

# Distributed Deep Learning (DDL) with HopsML

## RISE Machine Learning Study Group

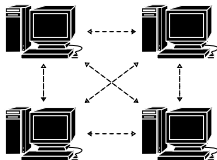
Kim Hammar

*kim@logicalclocks.com*

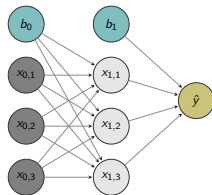
November 29, 2018

- 1 Distributed Deep Learning (DDL) Theory
- 2 HopsML: Distributed Deep Learning in Practice
- 3 Use-Case of DDL: Anti-Money-Laundering

## Distributed Computing



## Deep Learning



## Why Combine the two?

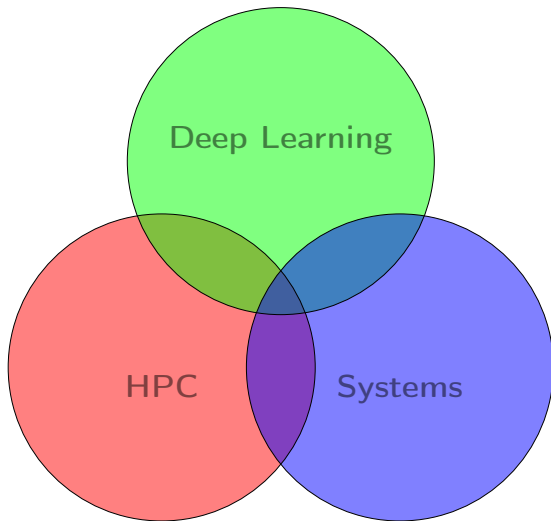
- More productive Data Science<sup>1</sup>
- Unreasonable effectiveness of data<sup>2</sup>
- To achieve state-of-the-art results<sup>3</sup>

<sup>1</sup>Alex Sergeev and title = Meet Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow howpublished = <https://eng.uber.com/horovod/> note = Accessed: 2018-11-24 Mike Del Balso year=2017.

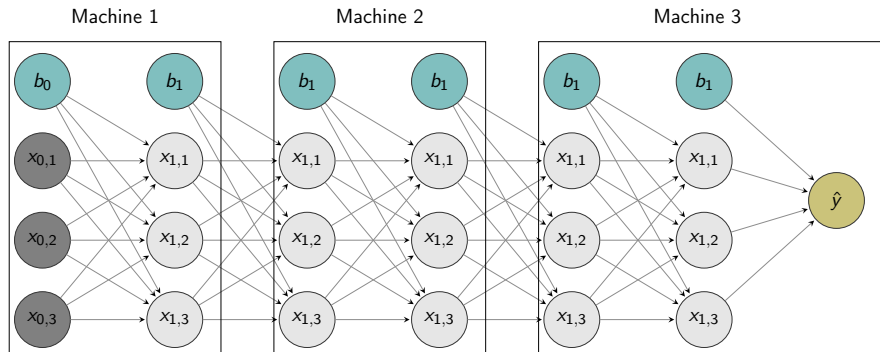
<sup>2</sup>Chen Sun et al. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In: *CoRR* abs/1707.02968 (2017). arXiv: 1707.02968. URL: <http://arxiv.org/abs/1707.02968>.

<sup>3</sup>Jeffrey Dean et al. "Large Scale Distributed Deep Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1223–1231. URL: <http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf>.

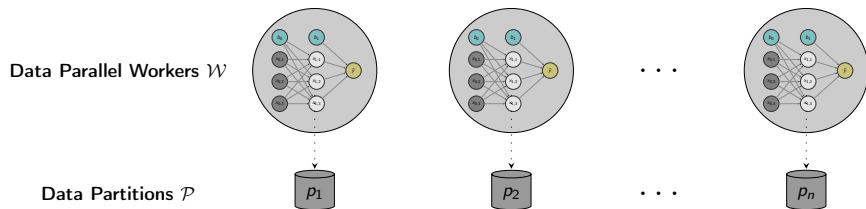
# What is Distributed Deep Learning?



# Model Parallelism



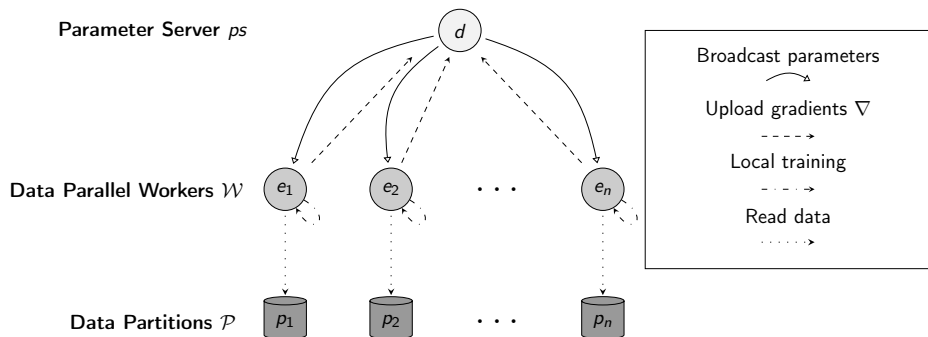
# Data Parallelism



# When to use Model Parallel and Data Parallel?

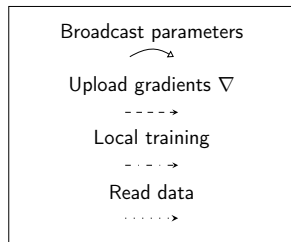
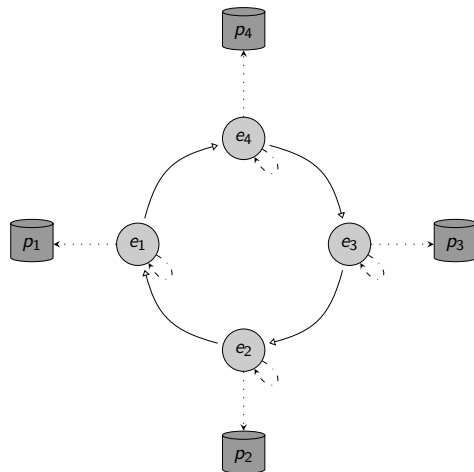
- How big is your model parameters  $\theta$  vs GPU memory? If  $size(\theta) > size(gpu)$  you have to use **model parallelism**
- If your model fits on a single GPU  $\implies$  in 99.999% you want to use **data parallelism** to reduce training time

# Parameter Server Architecture

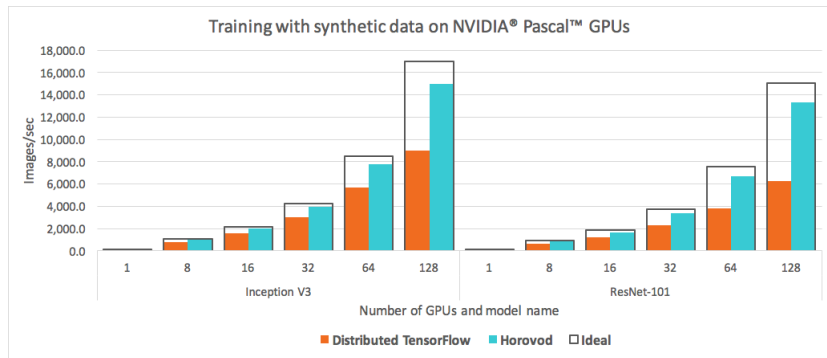




# Ring-All-Reduce Architecture



# When to use Parameter-Server and when to use Ring-All-Reduce?



Ring-all-reduce scales better  $\implies$  generally prefer ring-all-reduce<sup>4</sup>

<sup>4</sup>Alex Sergeev and title = Meet Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow howpublished = <https://eng.uber.com/horovod/> note = Accessed: 2018-11-24 Mike Del Balso year=2017.

# How to get started?

## ICE (RISE SICS NORTH) provides the hardware that you need

- GPU Machines for training ✓
- CPU Machines for data prep ✓
- Disks for storing large datasets ✓

## HopsML provides the ML infrastructure that you need

- Fast Distributed File System ✓
- Spark-jobs and notebooks for data prep ✓
- Framework for reproducible and versioned parallel experiments ✓
- Framework for distributed training ✓
- Framework for monitoring training ✓
- Support for auto-scaling model serving ✓
- Feature store ✗(Soon!)

# Hopsworks: UI-driven front-end to the ML infrastructure

The screenshot displays the Hopsworks interface for a project named 'demo\_deep\_learnin...'. The left sidebar contains navigation options: Jupyter, Jobs, Kafka, Model Serving, Experiments, Data Sets, Settings, Python, Members, and Metadata Designer. At the bottom of the sidebar, GPU resource usage is shown: 'Cluster Utilization: 2%', 'Allocated GPUs: 4', 'Available GPUs: 20', and 'Queued GPU Requests: 0'. A red arrow points to the 'Allocated GPUs' bar.

The main content area features a search bar at the top and tabs for 'Spark', 'YARN', 'Logs', and 'Metrics'. The 'TensorBoard' tab is active, showing a dropdown menu with four tasks: 'TensorBoard.task0', 'TensorBoard.task1', 'TensorBoard.task2', and 'TensorBoard.task3'. A red arrow points to 'TensorBoard.task1'. Below the dropdown, the 'accuracy' chart is visible, showing a fluctuating line graph over time (0.000 to 1.200k steps). The 'global\_step' chart below it shows 'sec' values ranging from 15.4 to 16.2. A red arrow points from the text 'Running and monitoring 4 parallel experiments with GPUs' to the TensorBoard task dropdown.

Running and monitoring 4 parallel experiments with GPUs

Allocated GPUs

# Python-First API-powered workflow

Write your regular tensorflow/python/pytorch/keras code and put it in a function, for example called `collective_all_reduce_mnist`, then you can create a reproducible experiment using many GPUs and collective-all-reduce as follows:

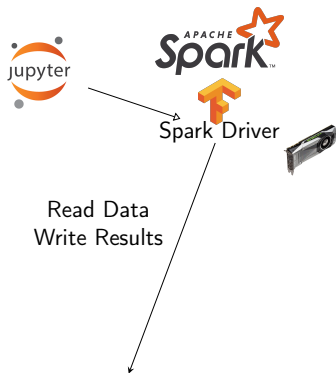
---

```
from hops import experiment
from hops import hdfs

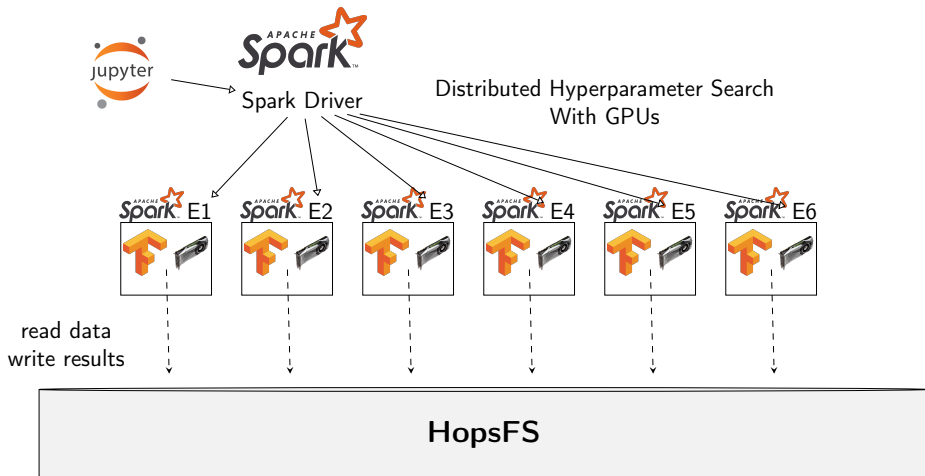
notebook = hdfs.project_path() +
"Jupyter/Distributed_Training/collective_allreduce_strategy/mnist.ipynb"
experiment.collective_all_reduce(collective_all_reduce_mnist,
                                name='mnist estimator',
                                description='A minimal mnist example with two hidden layers',
                                versioned_resources=[notebook], local_logdir=True)
```

---

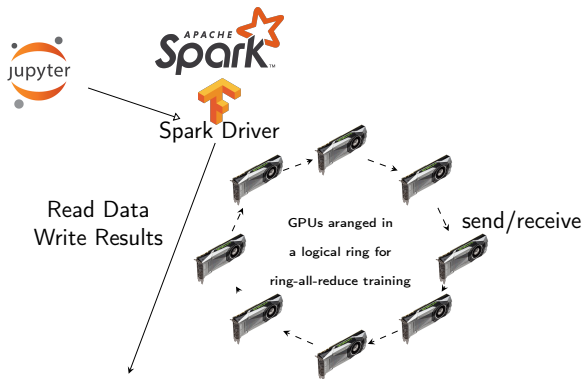
# Single-GPU Training on Hops



# Parallel Experiments on Hops

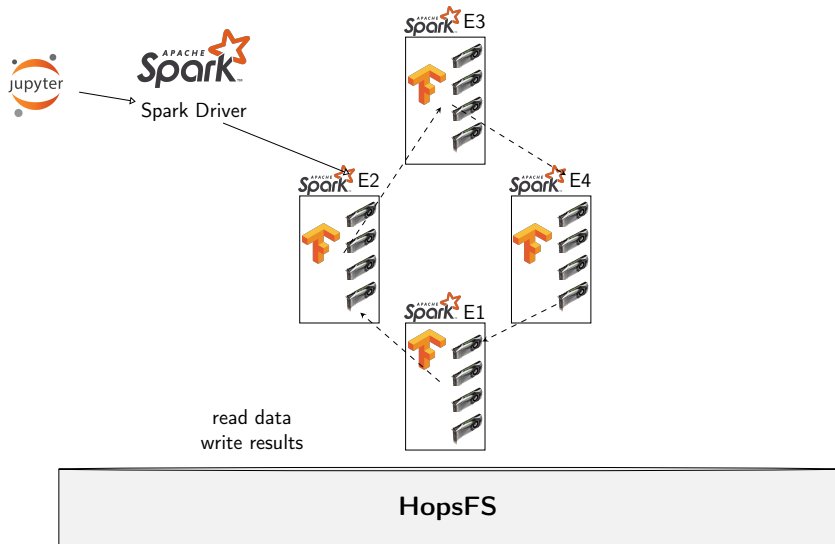


# Multi-GPU Training on Hops

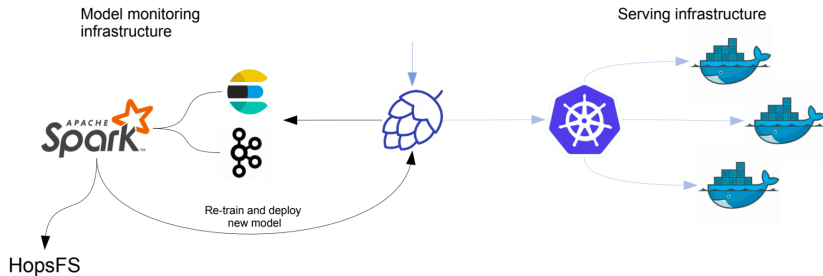




# Distributed GPU Training on Hops



## Model Monitoring



- Register at `hops.site`, email: `kim@logicalclocks.com` if your registration is not approved
- Try out the deep learning tour on `hopsworks`
- Example code:  
`https://github.com/logicalclocks/hops-examples`
- Look at the docs: `https://www.hops.io/`
- If you get stuck, write on gitter:  
`https://gitter.im/hopshadoop/hopsworks`

# DEMO