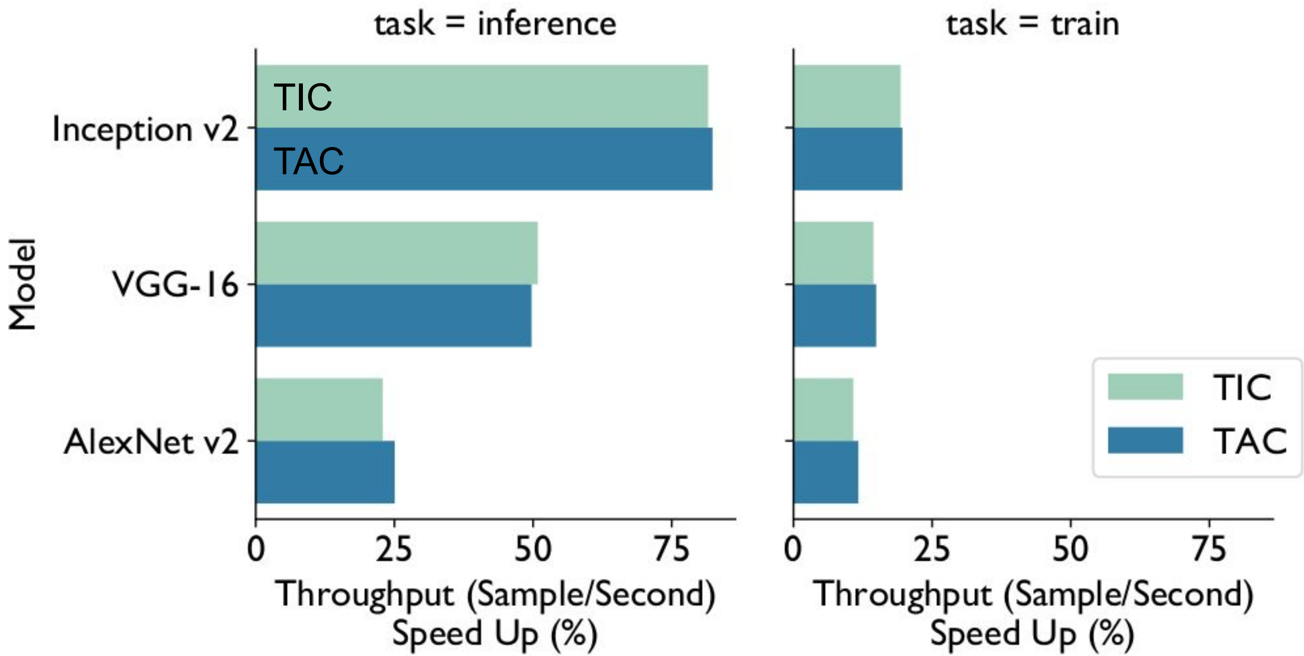
- The bigger the DAG, the greater the optimization opportunity

- **Discussion:** DAG size alone is not very informative, one could track:
    - Longest path
    - Avg/max number of direct dependencies
    - Avg number of parallel operations



Experiments: time-awareness

- TAC is only slightly better than TIC
- DAG structure alone is enough



Conclusions

- Being aware of computation/communication overlap when determining the execution order of a DAG allows optimizing for resource utilization
- Different architectures offer different opportunities for optimization
- Considering the DAG structure alone is enough,
  considering op time is slightly better
- Future work:
  - Storage/memory access
  - Network congestion
    (workers communicating at the same time could exhaust bandwidth)
  - Optimization in an AllReduce scenario

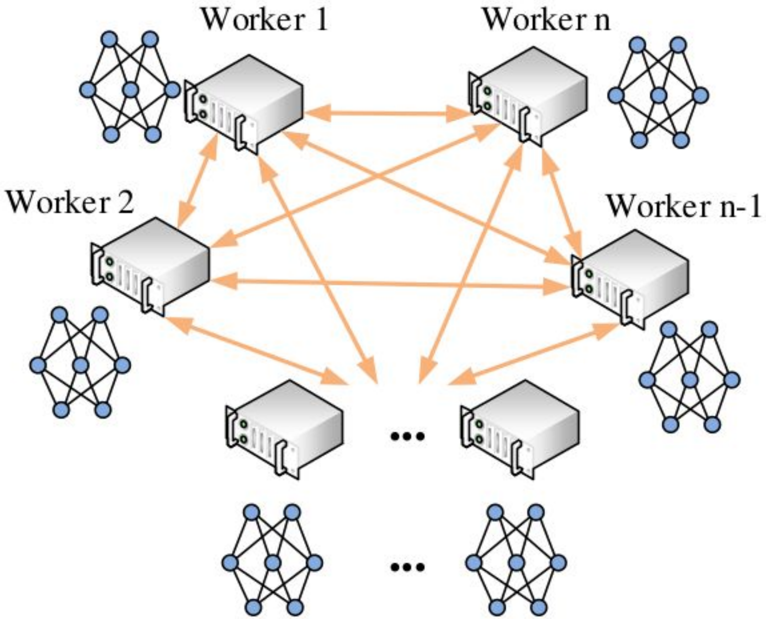
CARAMEL: Accelerating Decentralized Distributed Deep Learning with Computation Scheduling

Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, Brighten Godfrey, Roy H. Campbell

2020 - *preprint?*

Background

- Distributed SGD with AllReduce architecture

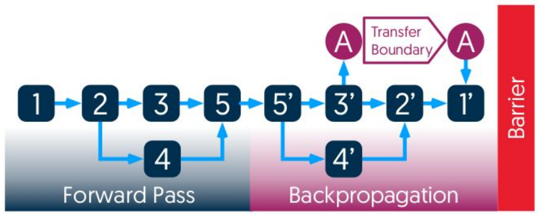


Observed problem

- Standard DL frameworks use the blocking variant of AllReduce
- Computation DAG, not optimized for efficient network communication

Random order of param activations can result in bad schedules

- Synchronization is parallelized with backward pass only
- Uneven network load between large and small parameter transfer times



(a) Example DAG

Proposed solution

- **Goal**: increase GPU utilization
- (Heuristic) scheduling to maximize overlap between communication and computation
  - Increase the time a parameter is available for transfering (*transfer window*)
- (Heuristic) network optimization to smooth the communication load
  - Smart *parameter batching*
  - Faster transfer of large parameters via *adaptive splitting* and pipelining

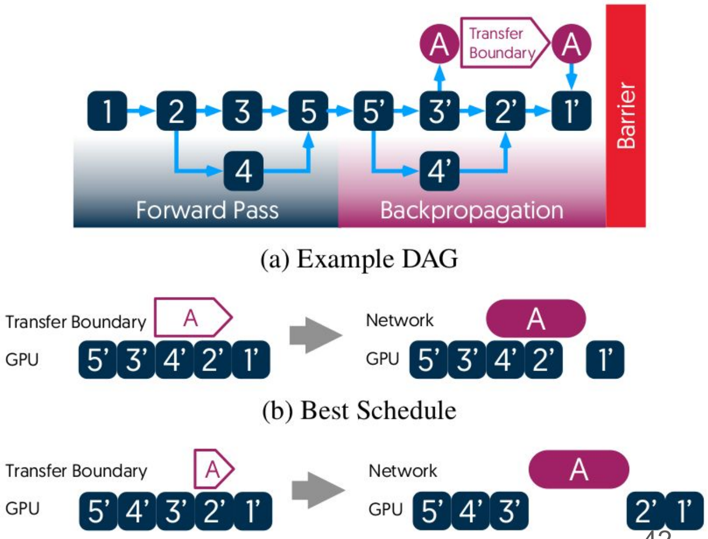
Definitions

- N Network communication time
- C Computation time
- T Total iteration time

Transfer boundary: time in which aggregation of a parameter is feasible

Start: end of param. update
End: comp. op that reads param.

- Comm/comp ratio $\rho = \frac{N}{C}$
- Overlap coefficient $\alpha = \frac{(N+C)-T}{\min(N,C)}$
- GPU utilization $U = \frac{C}{T}$



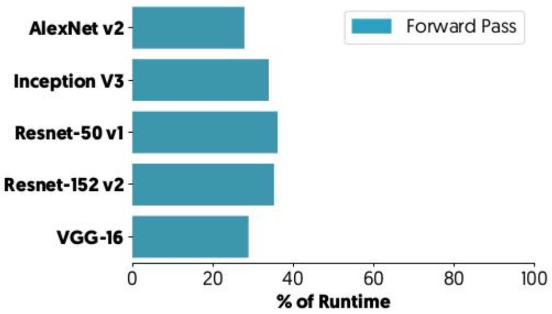
(a) Example DAG

(b) Best Schedule

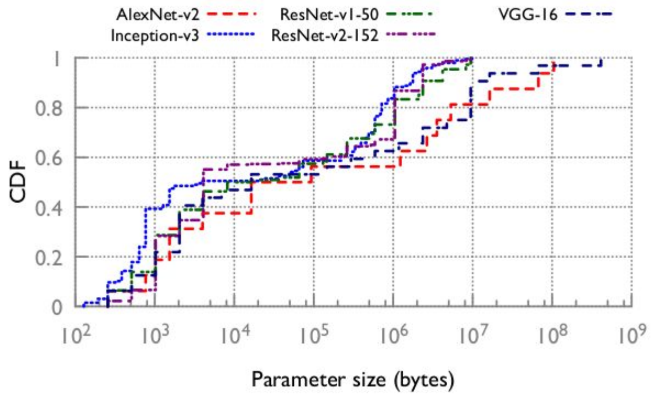
(c) Worst Schedule

Optimization opportunities

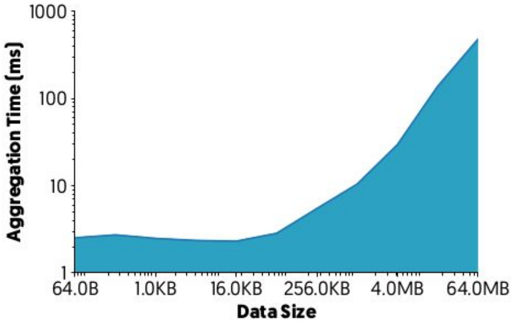
- Forward pass not exploited for parallelization
- Many small parameters incur in significant network overhead



(a) Percentage of Forward Pass in common DNN models



(b) Parameter Size distribution in 5 DNN models



(c) End-to-end transfer time within TensorFlow at different data sizes

Caramel Design

1. Dataflow DAG optimizer
    - **Goal**: maximize α
    - **Strategy**: prioritize computation so that transf. boundary starts earlier
    - **Heuristic**:
        1. Sort params by increasing cost of comp. ops they depend on
        2. Enforce best order in the DAG by introducing dependencies (ensure only one possible order of execution)
    - **Outcome**: earlier start boundaries with reduced variance

Caramel Design

2. Parameter batching
    - **Goal**: reduce ρ
    - **Strategy**: batch small parameters for optimal network communication
    - **Heuristic**:
        i. Fit a linear regression model to predict transfer times
        ii. Estimate threshold for batching small params
        iii. Either queue param for transfer or add to active batch for later transfer

$$\text{threshold} := \min_{d} \frac{f(2d)}{2f(d)} > 0.8$$

Caramel Design

3. Model-aware network transfer scheduler
   - **Goal**: increase α
   - **Strategy**: schedule transfers to either bwd or fwd pass
   - **Heuristic**: greedy bin-packing algorithm (over transfer time and data size)
     1. Sort batches by descending size
     2. Assign to bwd or fwd pass
     3. Unassigned batches are assigned to $\min_C(\mathrm{fwd}(B_i), \mathrm{bwd}(B_i))$

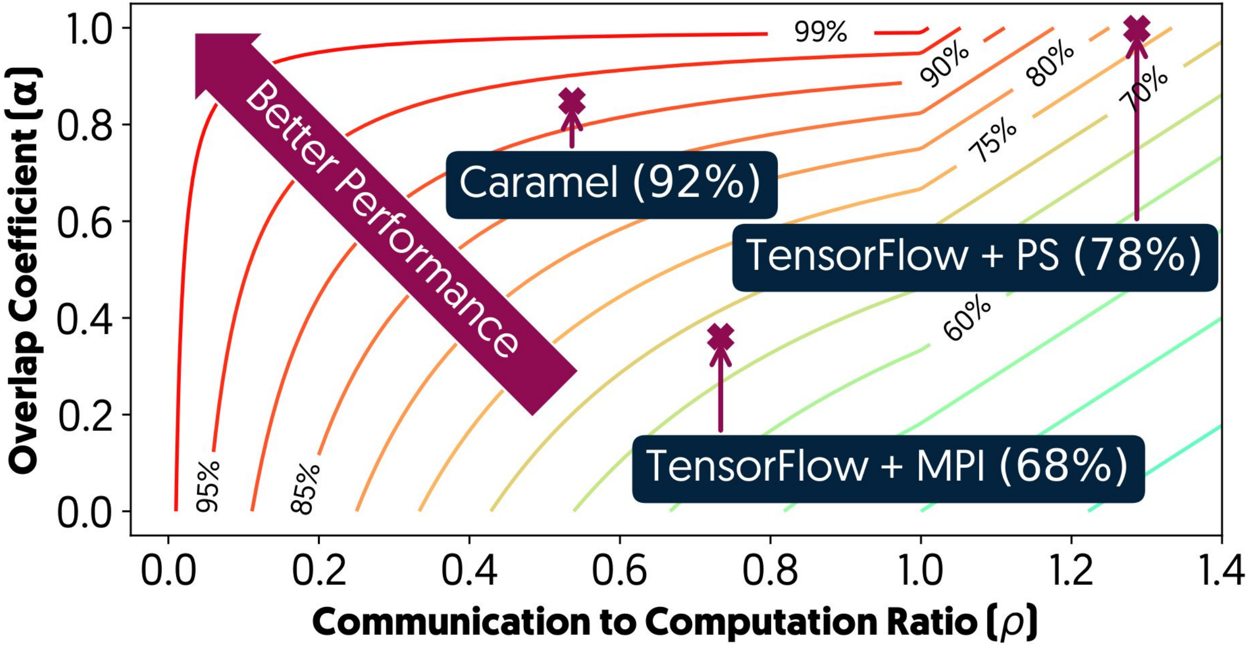
Caramel Design

4. Adaptive depth enforcer
   - **Goal**: reduce ρ
   - **Strategy**: choose data chunk sizes (*depth*) adaptively
   - **Heuristic**:
     1. Depth chosen from 1 to 8
     2. Determined based on batching threshold (as in parameter batching)

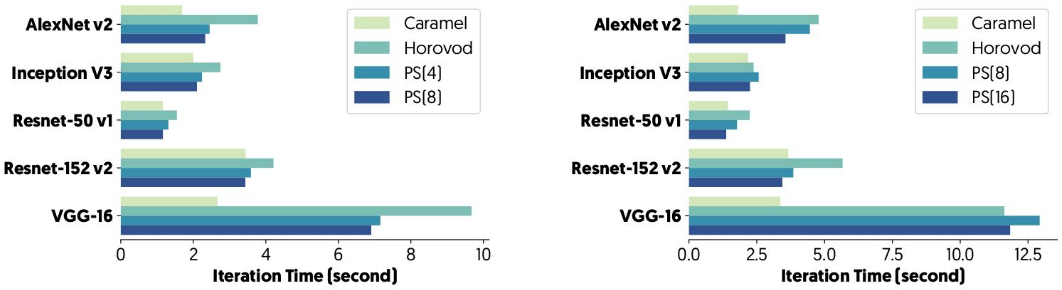
$$\text{threshold} := \min_{d} \frac{f(2d)}{2f(d)} > 0.8$$

Evaluation

- Performance for 8/16 workers over Azure cloud (10Gbps).
- Comparison with horovod and parameter server (PS).

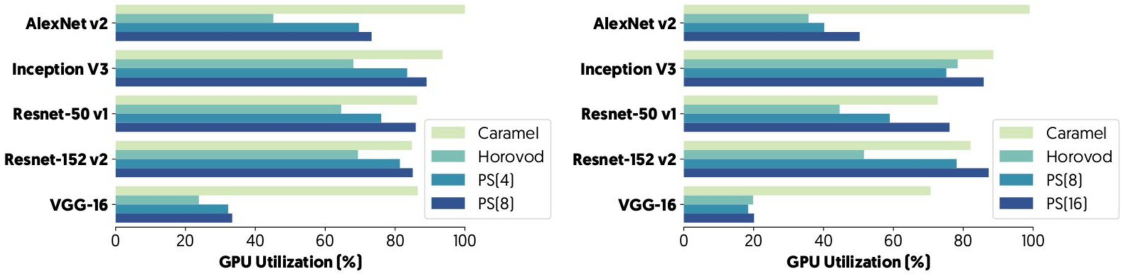


CARAMEL vs Horovod and PS



(a) 8-Worker (b) 16-Worker

Figure 8. Comparison of Iteration Time in Caramel with PS and Horovod. Lower is better.

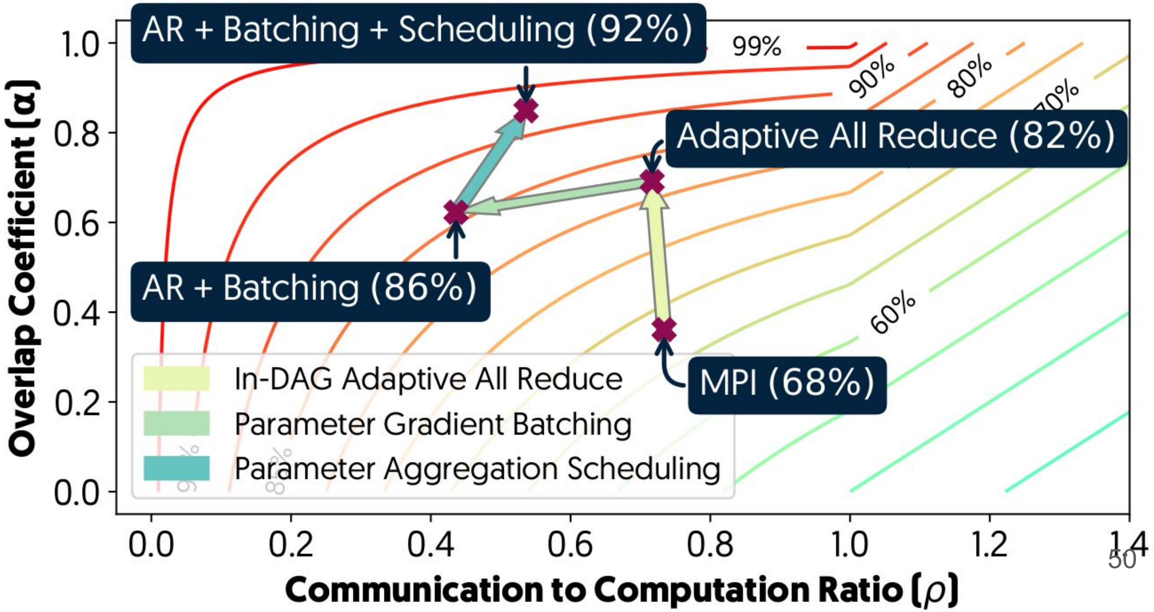


(a) 8-Worker (b) 16-Worker

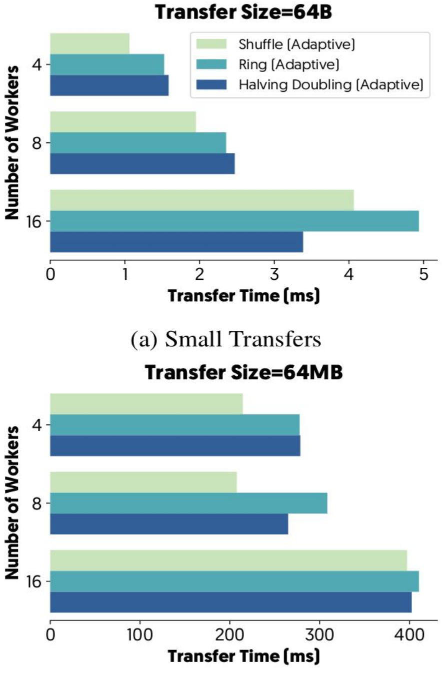
Figure 9. Comparison of GPU Utilization in Caramel with PS and Horovod. Higher is better.

Ablation study

- Adaptive All Reduce: improve overlap & communication cost.
- Batching: reduce communication overhead
- Transfer boundaries: improve overlap



Choice of adaptive decentralized schemes



(a) Small Transfers

(b) Large Transfers