Powering Multi-Task Federated Learning with Competitive GPU Resource Sharing

Yongbo Yu, Fuxun Yu, Zirui Xu
George Mason University
{yyu25,fyu2,zxu21}@gmu.edu

Di Wang, Mingjia Zhang
Microsoft
{wangdi,minjiaz}@microsoft.com

Ang Li
Duke University
ang.li630@duke.edu

Shawn Bray, Chenchen Liu
University of Maryland
{shawnb2,ccliu}@umbc.edu

Xiang Chen
George Mason University
xchen26@gmu.edu

ABSTRACT

Federated learning (FL) nowadays involves heterogeneous compound learning tasks as cognitive applications' complexity increases. For example, a self-driving system hosts multiple tasks simultaneously (e.g., detection, classification, segmentation, etc.) and expects FL to retain life-long intelligence involvement. However, our analysis demonstrates that, when deploying compound FL models for multiple training tasks on a GPU, certain issues arise: (1) As different tasks' skewed data distributions and corresponding models cause highly imbalanced learning workloads, current GPU scheduling methods lack effective resource allocations; (2) Therefore, existing FL schemes, only focusing on heterogeneous data distribution but runtime computing, cannot practically achieve optimally synchronized federation. To address these issues, we propose a *full-stack* FL optimization scheme to address both *intra-device* GPU scheduling and *inter-device* FL coordination for multi-task training. Specifically, our works illustrate two key insights in this research domain: (1) Competitive resource sharing is beneficial for parallel model executions, and the proposed concept of "virtual resource" could effectively characterize and guide the practical per-task resource utilization and allocation. (2) FL could be further improved by taking architectural level coordination into consideration. Our experiments demonstrate that the FL throughput could be significantly escalated.

CCS CONCEPTS

- **Computing methodologies** → **Parallel algorithms**.

KEYWORDS

Federated Learning; Multi-Task Learning; GPU Resource Allocation

ACM Reference Format:
Yongbo Yu, et al.. 2022. Powering Multi-Task Federated Learning with Competitive GPU Resource Sharing . In *Companion Proceedings of the Web Conference 2022 (WWW '22 Companion), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3487553.3524859

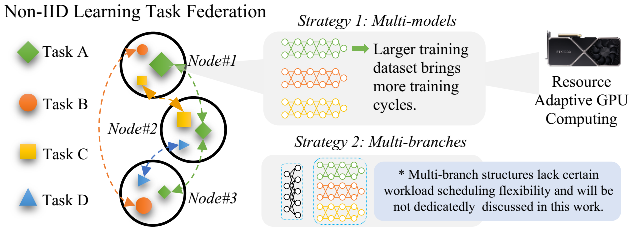

Non-IID Learning Task Federation
Task A
Task B
Task C
Task D
Node#1
Node#2
Node#3
Strategy 1: Multi-models
Larger training dataset brings more training cycles.
Resource Adaptive GPU Computing
Strategy 2: Multi-branches
* Multi-branch structures lack certain workload scheduling flexibility and will be not dedicatedly discussed in this work.

Figure 1: Federated Learning with Multi-Tasks

1 INTRODUCTION

Federated learning (FL) is a distributed training methodology allowing multiple computing devices to jointly train cognitive tasks without sharing private data [4]. As the deployment of intelligent applications is widely spreading in ever complex scenarios, FL becomes the essential technique for gathering individual computing resources and maintaining intelligence involvement timely. Meanwhile, the complexity of these applications is also getting far more complicated. As shown in Fig. 1: each device (e.g., an self-driving system) may hosts different FL tasks (e.g., detection, classification, etc.) [1]; each task involves a specific deep neural network model (DNN) or partial structures; and highly-biased data volume are distributed per task to train. Although some heterogeneous FL methods have resolved the algorithmic challenges for federation convergence regarding heterogeneous model coordination and parameter matching across unique devices [1], the practical imbalanced training workload per task may cause potential issues inside the computing architecture in each device. Taking GPU as the major FL computing unit, there are already some existing parallelism methods (e.g., CUDA MPS [7], MIG [2]) to coordinate the resource allocation for concurrent model training. However, facing the considerable workload imbalance and dynamics in multi-task FL scenarios, these methods lack sufficient management granularity and generally suffer from considerable parallelism contention and resource under-utilization.

Therefore, different from conventional FL works, this work dives into the GPU architectural level with the particular scheduling issue of multi-task FL deployment and re-examine the FL coordination problem. Specifically, we are focusing on two major challenges: (1) *How to demystify the practical resource utilization and parallelism contention for multi-task model training, and therefore guide the GPU scheduling?* (2) *How to re-invent the FL scheme by taking the architecture-level GPU scheduling into consideration?*

By tackling these challenges, we propose a *full-stack* multi-task FL optimization scheme, which addresses both *intra-device* GPU

WWW '22 Companion, April 25–29, 2022, Virtual Event, Lyon, France.

ACM ISBN 978-1-4503-9130-6/22/04…$15.00
https://doi.org/10.1145/3487553.3524859

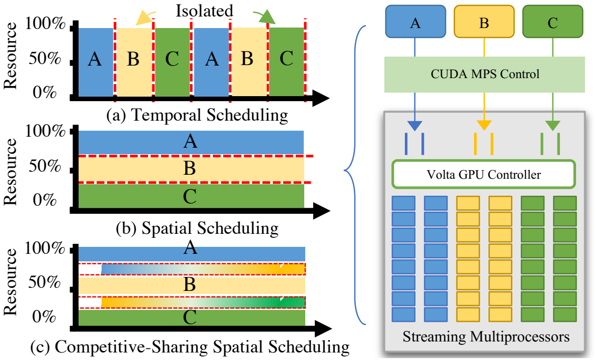


Figure 2: Temporal and Spatial GPU Scheduling Schemes

scheduling with a competitive resource sharing scheme; and *inter-device* FL coordination with realistic GPU runtime synchronization. Experiments show that we could greatly enhance the GPU resource utilization, and in turn improve the overall intra-device training throughput by 2.16×~2.38× and inter-device FL coordination throughput by 2.53×~2.80× in complex multi-task FL scenarios.

2 PRELIMINARY

To achieve the optimal performance of multi-task FL, we need to coordinate two computational aspects: **(a)** *Intra-device GPU scheduling and utilization*. Due to heterogeneous models parallel, this requires a resource allocation method that allocates the optimal resource to achieve optimal multi-task deployment on GPU and achieve maximum instantaneous throughput. **(b)** *Inter-device FL coordination*, which expects all devices in the cluster to perform synchronized parameter fusion, and each device could fully utilize the global synchronization cycle to train as much data as possible.

GPU Resource Allocation and Scheduling: As deploying the aforementioned multi-task FL into individual devices, the major computing unit – GPU is facing a complex scheduling issue to host multiple training models. Currently, there are two major resource allocation and scheduling schemes: *Temporal scheduling* in Fig. 2 (a) isolates GPU resources into sequential time slices for individual tasks (e.g., round-robin [8]). Each task takes the whole GPU resource in its time slice without interfering with others [3]. *Spatial scheduling* in Fig. 2 (b) processes multiple GPU tasks in parallel by assigning a sub-set of streaming multiprocessors (SMs) to individual tasks as independent processing threads [5]. Latest GPU scheduling technologies are still within these schemes, such as NVIDIA Multi-Instance GPU (MIG) [2], Multi-Process Service (MPS) [7].

Although spatial scheduling is more preferred by its parallelism, there occurs ***competitive resource sharing*** between tasks. It is used to describe the complex interactions between concurrent tasks within a single GPU, such as the resource contention between similar computing operators from different training models, or the resource under-utilization caused by insufficient resource allocation. However, it is still a newly emerging design consideration. Therefore, how to demystify it to achieve optimal resource allocation and bring it into a large-scale FL is our major research motivation.

Multi-Task FL involves many factors: J devices in a federation cluster have I tasks, and the i^{th} task corresponds to a specific DNN model structure, and a data set of $D_{i,j}$ on j^{th} device. Each model locally trains its weight parameters of $W_{i,j}$ to pursuit the minimum accuracy loss across different tasks, and fuse individual task's model with each other (based on their assignment training data set's portion across all devices) in every global synchronization cycle [6]:

$$\begin{cases} Local\ Training: \min\limits_{\{W_{0,j}\}_{j=0}^{J}} \sum_{j=0}^{J} Loss(D_{0,j}, W_{0,j}), \\ Fusing: W_{i,j}^{\{k^{th}\ cycle\}} = \sum_{j=0}^{J} \frac{|D_{0,j}|}{\sum_{k=0}^{J} |D_{0,k}|} W_{i,j}^{\{k-1^{th}\ cycle\}}. \end{cases} \tag{1}$$

In FL, the *local computing workload* – $O_{i,j}$ is specified by the task model size $M_{i,j}$ and the assigned batch size $B_{i,j}$. Through adjusting $O_{i,j}$ and *corresponding resource allocations* in a GPU, we are expecting to achieve the maximum throughput with parallel multi-task deployment per cycle and improve the overall FL speed.

3 INTRA-DEVICE GPU SCHEDULING WITH COMPETITIVE RESOURCE SHARING

In this section, we analyze the competitive resource sharing mechanism and identify the particular computing issue, and thus propose a multi-task dedicated GPU scheduling method.

3.1 Competitive Resource Sharing Analysis

Baseline Example of Spatial Scheduling: Fig. 2 (b) indicates a fully isolated spatial resource allocation approach, which is achieved by a very recent GPU scheduling technique (i.e., MPS). It assigns an exclusive set of hardware SMs per model (represented as a series of operators) to perform parallel computing, as shown in Fig. 3 ①. However, such an exclusive resource assignment lacks certain flexibility as some small operators in the model will not fully occupy the assigned resources, thus causing certain under-utilization.

Spatial Resource Sharing: Instead of exclusive resource allocation, we could enhance the GPU utilization by enabling operators from parallel models with complementary sizes to share certain resources, as shown in Fig. 3 ②. In runtime GPU scheduling, this is be done by assigning a resource budget to individual models. Thus, the overall resource assignment could exceed 100% due to overlapped assignment of physical SMs. However, by assigning the resource budget, the physical resource allocation dynamics is not well studied yet, as models may compete for shared resources.

Excessive Contention Overhead: As we increase the shared resources by making more overlapped resources assignments in Fig. 3 ③, task models compete for resources more fiercely, leading to resource competition and considerable contention overhead. This is the major research focus of our analysis.

Extreme Contention Kills Parallelism: The resource contention issue is not only causing overhead. Here, when large operators exist, it is hard to achieve complementary resource assignment and thus the scheduling mechanism is pushed back to temporal scheduling as shown in Fig. 3. Therefore, in addition to complementary operators, special attention is also required for such cases, which further increases the analysis complexity of competitive resource sharing.

3.2 Competitive Resource Sharing Coordination with *Virtual Resource*

Motivated by the analysis above, we propose a new prospect of resource allocation for multi-task GPU computing — "***virtual resource***" management to guide optimal runtime configuration.

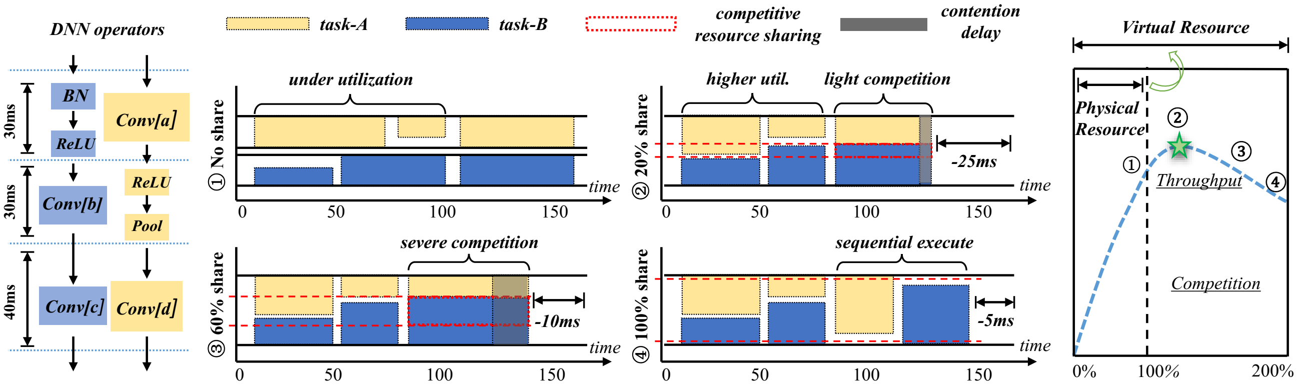


Figure 3: Competitive GPU Resource Sharing with Multi-Task DNN Training

Definition of Virtual Resource: As mentioned above, when scheduling the runtime resource assignment, a resource budget is applied to each task model. However, as the overall budget exceeds 100% of physical resources, the intrinsic allocation mechanism is not well studied. Thus, we extend such resource budget concept into ***virtual resource*** to illustrate the composition degree of competition.

As shown in Fig. 3 (right side), the virtual resource is a number between *(0%~100%)* $\times (I\ of\ tasks)$. For example, when deploying two task models, the virtual resource could reach as high as 200%. Finding an optimal virtual resource could manage an appropriate parallel resource competition and sharing, achieving optimal throughput.

Virtual Resource Specification for Optimal Multi-task Resource Allocation: To manage the competitive resource sharing of complex multi-task models, it needs to clarify the actual competitive resource sharing performance based on virtual resource. Thus, we propose a machine learning approach to estimate the GPU throughput P based on a series of task models' virtual resource R assignment and corresponding task models' workload O. Moreover, it is also necessary to identify the computing factors F_i [10] of different task models M_i, such as computing characteristics (FLOPS) and memory access. Then we multiply these factors with the batch size B_i to represent the i^{th} task model's workload O_i. Please note that, a larger batch size could make the task model have larger operators or overall larger workload, so the task model could occupy more resource in parallel deployment. We take the workload (represented by $F_i * B_i$) and the virtual resource allocation (R_i for M_i) as the machine learning's input. As each task model competes with other models when sharing the resource, we should consider the common impact between task models in parallel to estimate task models' throughput P. Therefore, we combine $F_i * B_i$ and R_i between different task models as the final inputs. So we construct a multi-input and multi-output prediction machine learning model, and formulate it as following:

$$P_1, .., P_i = Pre((F_1 * B_1, R_1), (F_2 * B_2, R_2), ..., (F_i * B_i, R_i)). \quad (2)$$

We profile extensive multi-task training cases to fit the parameters of the prediction model. We record the corresponding throughput when changing the virtual resource allocation R_1 to R_i and batch size B_1 to B_i respectively and use this recording as supervised information to train the prediction model [9]. Thus, the actual composition of competition and sharing could be effectively illustrated in Fig. 3.

By analysing competitive resource sharing, we demystify the practical resource utilization and parallelism contention for multi-task model training with virtual resource. And through establishing the relationship between virtual resource and throughput, we achieve the optimal multi-task intra-device GPU scheduling.

4 INTER-DEVICE MULTI-TASK FEDERATED LEARNING COORDINATION

Based on the intra-device virtual resource management, we further bring it into the inter-device FL cluster and rethink the FL coordination from a GPU scheduling perspective.

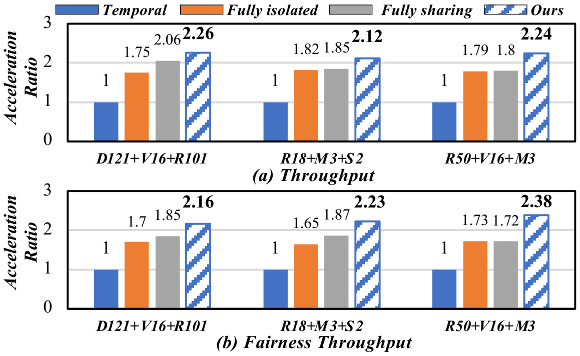
Coordination Design Motivation: Our goals are to make each device could be fully utilized during each synchronization cycle when multi-task models parallel in each device, and meanwhile maximize the overall GPU throughput to accelerate the overall FL training speed.

We achieve the first goal using the Eq. 2 through adjusting mismatch between the ratio of different tasks' data volume D and the ratio of different tasks' workload O. For the second goal, we adjust the resource allocation according to the workload which is influenced by batch size to obtain the maximum throughput. And the goals can be formulated by the following objectives:

$$\begin{cases} Objective\ 1: \ \min \sum_i \sum_j \frac{|D_i|}{|D_j|} - \frac{O_i}{O_j}, \\ Objective\ 2: \ \max \sum_i P_1, .., P_i. \end{cases} \quad (3)$$

Specifically, for the first objective, we can understand its principle using the relation between D and O. When a task model has larger data volume D, we improve its workload through increasing batch size to make this task model occupy more resource, so that this task model can consume correspondingly more amount of data during each synchronization cycle. Conversely, when a task model has smaller D, we decrease its workload, so that more resources can be occupied by other task models with larger D. Therefore, achieving co-scheduling of the multi-task FL coordination can be transformed into leveraging the workload adjustment and resource allocation to satisfy the above two objectives.

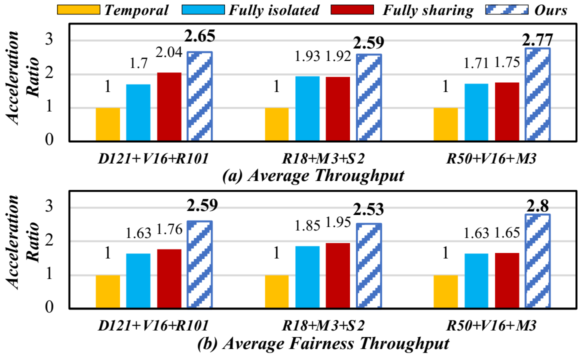
Multi-task FL Coordination: This multi-task coordination problem is hard due to the various resource allocation for different model combination and data imbalance between different tasks, so we adopt a greed optimization method to find the optimal batch size and resource allocation. We first search the optimal workload O of each task, and the O can be represented by the $F * B$. So we can search for the batch size B to optimize the second objective. This objective only has one variable B, and we can solve the B for each



(a) Throughput

(b) Fairness Throughput

Figure 4: Intra-Device Multi-Task GPU Throughput



(a) Average Throughput

(b) Average Fairness Throughput

Figure 5: Inter-Device Average Throughput in a Federated Learning Synchronization Cycle

task in a closed form. Then, we use the optimal batch size as one of the input to search the optimal resource allocation using Eq. 2 to maximize intra-device throughput. Using this two-step greedy scheduling method, we can quickly find the optimal batch size and resource allocation to achieve FL coordination.

Using the proposed method, we achieve inter-device multi-task FL coordination with realistic GPU runtime synchronization while enhancing the intra-device overall throughput.

5 EXPERIMENTAL EVALUATION

Experimental Setup: We construct various multi-task FL scenarios with following DNN models: VGG16 (V16), ResNet18 (R18), ResNet50 (R50), ResNet101 (R101) MobileNet_v3 (M3), ShuffleNet_v2 (S2) DensNet121 (D121). The selection of models has a rich diversity, covering different computational and memory requirements and having distinctive model depths with operator numbers. We evaluate three multi-task settings. D121+V16+R101 is a heavy multi-task training with larger computational workload. R18+M3+S2 is lighter since M3 and S2 are often deployed on mobile. R50+V16+M3 is a hybrid type and the requirements for each task in this combination vary. For simplicity, we evaluate the same batch size (128) for all the combinations. We evaluate the multi-task training on the CIFAR10 dataset and use the NVIDIA Titan V GPU.

Baseline and Metrics: We consider three baseline strategies. *(1) Temporal*: The default blackbox CUDA scheduler; *(2) Fully-Isolated*: Using MPS to allocate dedicated SMs equally to each process; *(3) Fully sharing*: Each task could compete and utilize all SMs (MPS default). We use two metrics to evaluate the performance: ***raw throughput*** T and ***fairness throughput*** T_f[1]. All methods' throughput are normalized to show the relative acceleration ratio.

Overall Speed-up: Our resource allocation method could consistently yield 2.16× to 2.38× speed-up compared to the temporal sequential training baseline across all three settings. Although the Spatial Isolated and Fully Sharing solution also yields a certain speed-up than Temporal baselines, but its acceleration ratio is much less than ours. Meanwhile, our approach achieves higher acceleration on Fairness Throughput. This is because it is easier for small models to get GPU resources by resource allocation. Without a good resource allocation, such as fully sharing, large models will take up resources for a long time, thus affecting the throughput of small models.

Inter-Device Multi-task FL Coordination: We use a FL system with several devices, each devices have three tasks with different model structure and imbalance data volume. We consider the same baseline strategies and assign the same batch size (128) for these baseline methods. We give the the average throughout and fairness throughput in one synchronization cycle of all devices, and the results show in the Fig. 5. From the Fig. 5, we can find that we achieve 2.53× to 2.80× speed-up compared to the baseline methods. More interestingly, we achieve faster acceleration on the inter-device as opposed to the intra-device acceleration. This is because the joint optimization of batch size and resource allocation allows our method to utilize GPU resources during the whole synchronization cycle, but the baseline methods can not fully utilize causing imbalance between workload and data volume.

6 CONCLUSION

In this work, we propose a full-stack multi-task FL optimization scheme, which addresses both intra-device GPU scheduling with a novel competitive resource sharing scheme; and inter-device multi-task FL coordination with realistic GPU runtime synchronization. Experiments show that we could greatly enhance the GPU resource utilization, and improve the overall training throughput.

REFERENCES

[1] Mehdi Salehi Heydar Abad *et al.* 2020. Hierarchical federated learning across heterogeneous cellular networks. In *Proceedings of the ICASSP IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 8866–8870.

[2] Nvidia Inc. 2021. NVIDIA Multi-Instance GPU User Guide. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/

[3] Shinpei Kato *et al.* 2011. TimeGraph: GPU scheduling for real-time multi-tasking environments. In *Proceedings of the USENIX ATC*. 17–30.

[4] Tian Li *et al.* 2020. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine* 3 (2020), 50–60.

[5] Yun Liang *et al.* 2014. Efficient GPU spatial-temporal multitasking. *IEEE Transactions on Parallel and Distributed Systems* 26, 3 (2014), 748–760.

[6] Brendan McMahan *et al.* 2016. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527* (2016).

[7] Nvidia. 2021. Multi-Process Service. https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

[8] Rasmus V Rasmussen*et al.* 2008. Round robin scheduling–a survey. *European Journal of Operational Research* 188, 3 (2008), 617–636.

[9] Xiangyu Ukidave *et al.* 2016. Mystic: Predictive scheduling for gpu based cloud servers using machine learning. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 353–362.

[10] Gingfung Yeung *et al.* 2021. Horus: Interference-aware and prediction-based scheduling in deep learning systems. *IEEE Transactions on Parallel and Distributed Systems* 33, 1 (2021), 88–100.

[1] Raw throughput is the direct summation of all models' training throughput (img/s), while fairness throughput is a normalized throughput metric by each model's standalone throughput when running alone. By such normalization, we treat each model's speedup/slowness with fair importance.