Network-accelerated Distributed Machine Learning for Multi-Tenant Settings

SoCC ’20

DML GROUP MEETING
12.31

OUTLINE

- Introduction
- Central ideas
- Evaluation
- Conclusion

Introduction

- 主流DML算法（例如SGD、LDA）是计算、通信密集的
- DML算法应用在共享集群中与其他任务并行（例如Hadoop、Spark）
- 竞争激烈的集群会导致DML算法遭遇计算以及通信的瓶颈
- Stragglers的出现：

  1）同步算法的每轮迭代时间增加

  2）异步算法的迭代收敛时间增加
- 根本原因：过于简单地看待网络状况

  1) AllReduce：所有worker之间具有固定带宽

  2) PS：worker之间带宽未知（黑盒）
- 系统无法细粒度地协调每次迭代的计算和通信

Introduction

- 问题：作业可能在多租户的集群中面临不可预见的网络和计算瓶颈
- 解决方案：提出一个竞争感知的 DML 通信库 MLfabric
- 设计思想

  1) worker将传送更新的任务移交给 MLfabric

  2) MLfabric 需要估计worker和/或parameter server之间端到端可用带宽

  3) MLfabric根据可用带宽制定合理的通信策略

  4) MLfabric制定合理的备份策略提供一定的容错性、鲁棒性

Central ideas

- MLfabric设计的核心思想：in-network control
- MLfabric设计的三个核心模块：

  1）控制更新延迟（Controlling update delays）

  2）动态汇聚或丢弃更新（Dynamically aggregating or dropping updates）

  3）复制更新以提供容错性（Dynamically aggregating or dropping updates）

Controlling update delays

$$u_t^j = -\eta \frac{\partial}{\partial \mathbf{w}} L(D^j, \mathbf{w}_{t-\tau}) + \lambda(\mathbf{w}_{t-\tau}) \quad (1)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + u_t^j + \gamma\{\mathbf{w}_t - \mathbf{w}_{t-1}\} \quad (2)$$

- Learning rate：$\eta = \frac{C}{\sqrt{\tau_{max} t}}$
- 问题：如果delay τ_{max} 过大，那么learning rate会减少，迭代次数将增加
- 将learning rate设置为delay的函数，如果delay符合 $\tau \in \bar{\text{U}}\text{niform}[0, 2\bar{\tau}]$，那么可以证明：

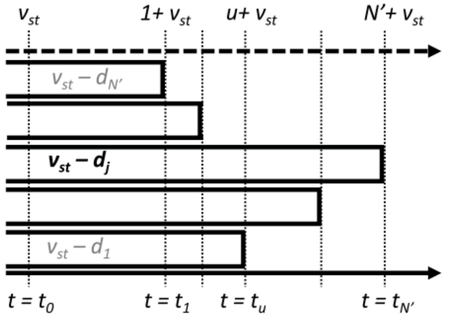
$$E[L(\mathbf{w}_t)] - L(\mathbf{w}^*) \leq O\left(\frac{\bar{\tau}\sqrt{t}}{t}\right)$$

- 如果delay符合 $\tau \in \text{Uniform}[\bar{\tau} - \epsilon, \bar{\tau} + \epsilon]$，作者证明：

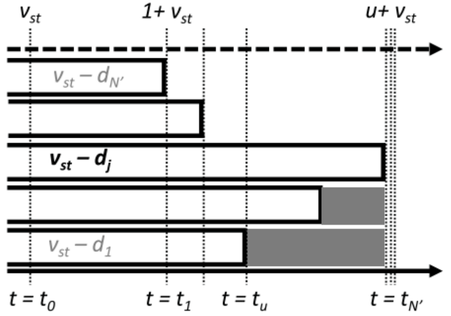
$$E[L(\mathbf{w}_t)] - L(\mathbf{w}^*) \leq O\left(\frac{\epsilon\sqrt{t + \bar{\tau} - \epsilon}}{t}\right)$$

- 换句话说，如果我们能控制delay的方差，那么我们可以得到常数因子的加速比

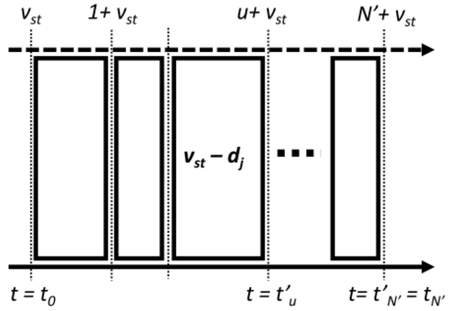
Controlling update delays



(a) Delay bound is violated for one update



(b) Buffering updates to bound delay at server



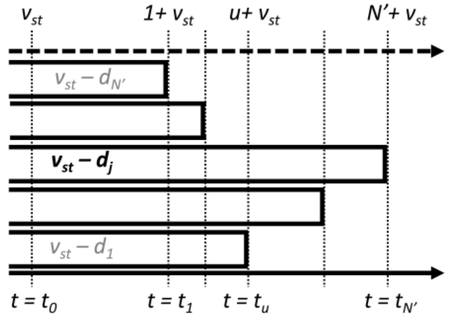
(c) Ordering gradient updates over the network

- 图(a)：当前有 N' 个更新，所有更新一起发送，争用带宽
- 问题：对于其中一个更新 $v_{st}-d_j$，其中 $d_j=\tau_{max}-u$，server端观察到它的delay为：

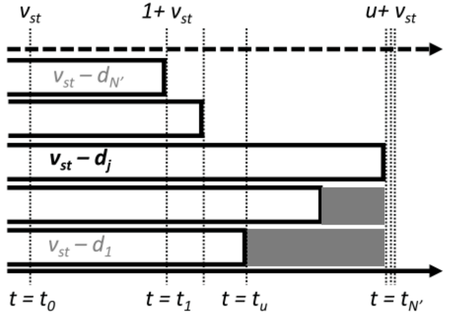
$$v_{st}+\bar{N'}-(v_{st}-\tau_{max}+u)=\tau_{max}+(N'-u)$$

- 解决方案：丢弃更新 j，会导致一定程度上的信息丢失

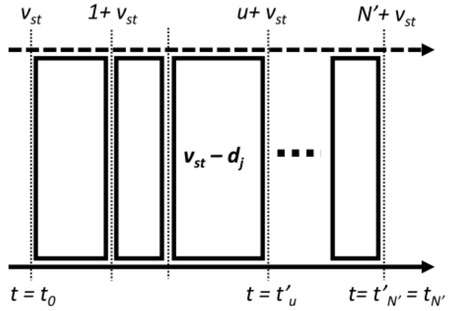
Controlling update delays



(a) Delay bound is violated for one update



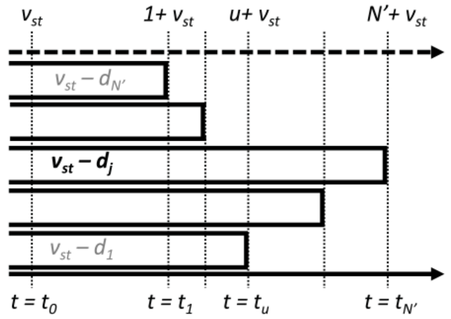
(b) Buffering updates to bound delay at server



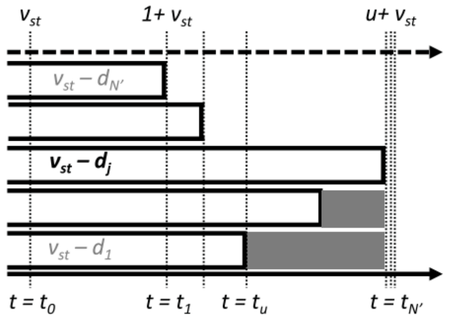
(c) Ordering gradient updates over the network

- 图(b)：server端在$t = t_u$时对之后的更新做缓存
- 问题：
- 尽管做了缓存后，update j 的delay正好为τ_{max}，但是在 $t = t_u$ 时缓存的update的delay都会加1，并且在 $[t_u, t_{N'})$ 时间内，worker不能获得最新的参数

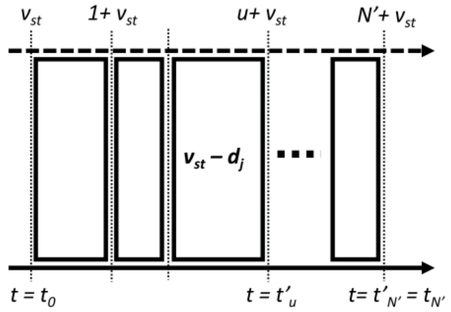
Controlling update delays



(a) Delay bound is violated for one update



(b) Buffering updates to bound delay at server



(c) Ordering gradient updates over the network

- 图(c)：in-network control
- 强制进行network time sharing，即不同的update由网络在精心选择的非重叠时间内进行传输
- 合理地安排update的传输顺序可以在不做缓存的情况下满足delay要求

Update ordering

- 给定一组update，采用network time sharing的策略
- 如何获得一组Update ordering?
- 要求：
- 1）最小化到服务器的平均更新传输时间以确保快速更新
- 2）满足延迟边界的约束
- 解决思路：
- 1）首先尝试最小化平均传输时间
- 2）修复任何违反的延迟界限的update

Update ordering

- Average completion time
- 在每次迭代中，给定当前可用带宽
- 1）计算每个更新的传输完成时间
- 2）选择完成时间最短的传输，并在其路径上预留容量。预留量等于瓶颈带宽，预留时长等于传输完成时间

Update ordering

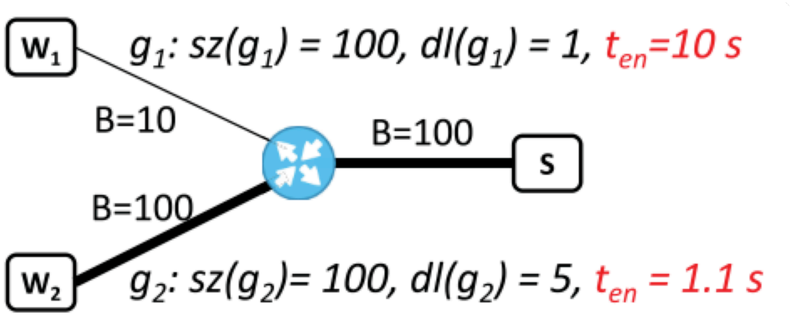
- Bounding delays
- Shortest-transfer-first-ordering会增加delay
- 为了确保满足延迟限制，引入了deadline：

$$dl(g) := v(g) + \tau_{max} - v_{init}$$

- 调整算法：
- 1）在迭代i中，如果存在未调度的更新 g ，使得$dl(g) = i$，那么我们在该迭代中选择g，并为传输update g 预留带宽
- 2）否则我们选择传输时间最短的update

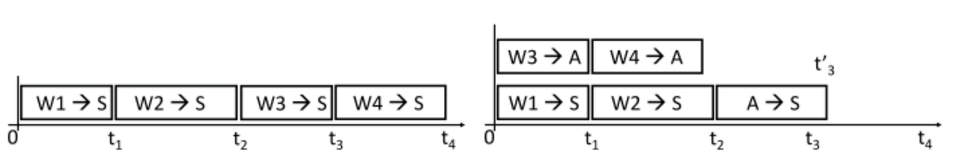
Update ordering

- Dropping delayed updates
- 仅考虑最短传输优先和deadline不足以确定“良好”的排序
- 增加了deadline后可能会不必要地导致网络或服务器资源闲置（右图例子）
- 解决方案：
- 丢弃update $g1$，update $g2$ 立即安排传输

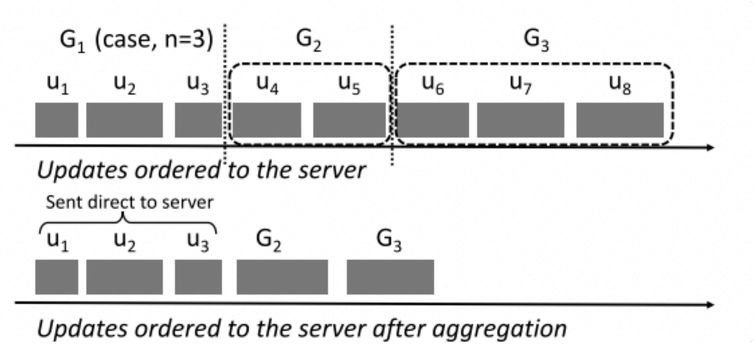


Dynamically aggregating or dropping updates

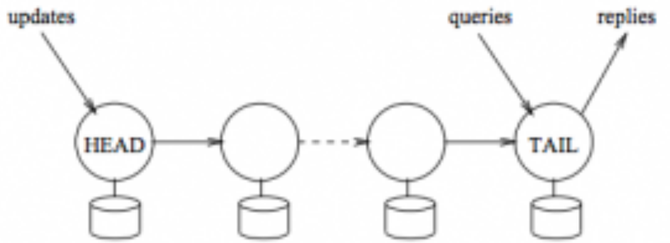
- In-network control可以使得有序更新能够在被到达服务器之前进一步聚集
- 网络负载降低，模型更新更快
- 具体实现：
- 1）对有序更新进行分组
- 2）每个组直接传输到服务器，或者首先传输到聚合器然后再到服务器
- 分组约束：
- 聚合第 i 组中的update不应晚于所有先前 $i-1$ 组的update聚合传输到服务器的时间



(a) Updates forwarded directly to server (b) Update forwarded to server through aggregator



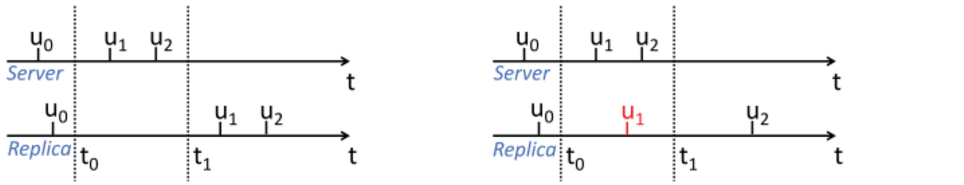
Replicating updates for fault tolerance



- 现有容错性策略：chain replication
- 问题：
- chain replication在每 n 次迭代中复制一次来减少数据开销
- 如果update是稀疏的，不频繁的复制只会分摊服务器数据开销
- 解决方案：采用基于worker的复制策略，每个update的副本直接转发到replica
- 如果没有in-network control，这种基于worker的复制策略可能导致服务器-副本模型出现分歧，这使得从副本恢复变得缓慢（出现update reordering）
- 采用in-network control之后：
- （1）它们以相同的顺序应用于服务器和副本，
- （2）我们可以实现有界性 （$||\mathbf{w}_s - \mathbf{w}_r||_2$ 小于 Div_{max}）

$$\begin{aligned} \mathbf{w}_2^s &= \mathbf{w}_0 + (\gamma h^0 + u_1) + \{\gamma(\gamma h^0 + u_1) + u_2\} \\ \mathbf{w}_2^r &= \mathbf{w}_0 + (\gamma h^0 + u_2) + \{\gamma(\gamma h^0 + u_2) + u_1\} \end{aligned}$$

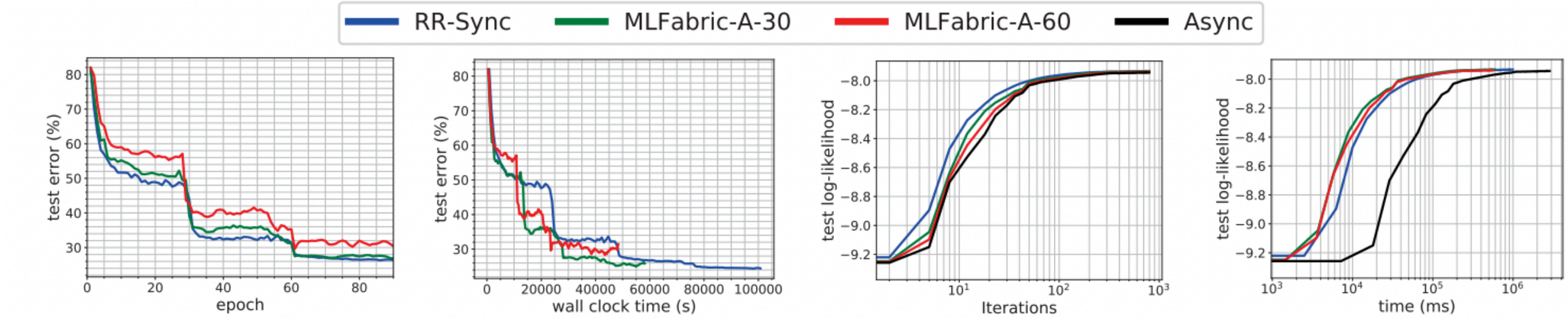
$$\begin{aligned} \mathbf{w}_2^s - \mathbf{w}_2^r &= \gamma(u_1 - u_2) \\ ||\mathbf{w}_2^s - \mathbf{w}_2^r||_2 &= \gamma||u_1 - u_2||_2 \end{aligned}$$



(a) Replica lags the server by two updates

(b) u_1 scheduled ahead at replica to satisfy divergence bound

Evaluation



(a) Deeplearning - #iter vs error rate (b) Deeplearning - time vs error rate (c) LDA - #iter vs log-likelihood (d) LDA - time vs log-likelihood

THANKS!