MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

HiDL
High-Performance Deep Learning

HyperSack: Distributed Hyperparameter Optimization for Deep Learning using Resource-Aware Scheduling on Heterogeneous GPU Systems

Presented at MUG '25

Nawras Alnaasan, Bharath Ramesh, Jinghan Yao, Aamir Shafi, Hari Subramoni, and Dhabaleswar K (DK) Panda

Department of Computer Science and Engineering,

The Ohio State University, Columbus, Ohio, USA

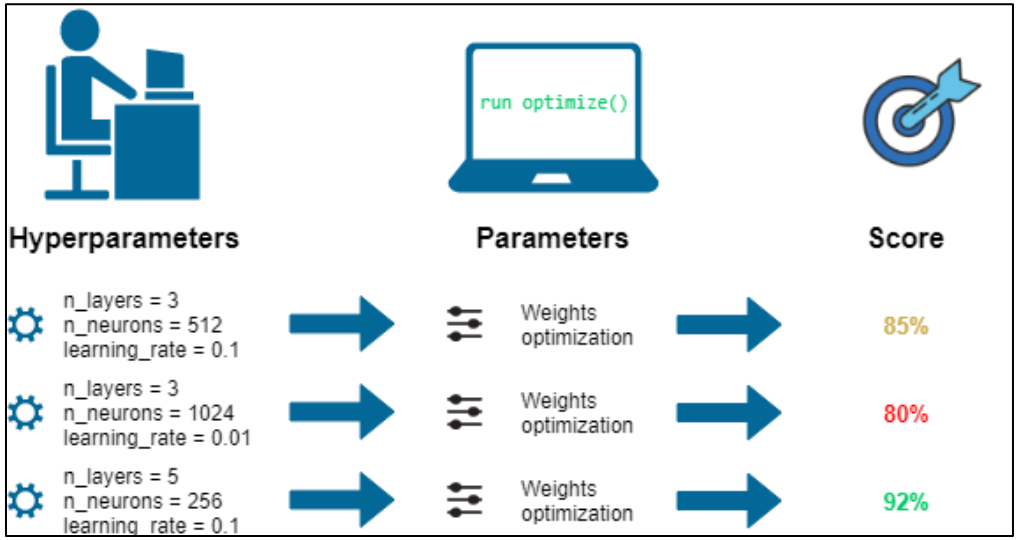
Presentation Outline

- **Introduction and Motivation**
- Problems and Challenges
- Proposed Design and Implementation
- Analysis and Performance Evaluation
- Conclusion

Hyperparameter Optimization (HPO)

- A hyperparameter value is used to control the learning process. It is set prior to the training. By contrast, the values of model parameters are derived via training.
- Manual tuning is a widely used strategy to optimize models where the hyperparameter values are selected based on a trial-and-error approach by an ML/DL expert.
- Automated Hyperparameter Optimization (HPO) strategies systematically navigate search spaces to tune the training configurations.
- However, HPO comes at a significant cost due to multi-dimensional search spaces and complex neural network architectures.

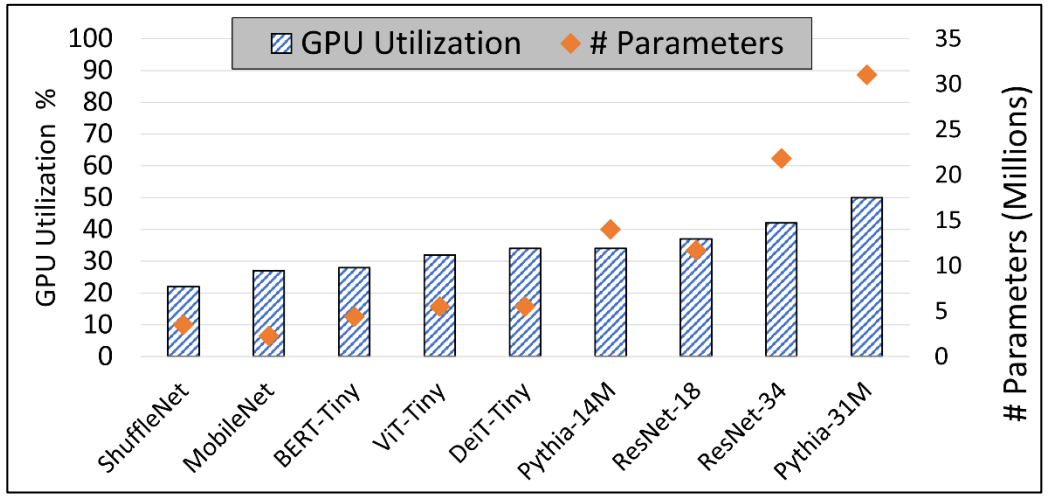
Hyperparameters determine the quality of a solution to which the model under training converges.



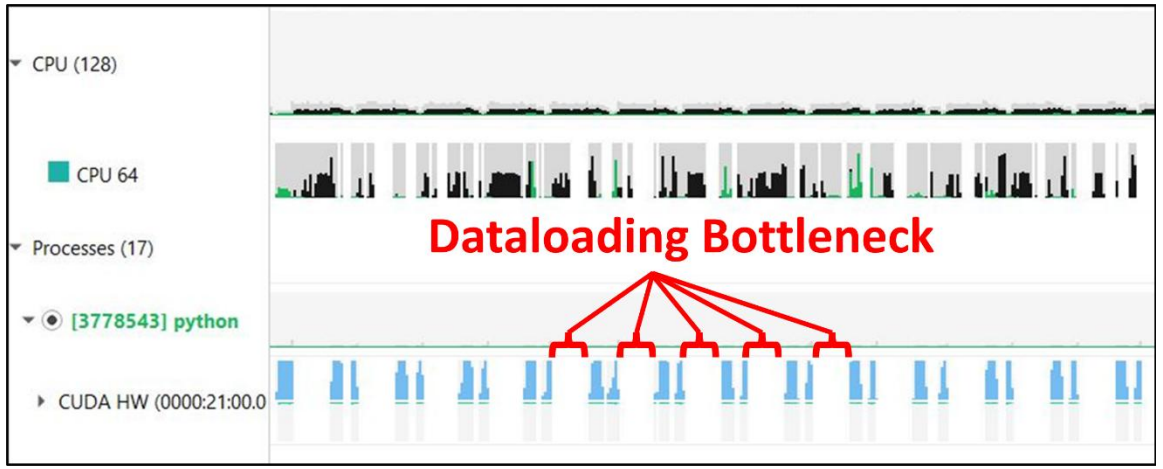
Courtesy: https://www.codegigs.app/parameter-and-hyperparameter-in-machine-learning/

Existing Parallelization Strategies for HPO

- Current HPO parallelization techniques underutilize powerful GPUs such as the NVIDIA A100s and H100s, especially for optimizing light weight DNNs.
- Lightweight DNNs have become increasingly essential due to their ability to perform low-latency and power-efficient inference in resource-constrained environments.
- We conduct a GPU utilization evaluation on Transformers, Vision Transformers, and CNNs with model sizes ranging between 2.3M and 31M parameters.



GPU utilization of the NVIDIA A100 training on different vision and language models.



Nsight Systems profile of model training showing dataloading bottlenecks

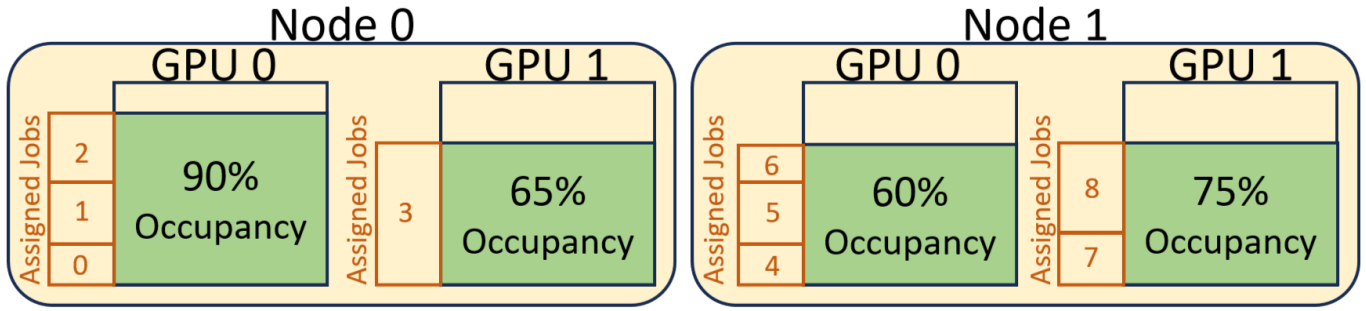
We observe that utilization is low for all models, ranging between 22% to 50% on the A100 GPU. We also observe that many lightweight DL models are bottlenecked by dataloading.

Presentation Outline

- Introduction and Motivation
- **Problems and Challenges**
- Proposed Design and Implementation
- Analysis and Performance Evaluation
- Conclusion

Problem Overview

- Given the underutilization of optimizing lightweight DL models, the main challenge lies in deciding the placement of HPO jobs on GPUs. We refer to this challenge as the *GPU Assignment Problem* ***(GAP).***
- Multiple factors must be considered to efficiently solve GAP, including the memory and compute requirements for DNN training, GPU architecture, number of GPUs, available CPU cores, GPU memory, and GPU compute.
- Naïve job scheduling policies may result in poor performance due to oversubscription or underutilization of resources.



Job Queue — Assigned — Pending

Job ID	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Weight	15	35	40	65	15	30	15	25	50	45	55	50	65	70	45	55

An example of the GPU Assignment Problem (GAP) with two nodes, two GPUs per node, and an HPO workload of 16 DNN training jobs.

This solution is non-optimal. Can we do better?

Problem Statements

1. **How can we schedule training jobs** of HPO workloads on modern heterogeneous GPU clusters to improve the GPU utilization and reduce the HPO makespan?
2. **What are the hardware constraints** for scheduling HPO workloads and how can we design scheduling policies that are aware of such constraints for job assignment?
3. **How can we design an HPO workflow/framework** that is adaptable to handle different scheduling policies, search spaces, DNN architectures including both vision and language models, and hardware architectures?
4. **How can we analyze job requirements**, monitor job status and hardware state, support efficient GPU space sharing, and provide resource elasticity and fault tolerance?

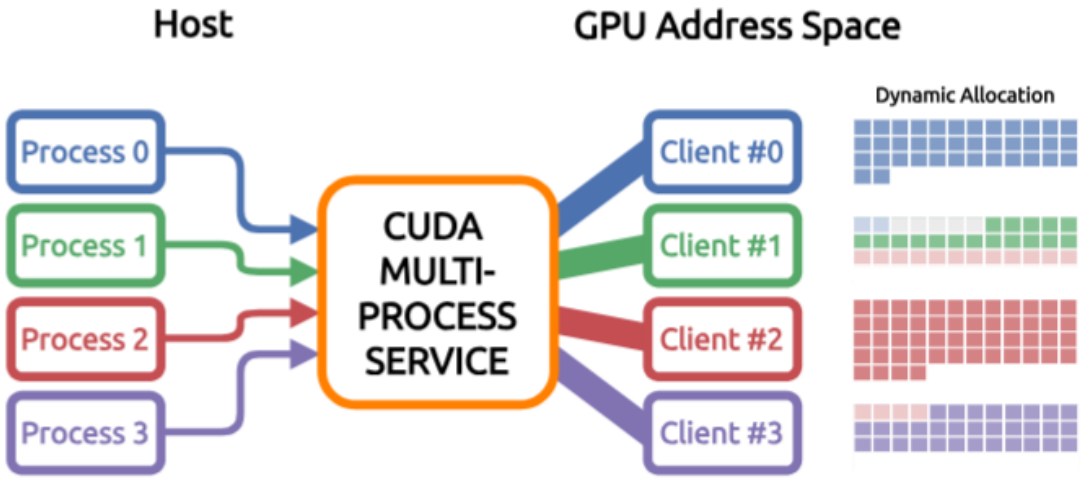
Note: In the context of this paper, makespan refers to the time taken to execute all HPO jobs in the search space until completion.

Presentation Outline

- Introduction and Motivation
- Problems and Challenges
- **Proposed Design and Implementation**
- Analysis and Performance Evaluation
- Conclusion

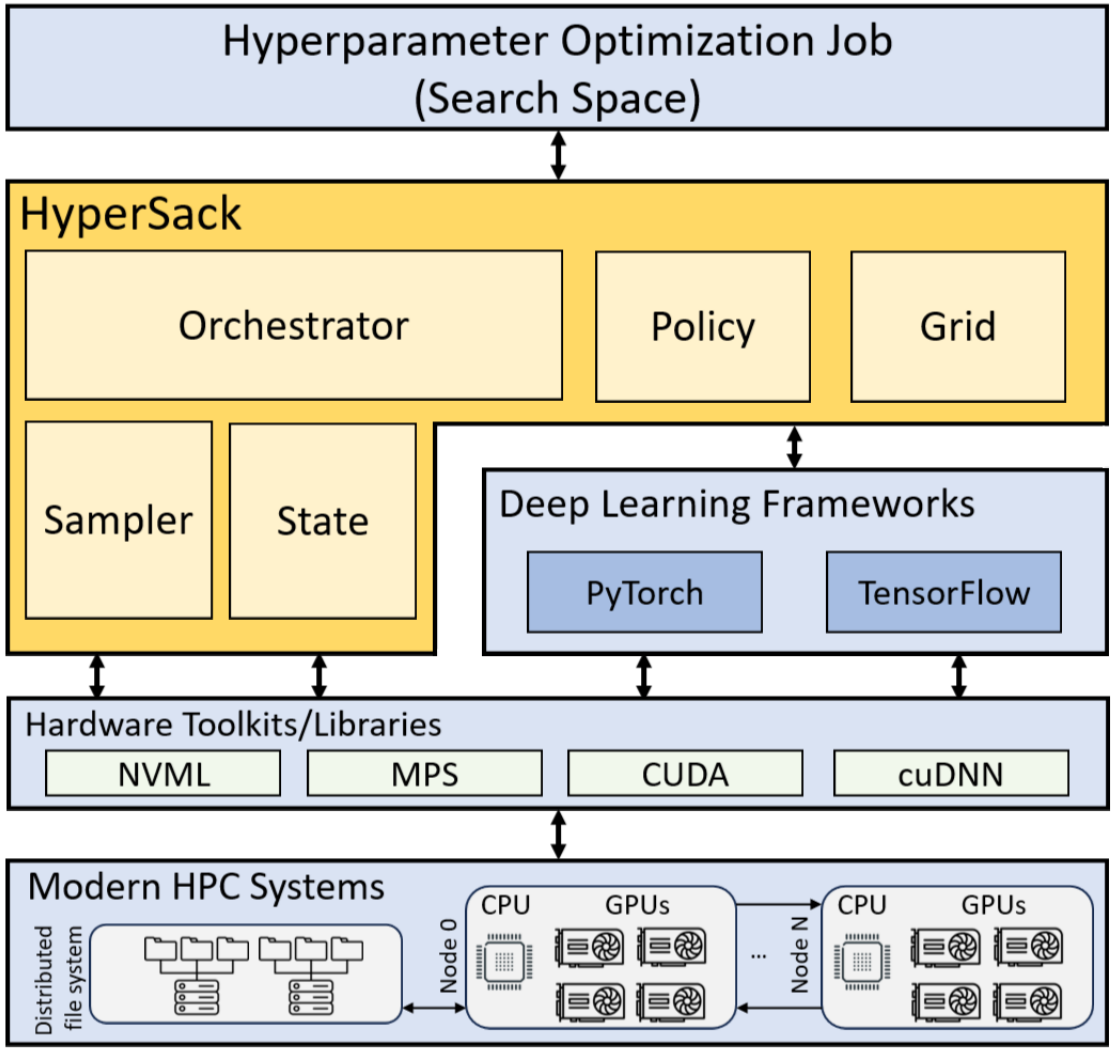
Background: NVIDIA Multi-Process Service (MPS)

- NVIDIA Multi-Process Service (MPS) is a feature provided by the CUDA API designed to improve the performance of multi-process GPU applications by enabling concurrent operations via a client-server paradigm.
- The MPS infrastructure assigns each CUDA process to an individual client context, each operating within a dedicated and secure GPU address space
- MPS particularly effective when dealing with many small-scale tasks that can be executed simultaneously using space-sharing to partition resources logically



Example workflow of NVIDIA Multi-Process Service (MPS) with 4 clients

High-level Architecture of the HyperSack

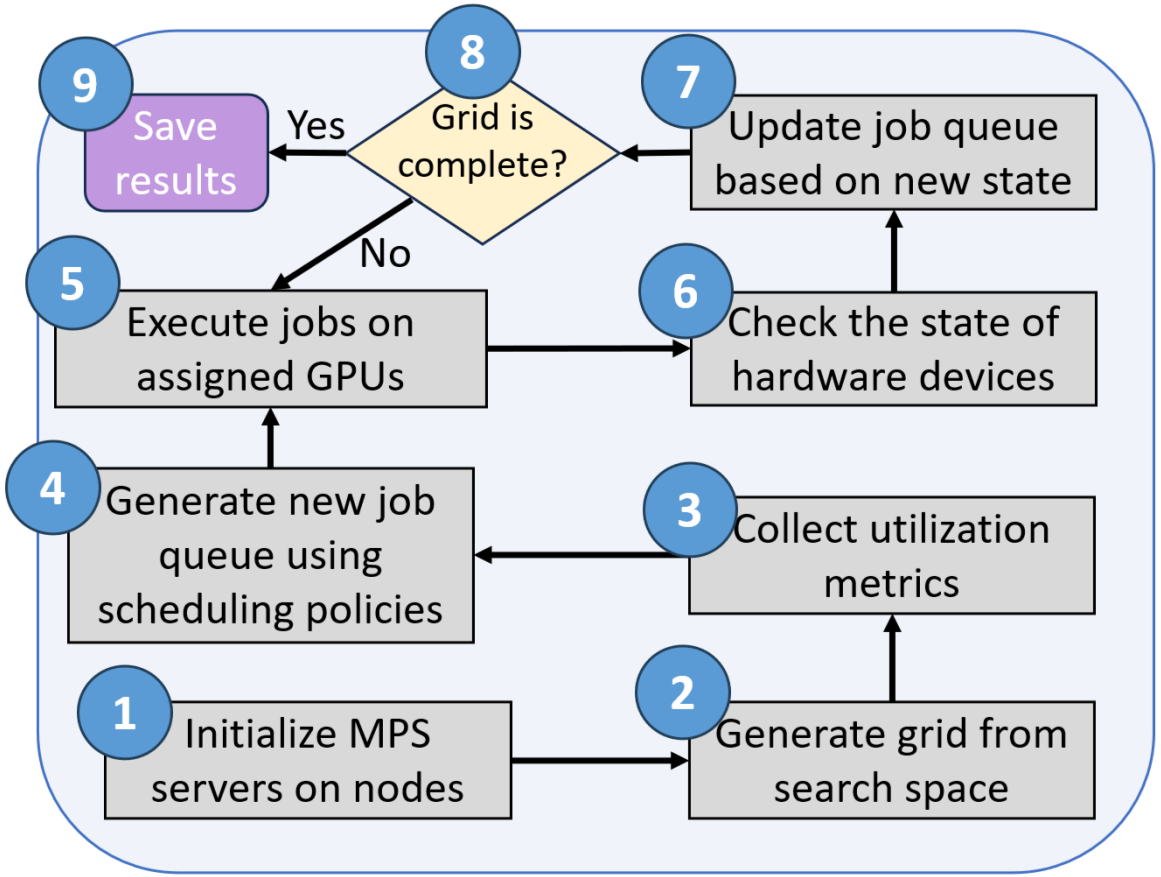


Layered Architecture of the proposed HyperSack framework.

Key components of HyperSack:

1. **Sampler:** captures hardware and DNN training metrics prior to training to make a scheduling decision.
2. **Grid:** a multi-dimensional matrix that represents the search space. Individual jobs can be in pending, in-progress, complete, or failed status.
3. **State:** responsible for monitoring the status of jobs and utilization of hardware devices and ensuring resource elasticity and fault tolerance.
4. **Policy:** responsible for making scheduling decision, by generating and updating job queues given the status of the Grid and State components.

Workflow of HyperSack



Workflow of the Orchestrator component responsible for coordinating all components of the HyperSack framework.

Proposed policies HyperSack's policies:

a) **First-fit (FF):** schedule a job on the any GPU that satisfies the problem constraints.

b) **First-fit decreasing (FFD):** sorts jobs in decreasing order with respect to the expected completion time of the individual jobs

c) **Worst-fit (WF):** gives assignment priority to the GPUs with the most compute capacity available.

d) **Worst-fit decreasing (WFD):** sorts jobs in decreasing order ensuring maximum overlap between long and short-running jobs and workload balancing across all available GPUs

5. **Orchestrator:** The Orchestrator controls the main logic of the HyperSack framework and coordinates between all of its components. Sequence of steps:

1. Initialize MPS server per GPU
2. Generate job configurations from the search space.
3. Collect utilization metrics using the Sampler.
4. Generate a new job queue using the Policy component.
5. Schedule jobs in the job queue on their assigned GPUs.
6. Continuously check the state of the hardware devices.
7. Update the job queue using the scheduling policies
8. Repeat step 5 until all jobs in the Grid are fully completed.
9. Save results and finalize the MPS server

Presentation Outline

- Introduction and Motivation
- Problems and Challenges
- Proposed Design and Implementation
- **Analysis and Performance Evaluation**
- Conclusion

Experimental Setup

Hardware:

- Ascend system (specs per node)
  - 2 AMD EPYC 7643 processors (64-cores per socket)
  - NVIDIA 200G HDR InfiniBand interconnect
  - 4 NVIDIA A100 graphics cards
- Lonestar6 system (specs per node)
  - A100 partition:
    - 2 AMD EPYC 7763 processors (64-cores per socket)
    - 3 NVIDIA A100 graphics cards
  - H100 partition:
    - 3 NVIDIA A100 graphics cards
    - 2 AMD EPYC 9454 processors (48-cores per socket)
  - NVIDIA 100G HDR InfiniBand

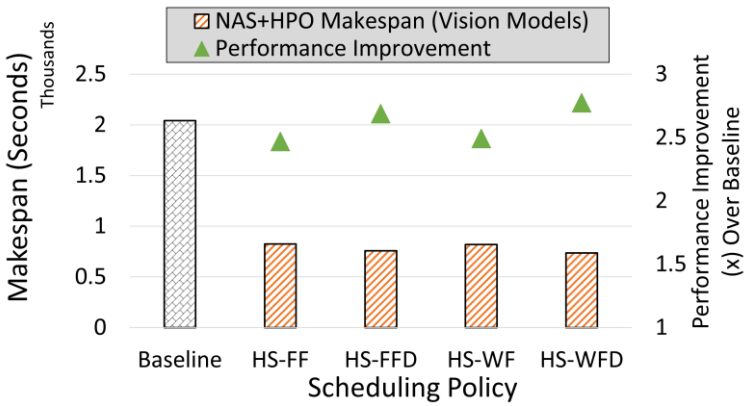
Software packages:

- CUDA 11.7, cuDNN 8.9.2, Python3.8.16, pyNVML 11.5, PyTorch 2.0.1, Transformers 4.30.2, and pandas 2.0.2

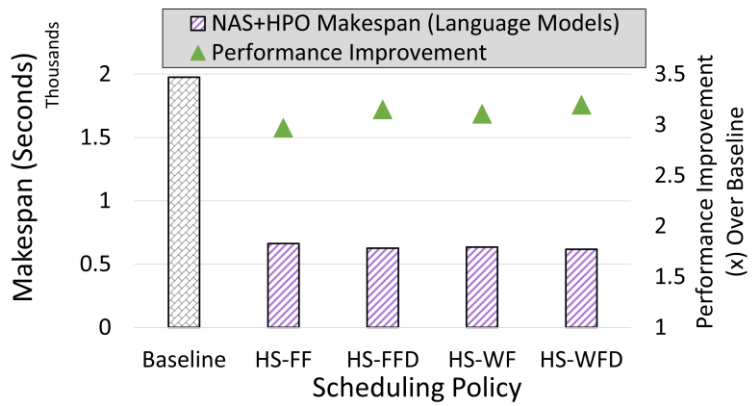
- Models:
  - Language Transformers: Pythia-14M, Pythia-34M, and BERT-Tiny.
  - Vision Transformers: ViT-Tiny, DeiT-Tiny
  - CNNs: ResNet-18, ResNet-34, ShuffleNet, and MobileNet
- Datasets:
  - Vision: CIFAR-10
  - Language: Stanford's IMDB dataset
- Baseline:
  - The baseline for all evaluations is the standard HPO scheduling scheme of one training job per GPU.
- Experimental Search Spaces:

	SP1	SP2	SP3	SP4
Models	1,2,3,4,5,6	7,8,9	5	1,2,3,4,5,6
Dataset	CIFAR-10	IMDB	CIFAR-10	CIFAR-10
Batch Size	64,128, 256,512	8,16, 32,64	64,128, 256,512	64,128, 256,512
Learning Rate	1e-5 to 1e-1	5e-5 to 5e-1	1e-5 to 1e-1	1e-5 to 1e-1
Total number of jobs	96	96	96	384

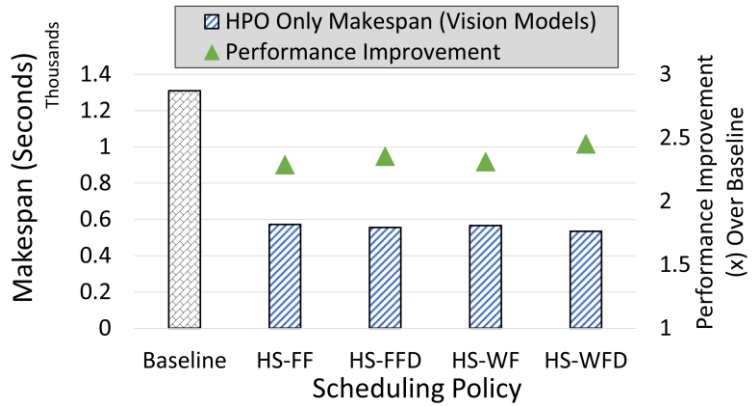
Scheduling Policies Comparison



Evaluation of the proposed scheduling policies for vision (multiple models) on SP1.



Evaluation of the proposed scheduling policies for language (multiple models) on SP2.



Evaluation of the proposed scheduling policies for vision (one model) on SP3.

- The HS-WFD (worst-fit decreasing) policy delivers the best performance across all workloads.
- HS-WFD delivers the best performance due to:
  1. It achieves maximum overlap between long and short-running jobs due to descending order.
  2. It achieves workload balancing across all available GPUs due to selecting GPUs with maximum capacity available.

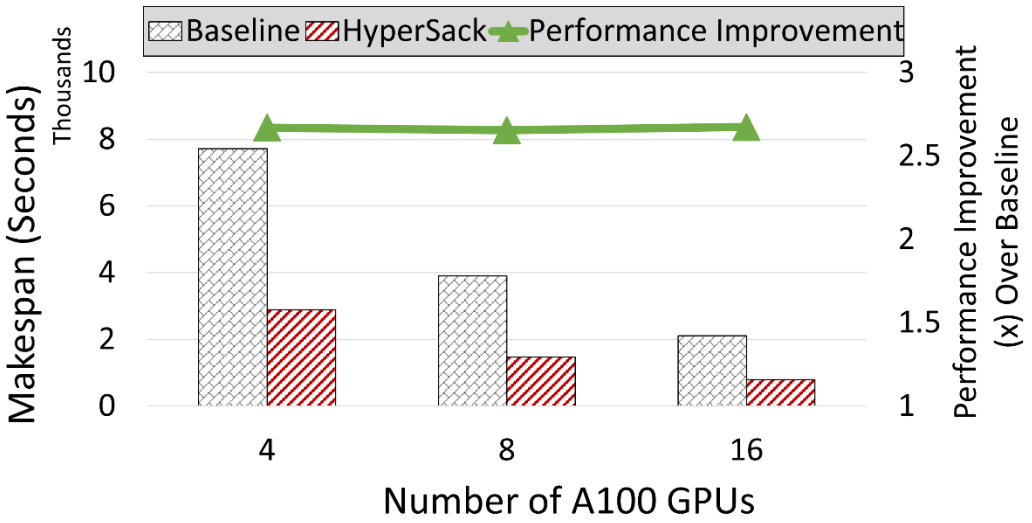
Policy	SP3 / 4 GPUs		SP4 / 16 GPUs	
	Occupancy achieved	Time to solve	Occupancy achieved	Time to solve
MIP Solver (Baseline)	99%	181.6 s	100%	212.8 s
HS-FF	88%	0.129 ms	97%	0.264 ms
HS-FFD	97%	0.163 ms	98%	0.298 ms
HS-WF	91%	0.135 ms	97%	0.319 ms
HS-WFD (Best)	98%	0.171 ms	100%	0.332 ms

The table shows the time needed for each algorithm to reach a scheduling decision and the maximum GPU occupancy achieved with different policies.

MIP solver from Google's OR-Tools is used as the baseline, where the problem is formed as a linear programming problem.

Scaling Efficiency

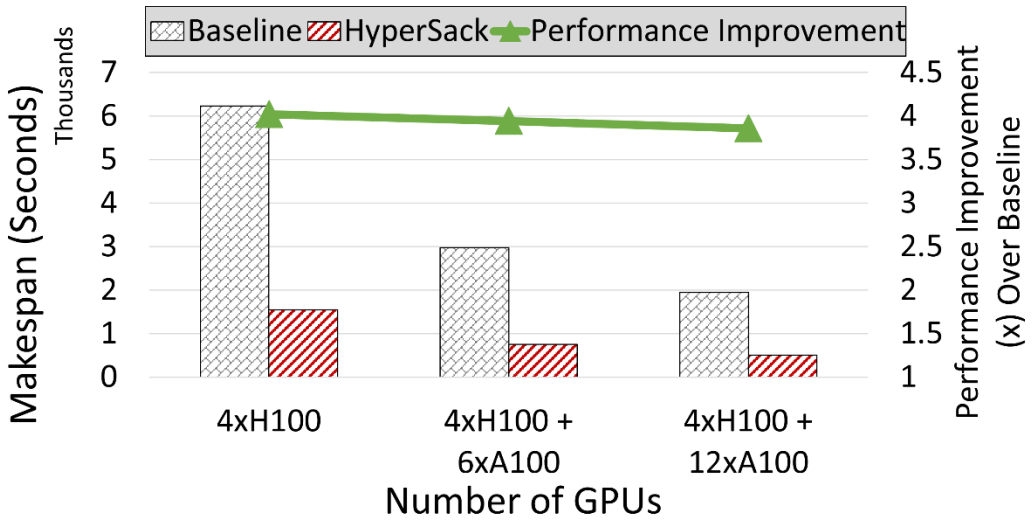
Homogenous Hardware



Evaluation of the scaling efficiency of HyperSack on multiple nodes and GPUs on SP4

- Using HyperSack, the makespan is reduced from:
  - 2.1 hours to 48.2 minutes on 4 GPUs
  - 1.1 hours to 24.5 minutes on 8 GPUs
  - 35.1 minutes to 13.1 minutes on 16 GPUs
- We observe a consistent performance improvement over the baseline of around 2.7x.

Heterogenous Hardware



Evaluation of HyperSack with heterogenous hardware resource of A100 and H100 GPUs on SP4

- Using HyperSack, the makespan is reduced from:
  - 1.7 hours to 25.8 minutes on 4 H100 GPUs
  - 49.5 minutes to 12.5 minutes on 4 H100 + 6 A100 GPUs
  - 2.5 minutesto 8.4 minutes on 4 H100 + 12 A100 GPUs.
- We observe performance improvement over the baseline of around 3.9-4x depending on the GPUs config.

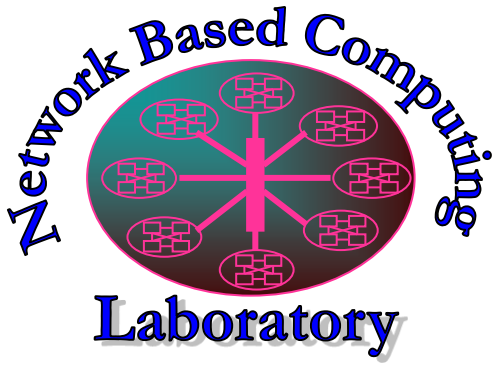
SP4 consists of 384 training jobs on lightweight vision models.

Conclusion and Future Work

- Existing HPO parallelization strategies for tuning lightweight DL models underutilize powerful GPU devices, such as the NVIDIA A100 and H100.
- Assigning multiple jobs to a GPU can improve resource utilization; still, naïve scheduling of HPO jobs may lead to poor performance.
- We refer to this challenge as the GPU Assignment Problem *GAP*. To address GAP, we propose HyperSack—a distributed HPO framework designed for dynamic and resource-aware scheduling on heterogeneous GPU-based HPC with resource elasticity and fault tolerance.
- For future work, we plan to extend HyperSack with more scheduling policies, HPO algorithms, and support for other GPU architectures from AMD and Intel.

Thank You!

{alnaasan.1, ramesh.113, yao.877, shafi.16, subramoni.1}@osu.edu, panda@cse.ohio-state.edu



Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

Full paper (HiPC 2024) available here:
https://ieeexplore.ieee.org/document/10884158/

The High-Performance MPI/PGAS Project
http://mvapich.cse.ohio-state.edu/

Follow us on
https://x.com/mvapich

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/