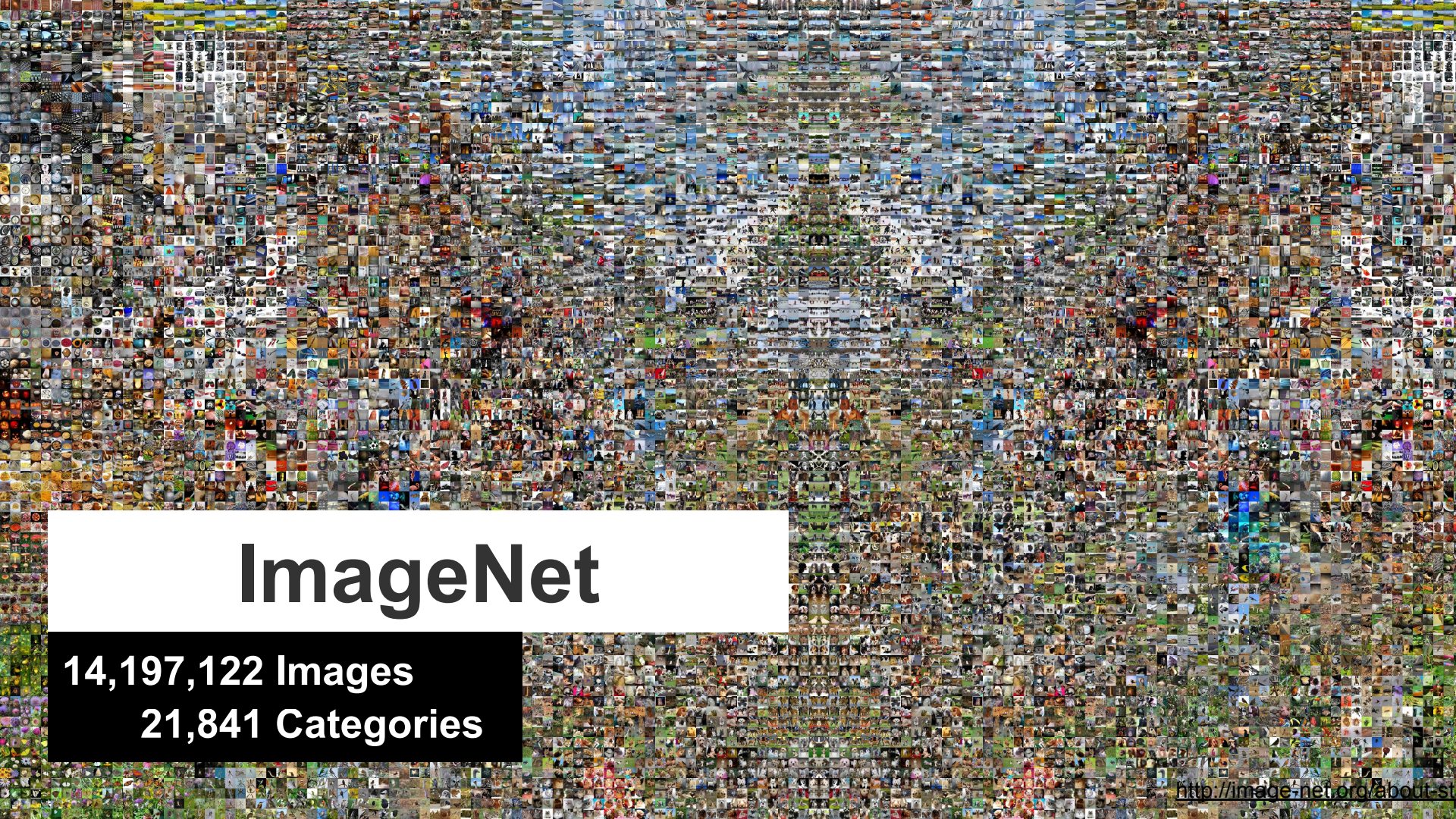
Training large deep learning models using Spark, TensorFlow and R

@javierluraschi
SDSS 2020 / RStudio PBC


ImageNet
14,197,122 Images
21,841 Categories
http://image-net.org/about-st

fast.ai

Making neural nets uncool again

Home

About

Our MOOC

Now anyone can train Imagenet in 18 minutes

Written: 10 Aug 2018 by *Jeremy Howard*

This post extends the work described in a previous post, Training Imagenet in 3 hours for $25; and CIFAR10 for $0.26.

A team of fast.ai alum Andrew Shaw, DIU researcher Yaroslav Bulatov, and I have managed to train Imagenetto 93% accuracy in just 18 minutes, using 16 public AWScloud instances, each with 8 NVIDIA

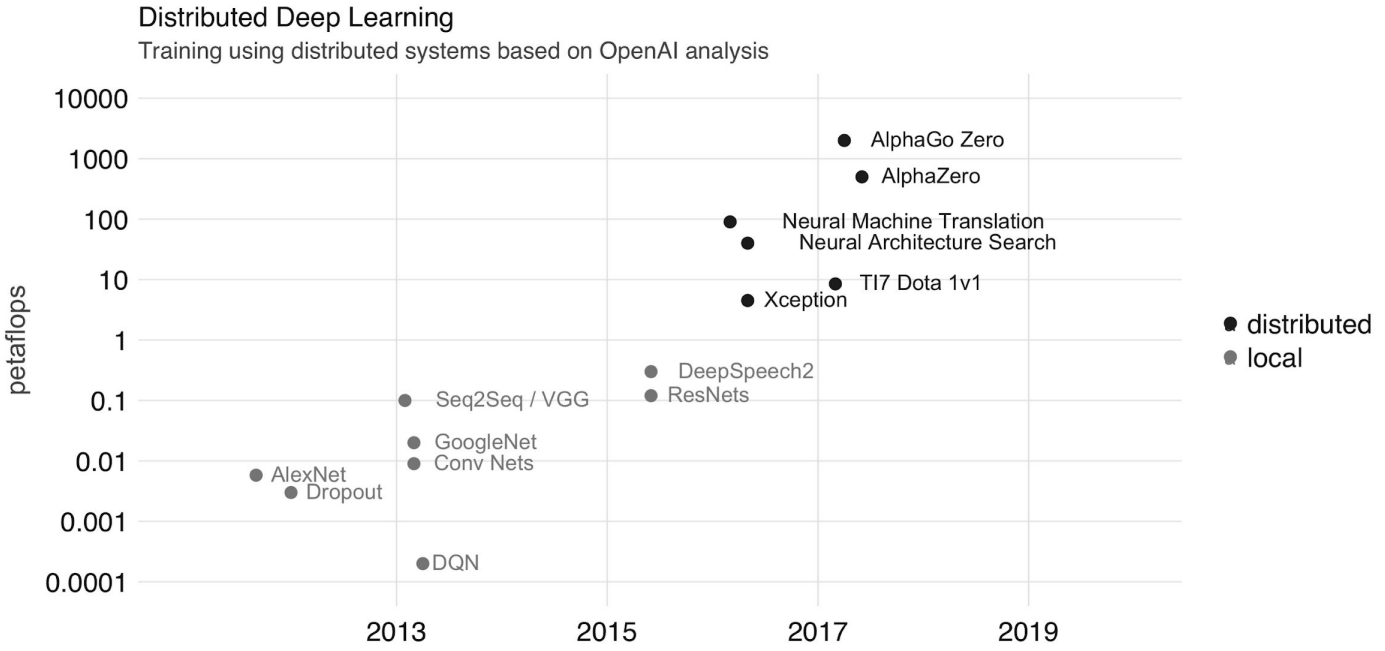
V100 GPUs, running the fastai and PyTorch libraries. This is a new speed record for training Imagenet to this accuracy on publicly available infrastructure, and is 40% faster than Google's DAWNBench record on their proprietary TPU Pod cluster. Our approach uses the same number of processing units as Google's benchmark (128) and costs around $40 to run.

DIU and fast.ai will be releasing software to allow anyone to easily train and monitor their own distributed models on AWS, using the best practices developed in this project. The main training methods we used (details below) are: fast.ai's progressive resizing for classification, and rectangular image validation; NVIDIA's NCCL with PyTorch's all-reduce; Tencent's weight decay tuning; a variant of Google Brain's dynamic batch sizes, gradual learning rate warm-up (Goyal et al 2018, and Leslie Smith 2018). We used the classic ResNet-50 architecture, and SGD with momentum.

https://www.fast.ai/2018/08/10/fastai-diu-imagenet/

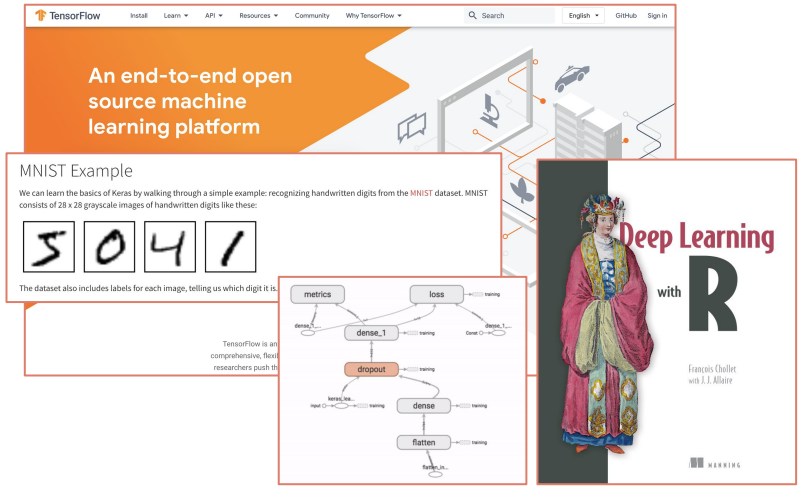
Distributed Deep Learning

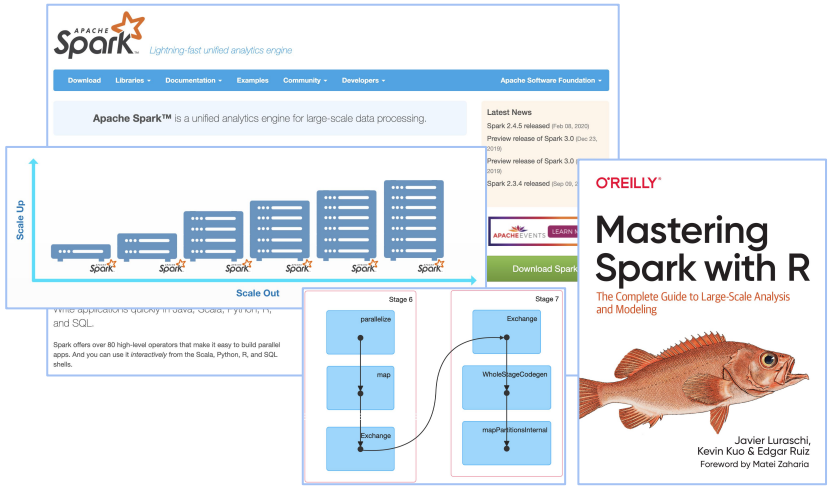
Recent state-of-the-art models require distributed training.



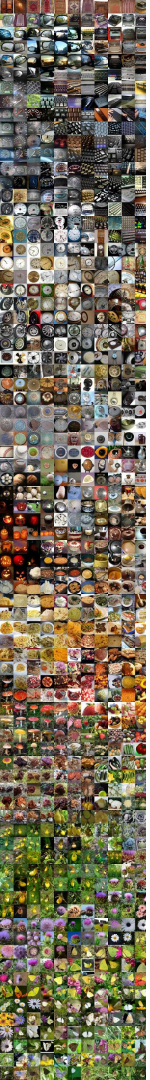
Overview

This talk assumes you are somewhat familiar with TensorFlow and Spark:





Talk Outline: **Distributed Deep Learning, TensorFlow** and **Spark.**



Distributed Deep Learning

Deep Learning Recap: AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

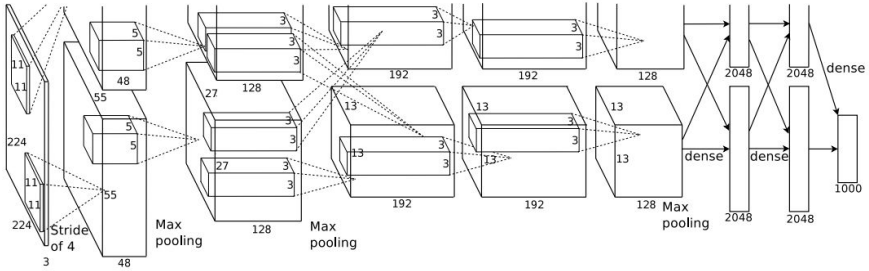
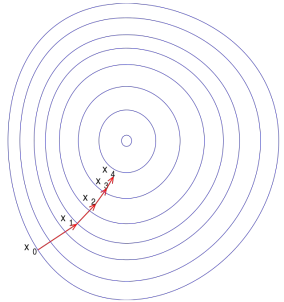


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.



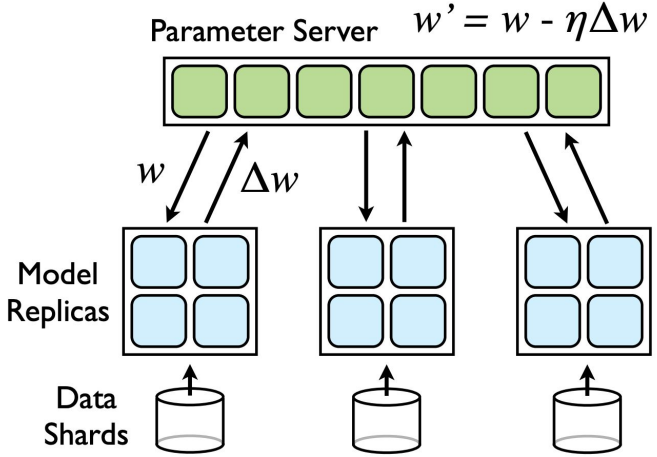
$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Parameter Server @Google

Large Scale Distributed Deep Networks

Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng
{jeff, gcorrado}@google.com
Google Inc., Mountain View, CA



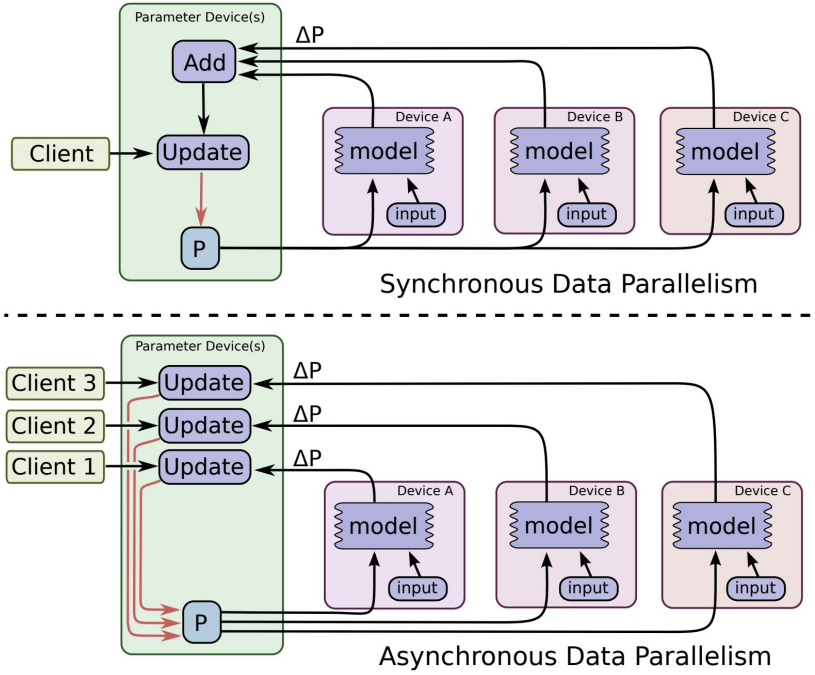
the same order. Moreover, because the model replicas are permitted to fetch parameters and push gradients in separate threads, there may be additional subtle inconsistencies in the timestamps of parameters. There is little theoretical grounding for the safety of these operations for nonconvex problems, but in practice we found relaxing consistency requirements to be remarkably effective.

http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf

Parameter Server @TensorFlow

TensorFlow:
Large-Scale Machine Learning on Heterogeneous Distributed Systems
(Preliminary White Paper, November 9, 2015)

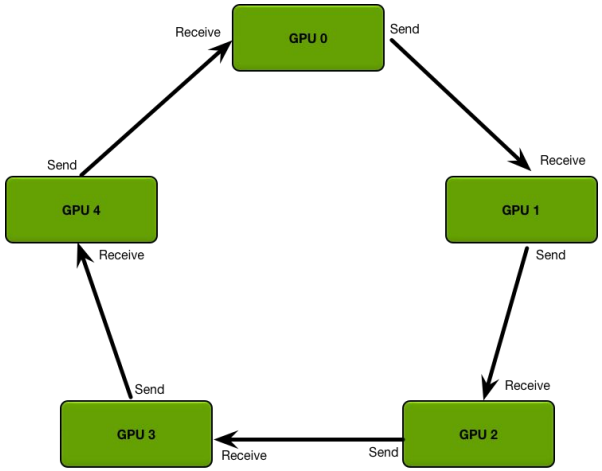
Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng
Google Research*



https://arxiv.org/pdf/1603.04467.pdf

Ring All-Reduce @Baidu

Two steps: First, a scatter-reduce, and then, an allgather.



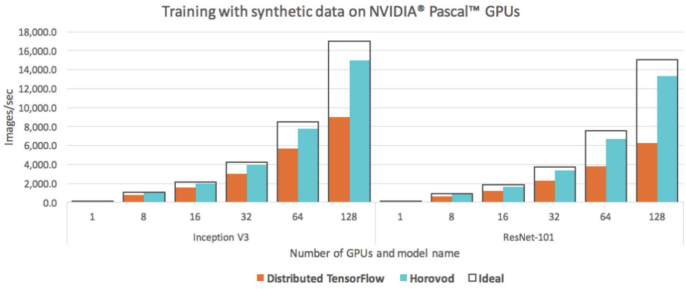
https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

Ring All-Reduce @Horovod

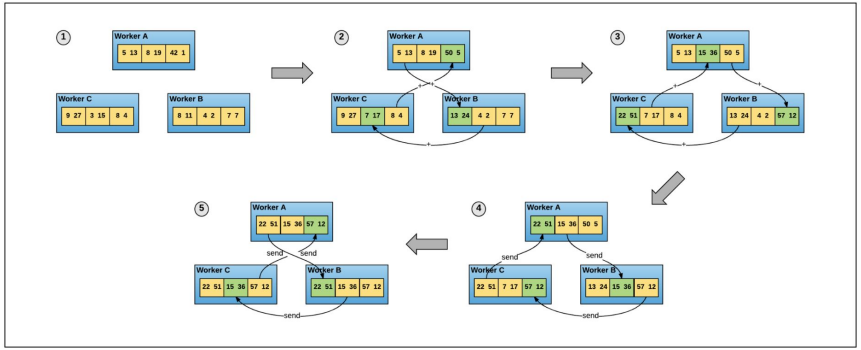
Horovod: fast and easy distributed deep learning in TensorFlow

Alexander Sergeev
Uber Technologies, Inc.
asergeev@uber.com

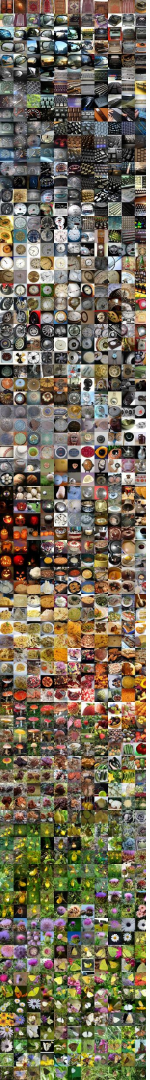
Mike Del Balso
Uber Technologies, Inc.
mdb@uber.com



In early 2017 Baidu published an article [8] evangelizing a different algorithm for averaging gradients and communicating those gradients to all nodes (Steps 2 and 3 above), called ring-allreduce, as well as a fork of TensorFlow through which they demonstrated a draft implementation of this algorithm. The algorithm was based on the approach introduced in the 2009 paper by Patarasuk and Yuan [9].

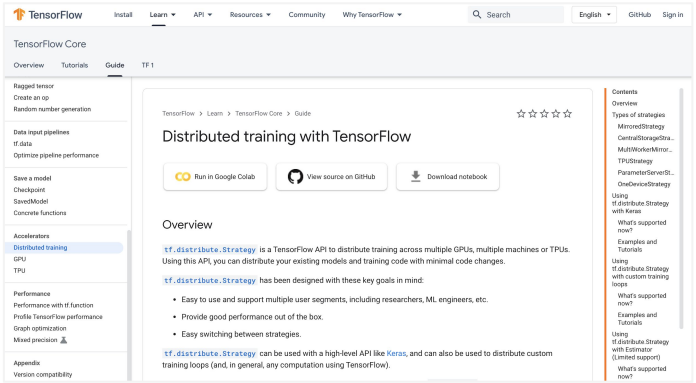


https://arxiv.org/pdf/1802.05799.pdf



TensorFlow

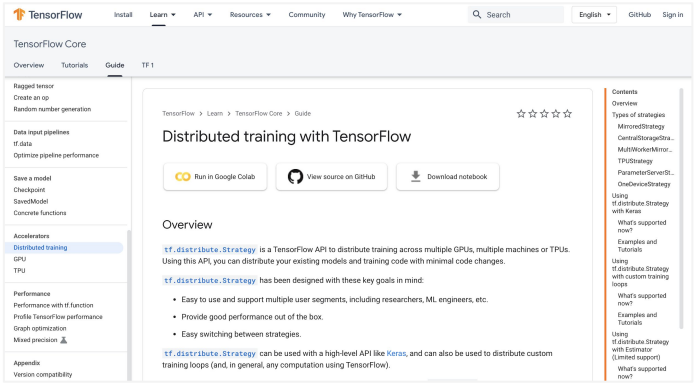
Distributed Training with TensorFlow



Training API	MirroredStrategy	TPUStrategy	MultiWorkerMirroredStrategy	CentralStorageStrategy	ParameterServerStrategy	OneDeviceStrategy
Keras API	Supported	Experimental support	Experimental support	Experimental support	Supported planned post 2.0	Supported
Custom training loop	Experimental support	Experimental support	Support planned post 2.0	Support planned post 2.0	No support yet	Supported
Estimator API	Limited Support	Not supported	Limited Support	Limited Support	Limited Support	Limited Support

https://www.tensorflow.org/guide/distributed_training

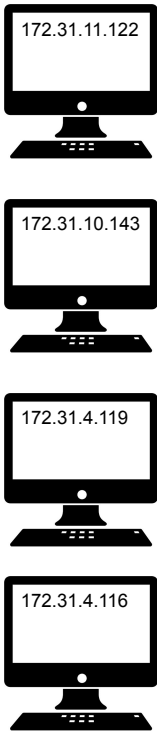
Distributed Training with TensorFlow



Training API	MirroredStrategy	TPUStrategy	MultiWorkerMirroredStrategy	CentralStorageStrategy	ParameterServerStrategy	OneDeviceStrategy
Keras API	Supported	Experimental support	Experimental support	Experimental support	Supported planned post 2.0	Supported
Custom training loop	Experimental support	Experimental support	Support planned post 2.0	Support planned post 2.0	No support yet	Supported
Estimator API	Limited Support	Not supported	Limited Support	Limited Support	Limited Support	Limited Support

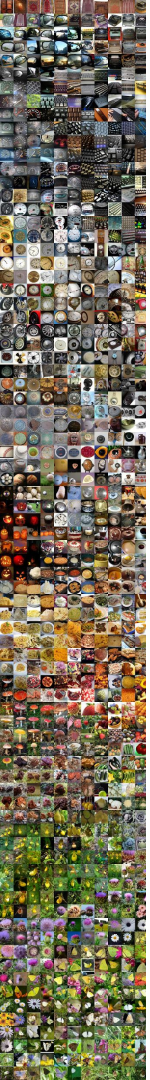
https://www.tensorflow.org/guide/distributed_training

Configuration File




```
# run in main worker
Sys.setenv(TF_CONFIG = jsonlite::toJSON(list(
    cluster = list(
        worker = c("172.31.11.122:10090", "172.31.10.143:10088", "172.31.4.119:10087", "172.31.4.116:10089")
    ),
    task = list(type = 'worker', index = 0)
), auto_unbox = TRUE))

# run in worker(1)
Sys.setenv(TF_CONFIG = jsonlite::toJSON(list(
    cluster = list(
        worker = c("172.31.11.122:10090", "172.31.10.143:10088", "172.31.4.119:10087", "172.31.4.116:10089")
    ),
    task = list(type = 'worker', index = 1)
), auto_unbox = TRUE))

# run in worker(2)
Sys.setenv(TF_CONFIG = jsonlite::toJSON(list(
    cluster = list(
        worker = c("172.31.11.122:10090", "172.31.10.143:10088", "172.31.4.119:10087", "172.31.4.116:10089")
    ),
    task = list(type = 'worker', index = 2)
), auto_unbox = TRUE))

# run in worker(3)
Sys.setenv(TF_CONFIG = jsonlite::toJSON(list(
    cluster = list(
        worker = c("172.31.11.122:10090", "172.31.10.143:10088", "172.31.4.119:10087", "172.31.4.116:10089")
    ),
    task = list(type = 'worker', index = 3)
), auto_unbox = TRUE))
```

https://github.com/mlverse/distributed



Distributed Pseudocode

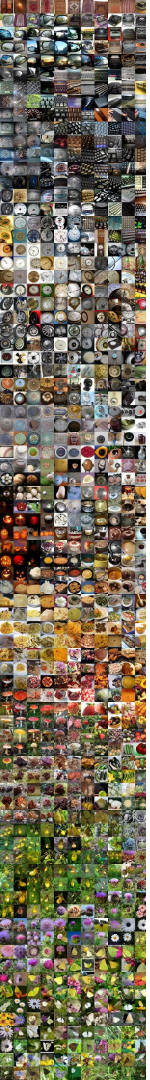
Local Model

```
library(tensorflow)
library(keras)

# create model
model ← keras_model_sequential() # %>% ...

# compile model
model %>% compile()

# fit model
model %>% fit()
```

Distributed Model

```
# define configuration
Sys.setenv(TF_CONFIG = "")

library(tensorflow)
library(keras)

# define strategy
strategy ← tf$distribute$experimental$MultiWorkerMirroredStrategy()
with (strategy$scope(), {
  # create model
  model ← keras_model_sequential() # %>% ...

  # compile model
  model %>% compile()
})

# fit model
model %>% fit()
```

https://github.com/mlverse/distributed



Distributed Code

Local Model

```
library(tensorflow)
library(keras)

batch_size <- 64L

mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y

x_train <- array_reshape(x_train, c(nrow(x_train), 28, 28, 1))
x_train <- x_train / 255

model <- keras_model_sequential() %>%
  layer_conv_2d(
    filters = 32,
    kernel_size = 3,
    activation = 'relu',
  input_shape = c(28, 28, 1)
  ) %>%
    layer_max_pooling_2d() %>%
    layer_flatten() %>%
    layer_dense(units = 64, activation = 'relu') %>%
    layer_dense(units = 10)

model %>% compile(
  loss = tf$keras$losses$SparseCategoricalCrossentropy(from_logits = TRUE),
  optimizer = tf$keras$optimizers$SGD(learning_rate = 0.001),
  metrics = 'accuracy')

model %>% fit(x_train, y_train, batch_size = batch_size, epochs = 3, steps_per_epoch = 5)
```

Distributed Model

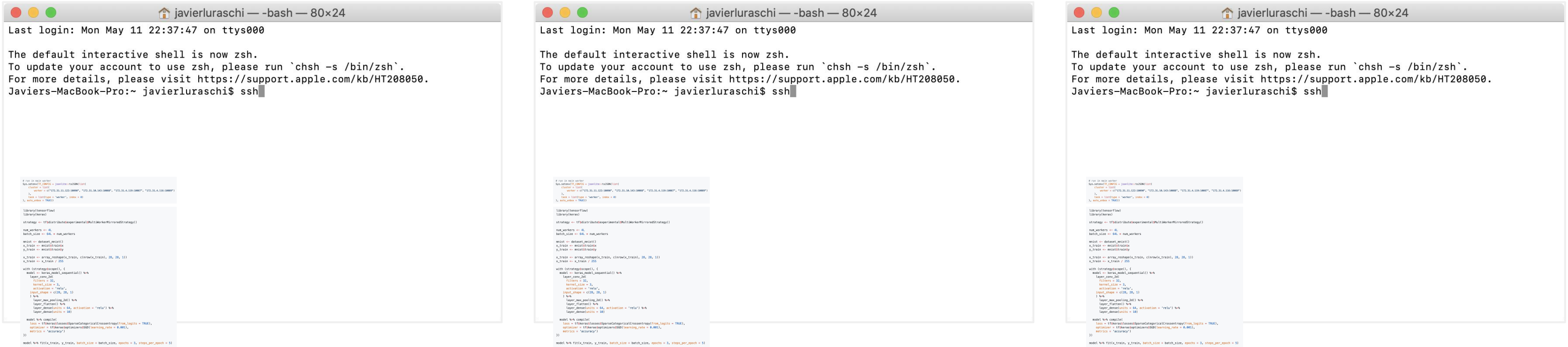
```
# run in main worker
Sys.setenv(TF_CONFIG = jsonlite::toJSON(list(
    cluster = list(
        worker = c("172.31.11.122:10090", "172.31.10.143:10088", "172.31.4.119:10087", "172.31.4.116:10089")
    ),
    task = list(type = 'worker', index = 0)
), auto_unbox = TRUE))
```

```
library(tensorflow)
library(keras)

strategy <- tf$distribute$experimental$MultiWorkerMirroredStrategy()

num_workers <- 4L
batch_size <- 64L * num_workers

mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y

x_train <- array_reshape(x_train, c(nrow(x_train), 28, 28, 1))
x_train <- x_train / 255

with (strategy$scope(), {
  model <- keras_model_sequential() %>%
    layer_conv_2d(
      filters = 32,
      kernel_size = 3,
      activation = 'relu',
    input_shape = c(28, 28, 1)
    ) %>%
      layer_max_pooling_2d() %>%
      layer_flatten() %>%
      layer_dense(units = 64, activation = 'relu') %>%
      layer_dense(units = 10)

  model %>% compile(
    loss = tf$keras$losses$SparseCategoricalCrossentropy(from_logits = TRUE),
    optimizer = tf$keras$optimizers$SGD(learning_rate = 0.001),
    metrics = 'accuracy')
})

model %>% fit(x_train, y_train, batch_size = batch_size, epochs = 3, steps_per_epoch = 5)
```

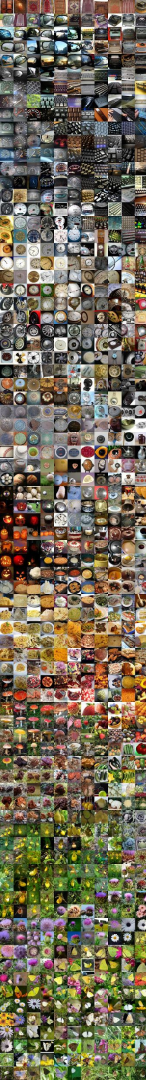
https://github.com/mlverse/distributed

Distributed Training

SSH to each machine, transfer **data subset**, run the code in each machine.

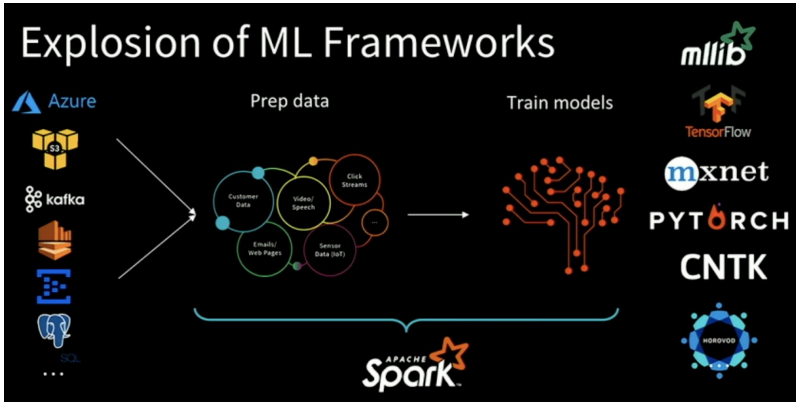


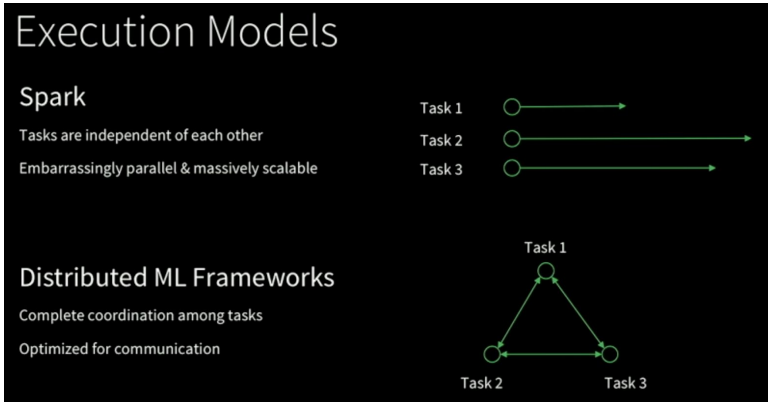
What if you need dozens of machines?

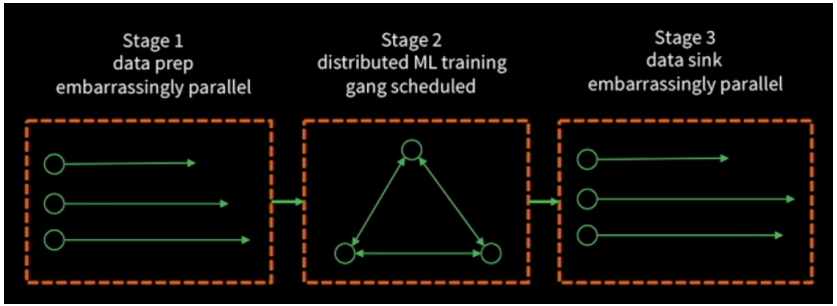


Spark

Deep Learning Frameworks and Spark



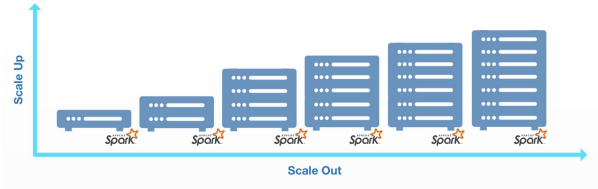


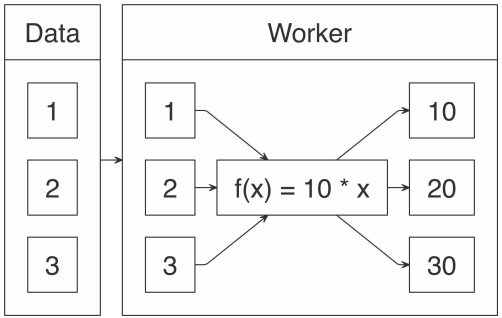


[Project Hydrogen: State-of-the-Art Deep Learning on Apache Spark Reynold Xin]

Spark Apply

You can use **spark_apply()** to apply an arbitrary transformation in R:

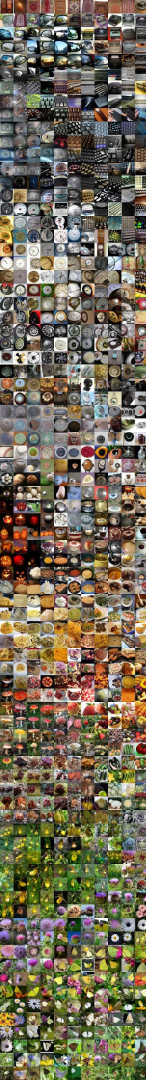





```
sdf_len(sc, 3) %>%
  spark_apply(~ 10 * .x)
```

```
# Source: spark<?> [?? x 1]
     id
* <dbl>
1    10
2    20
3    30
```

Can't we just use **spark_apply()** to run TensorFlow code?



Barrier Execution

Barrier Execution supported since sparklyr 1.1 and Spark 2.4. Used to schedule all-or-nothing resources, retrieve IP addresses and process data.

Barrier Execution

Barrier execution is a new feature introduced in Spark 2.4 which enables Deep Learning on Apache Spark by implementing an all-or-nothing scheduler into Apache Spark. This allows Spark to not only process analytic workflows, but also to use Spark as a high-performance computing cluster where other framework, like OpenMP or TensorFlow Distributed, can reuse cluster machines and have them directly communicate with each other for a given task.

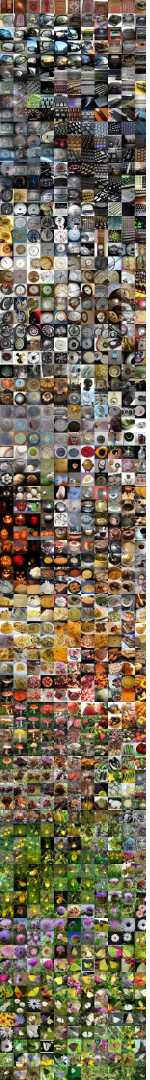
In general, we don't expect most users to use this feature directly; instead, this is a feature relevant to advanced users interested in creating extensions that support additional modeling frameworks. You can learn more about barrier execution in Reynold Xin's keynote.

To use barrier execution from R, set the `barrier = TRUE` parameter in `spark_apply()` and then make use of a new parameter in the R closure to retrieve the network address of the additional nodes available for this task. A simple example follows:

```
library(sparklyr)
sc <- spark_connect(master = "local", version = "2.4")

sdf_len(sc, 1, repartition = 1) %>%
  spark_apply(~ .y$address, barrier = TRUE, columns = c(address = "character")) %>%
  collect()
```

```
# A tibble: 1 x 1
  address
  <chr>
1 localhost:50693
```

https://blog.rstudio.com/2020/01/29/sparklyr-1-1/#barrier-execution



Spark and TensorFlow Pseudocode

```
library(sparklyr)
sc ← spark_connect(master = "local|yarn|etc")

# partition dataset
sdf_len(sc, 3, repartition = 3) %>%
  spark_apply(function(df, barrier) {
      library(tensorflow)
      library(keras)

      # define configuration from barrier
      Sys.setenv(TF_CONFIG = "")

      # define strategy and model
      strategy ← MultiWorkerMirroredStrategy()
      with (strategy$scope(), {
        model ← keras_model_sequential() # %>% ...
        model %>% compile()
      })

      # fit and retrieve model
      model %>% fit()
  }, barrier = TRUE) %>% collect()
```

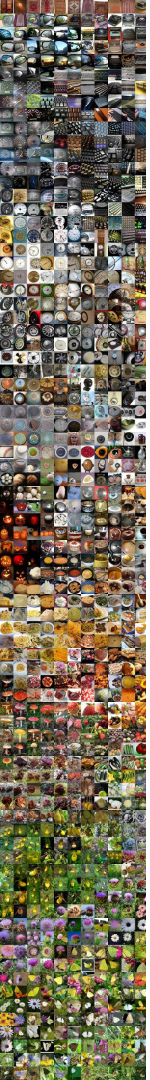
- ❏ Connect
- ❏ Partition Data
- ❏ Apply Transform
- ❏ Load Libraries
- ❏ Set Configuration
- ❏ Define Strategy
- ❏ Define Model
- ❏ Compile Model
- ❏ Fit Model
- ❏ Retrieve Model

https://github.com/mlverse/distributed

Example (1/2)

```
library(sparklyr)
sc <- spark_connect(master = "yarn", spark_home = "/usr/lib/spark/", config =
list(spark.dynamicAllocation.enabled= FALSE, `sparklyr.shell.executor-cores` = 8,
`sparklyr.shell.num-executors` = 3, sparklyr.apply.env.WORKON_HOME= "/tmp/.virtualenvs"))

sdf_len(sc, 3, repartition = 3) %>%
  spark_apply(function(df, barrier) {
    tryCatch({
      library(tensorflow)
      library(keras)

      Sys.setenv(TF_CONFIG = jsonlite::toJSON(list(
        cluster = list(worker = paste(gsub(":[0-9]+$", "", barrier$address), 8000 +
seq_along(barrier$address), sep = ":")),
        task = list(type = 'worker', index = barrier$partition)
      ), auto_unbox = TRUE))

      if (is.null(tf_version())) install_tensorflow()

      // << tensorflow model >>

    }, error = function(e) e$message)
  }, barrier = TRUE, columns = c(address = "character"), ) %>% collect()
```

https://github.com/mlverse/distributed

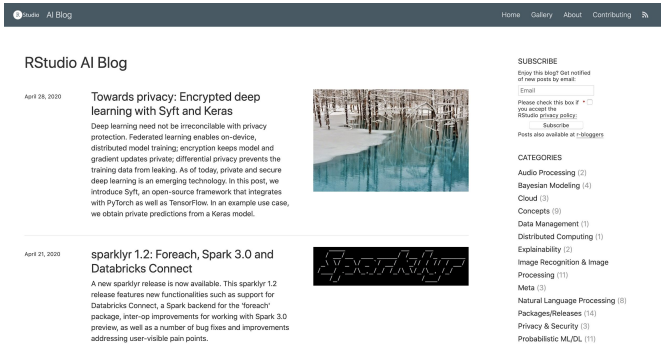


Example (1/2)

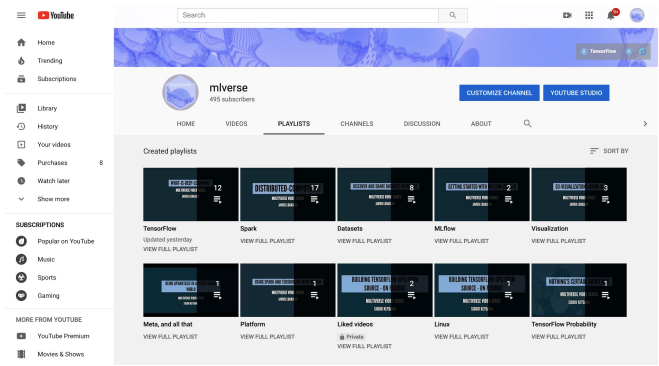
```
strategy <- tf$distribute$experimental$MultiWorkerMirroredStrategy()
mnist <- dataset_mnist()

x_train <- mnist$train$x
y_train <- mnist$train$y
x_train <- array_reshape(x_train, c(nrow(x_train),28, 28, 1))
x_train <- x_train / 255

with (strategy$scope(), {
   model <- keras_model_sequential() %>%
     layer_conv_2d(filters = 32, kernel_size = 3, activation = 'relu', input_shape = c(28, 28, 1)) %>%
     layer_max_pooling_2d() %>%
     layer_flatten() %>%
     layer_dense(units = 64, activation = 'relu') %>%
     layer_dense(units = 10)

   model %>% compile(
     loss = tf$keras$losses$SparseCategoricalCrossentropy(from_logits = TRUE),
     optimizer = tf$keras$optimizers$SGD(learning_rate = 0.001),
     metrics = 'accuracy') })

model %>% fit(x_train, y_train, batch_size = 64L * 3L, epochs = 3, steps_per_epoch = 5)
```

https://github.com/mlverse/distributed

Thanks!



blogs.rstudio.com/ai



youtube.com/c/mlverse

@zkajdan

@javierluraschi

@dfalbel

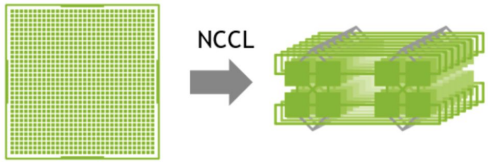
@yl790

NVIDIA NCCL

NVIDIA NCCL

The NVIDIA Collective Communications Library (NCCL) implements multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs. NCCL provides routines such as all-gather, all-reduce, broadcast, reduce, reduce-scatter, that are optimized to achieve high bandwidth and low latency over PCIe and NVLink high-speed interconnect.

Get Started



Performance

NCCL conveniently removes the need for developers to optimize their applications for specific machines. NCCL provides fast collectives over multiple GPUs both within and across nodes.

Ease of Programming

NCCL uses a simple C API, which can be easily accessed from a variety of programming languages.NCCL closely follows the popular collectives API defined by MPI (Message Passing Interface).

Compatibility

NCCL is compatible with virtually any multi-GPU parallelization model, such as: single-threaded, multi-threaded (using one thread per GPU) and multi-process (MPI combined with multi-threaded operation on GPUs).

https://developer.nvidia.com/nccl

Infiniband

		Single Data Rate SDR	Double DDR	Quadruple QDR	FDR10	Fourteenth FDR	Enhanced EDR	HDR	NDR	XDR
Signaling rate (Gbit/s)		2.5	5	10	10.3125	14.0625[6]	25.78125	50	100	250
Theoretical effective throughput (Gb/s)[7]	**for 1 link**	2	4	8	10	13.64	25	50	100	250
	for 4 links	8	16	32	40	54.54	100	200	400	1000
	for 8 links	16	32	64	80	109.08	200	400	800	2000
	for 12 links	24	48	96	120	163.64	300	600	1200	3000
Encoding (bits)		8b/10b			64b/66b				t.b.d.	t.b.d.
Adapter latency (μs)[8]		5	2.5	1.3	0.7	0.7	0.5	less?	t.b.d.	t.b.d.
Year[9]		2001, 2003	2005	2007	2011	2011	2014[7]	2017[7]	after 2020	after 2023?

InfiniBand also provides RDMA capabilities for low CPU overhead.

https://en.wikipedia.org/wiki/InfiniBand