Module: tf.compat.v1.distribute

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute#top_of_page)
* [Modules](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute#modules)
* [Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute#classes)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute#functions)

Library for running a computation across multiple devices.

See the guide for overview and examples: [TensorFlow v1.x](https://www.tensorflow.org/guide/distribute_strategy), [TensorFlow v2.x](https://www.tensorflow.org/alpha/guide/distribute_strategy).

The intent of this library is that you can write an algorithm in a stylized way and it will be usable with a variety of different [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementations. Each descendant will implement a different strategy for distributing the algorithm across multiple devices/machines. Furthermore, these changes can be hidden inside the specific layers and other library classes that need special treatment to run in a distributed setting, so that most users' model definition code can run unchanged. The [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) API works the same way with eager and graph execution.

*Glossary*

* *Data parallelism* is where we run multiple copies of the model on different slices of the input data. This is in contrast to *model parallelism* where we divide up a single copy of a model across multiple devices. Note: we only support data parallelism for now, but hope to add support for model parallelism in the future.
* A *device* is a CPU or accelerator (e.g. GPUs, TPUs) on some machine that TensorFlow can run operations on (see e.g. [tf.device](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/device)). You may have multiple devices on a single machine, or be connected to devices on multiple machines. Devices used to run computations are called *worker devices*. Devices used to store variables are *parameter devices*. For some strategies, such as [tf.distribute.MirroredStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/MirroredStrategy), the worker and parameter devices will be the same (see mirrored variables below). For others they will be different. For example, [tf.distribute.experimental.CentralStorageStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/CentralStorageStrategy) puts the variables on a single device (which may be a worker device or may be the CPU), and [tf.distribute.experimental.ParameterServerStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/ParameterServerStrategy) puts the variables on separate machines called parameter servers (see below).
* A *replica* is one copy of the model, running on one slice of the input data. Right now each replica is executed on its own worker device, but once we add support for model parallelism a replica may span multiple worker devices.
* A *host* is the CPU device on a machine with worker devices, typically used for running input pipelines.
* A *worker* is defined to be the physical machine(s) containing the physical devices (e.g. GPUs, TPUs) on which the replicated computation is executed. A worker may contain one or more replicas, but contains at least one replica. Typically one worker will correspond to one machine, but in the case of very large models with model parallelism, one worker may span multiple machines. We typically run one input pipeline per worker, feeding all the replicas on that worker.
* *Synchronous*, or more commonly *sync*, training is where the updates from each replica are aggregated together before updating the model variables. This is in contrast to *asynchronous*, or *async* training, where each replica updates the model variables independently. You may also have replicas partitioned into gropus which are in sync within each group but async between groups.
* *Parameter servers*: These are machines that hold a single copy of parameters/variables, used by some strategies (right now just [tf.distribute.experimental.ParameterServerStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/ParameterServerStrategy)). All replicas that want to operate on a variable retrieve it at the beginning of a step and send an update to be applied at the end of the step. These can in priniciple support either sync or async training, but right now we only have support for async training with parameter servers. Compare to [tf.distribute.experimental.CentralStorageStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/CentralStorageStrategy), which puts all variables on a single device on the same machine (and does sync training), and[tf.distribute.MirroredStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/MirroredStrategy), which mirrors variables to multiple devices (see below).
* *Mirrored variables*: These are variables that are copied to multiple devices, where we keep the copies in sync by applying the same updates to every copy. Normally would only be used with sync training.
* Reductions and all-reduce: A *reduction* is some method of aggregating multiple values into one value, like "sum" or "mean". If a strategy is doing sync training, we will perform a reduction on the gradients to a parameter from all replicas before applying the update. *All-reduce* is an algorithm for performing a reduction on values from multiple devices and making the result available on all of those devices.

Note that we provide a default version of [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) that is used when no other strategy is in scope, that provides the same API with reasonable default behavior.

Modules

[cluster\_resolver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/cluster_resolver) module: Library Imports for Cluster Resolvers.

[experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental) module: Experimental Distribution Strategy library.

Classes

[class CrossDeviceOps](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/CrossDeviceOps): Base class for cross-device reduction and broadcasting algorithms.

[class HierarchicalCopyAllReduce](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/HierarchicalCopyAllReduce): Reduction using hierarchical copy all-reduce.

[class InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext): A class wrapping information needed by an input function.

[class InputReplicationMode](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputReplicationMode): Replication mode for input function.

[class MirroredStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/MirroredStrategy): Mirrors vars to distribute across multiple devices and machines.

[class NcclAllReduce](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/NcclAllReduce): Reduction using NCCL all-reduce.

[class OneDeviceStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/OneDeviceStrategy): A distribution strategy for running on a single device.

[class ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp): Indicates how a set of values should be reduced.

[class ReductionToOneDevice](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReductionToOneDevice): Always do reduction to one device first and then do broadcasting.

[class ReplicaContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReplicaContext): [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) API when in a replica context.

[class Server](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Server): An in-process TensorFlow server, for use in distributed training.

[class Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy): A list of devices with a state & compute distribution policy.

[class StrategyExtended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/StrategyExtended): Additional APIs for algorithms that need to be distribution-aware.

Functions

[experimental\_set\_strategy(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental_set_strategy): Set a [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) as current without with strategy.scope().

[get\_loss\_reduction(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/get_loss_reduction): [tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) corresponding to the last loss reduction.

[get\_replica\_context(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/get_replica_context): Returns the current [tf.distribute.ReplicaContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReplicaContext) or None.

[get\_strategy(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/get_strategy): Returns the current [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) object.

[has\_strategy(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/has_strategy): Return if there is a current non-default [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy).

[in\_cross\_replica\_context(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/in_cross_replica_context): Returns True if in a cross-replica context.

# tf.compat.v1.distribute.get\_loss\_reduction

[tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) corresponding to the last loss reduction.

tf.compat.v1.distribute.get\_loss\_reduction()

Defined in [python/distribute/distribute\_lib.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/distribute/distribute_lib.py).

This is used to decide whether loss should be scaled in optimizer (used only for estimator + v1 optimizer use case).

#### Returns:

[tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) corresponding to the last loss reduction for estimator and v1 optimizer use case. [tf.distribute.ReduceOp.SUM](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp#SUM) otherwise.

# tf.compat.v1.distribute.MirroredStrategy

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/MirroredStrategy#top_of_page)
* [Class MirroredStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/MirroredStrategy#class_mirroredstrategy)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/MirroredStrategy#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/MirroredStrategy#properties)
  + [extended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/MirroredStrategy#extended)

## Class MirroredStrategy

Mirrors vars to distribute across multiple devices and machines.

Inherits From: [Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy)

Defined in [python/distribute/mirrored\_strategy.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/distribute/mirrored_strategy.py).

This strategy uses one replica per device and sync replication for its multi-GPU version.

The multi-worker version will be added in the future.

#### Args:

* **devices**: a list of device strings. If None, all available GPUs are used. If no GPUs are found, CPU is used.
* **cross\_device\_ops**: optional, a descedant of CrossDeviceOps. If this is not set, nccl will be used by default.

## \_\_init\_\_

\_\_init\_\_(  
    devices=None,  
    cross\_device\_ops=None  
)

## Properties

### extended

[tf.distribute.StrategyExtended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended) with additional methods.

### num\_replicas\_in\_sync

Returns number of replicas over which gradients are aggregated.

## Methods

### experimental\_distribute\_dataset

experimental\_distribute\_dataset(dataset)

Distributes a tf.data.Dataset instance provided via dataset.

In a multi-worker setting, we will first attempt to distribute the dataset by attempting to detect whether the dataset is being created out of ReaderDatasets (e.g. TFRecordDataset, TextLineDataset, etc.) and if so, attempting to shard the input files. Note that there has to be at least one input file per worker. If you have less than one input file per worker, we suggest that you should disable distributing your dataset using the method below.

If that attempt is unsuccessful (e.g. the dataset is created from a Dataset.range), we will shard the dataset evenly at the end by appending a .shard operation to the end of the processing pipeline. This will cause the entire preprocessing pipeline for all the data to be run on every worker, and each worker will do redundant work. We will print a warning if this method of sharding is selected. In this case, consider using experimental\_distribute\_datasets\_from\_function instead.

You can disable dataset distribution using the auto\_shard option in[tf.data.experimental.DistributeOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/experimental/DistributeOptions).

Within each host, we will also split the data among all the worker devices (if more than one a present), and this will happen even if multi-worker sharding is disabled using the method above.

The following is an example:

strategy = tf.distribute.MirroredStrategy()  
  
# Create a dataset  
dataset = dataset\_ops.Dataset.TFRecordDataset([  
  "/a/1.tfr", "/a/2.tfr", "/a/3.tfr", /a/4.tfr"])  
  
# Distribute that dataset  
dist\_dataset = strategy.experimental\_distribute\_dataset(dataset)  
# Iterate over the distributed dataset  
for x in dist\_dataset:  
  # process dataset elements  
  strategy.experimental\_run\_v2(train\_step, args=(x,))

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be sharded across all replicas using the rules stated above.

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_distribute\_datasets\_from\_function

experimental\_distribute\_datasets\_from\_function(dataset\_fn)

Distributes [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) instances created by calls to dataset\_fn.

dataset\_fn will be called once for each worker in the strategy. Each replica on that worker will dequeue one batch of inputs from the local Dataset (i.e. if a worker has two replicas, two batches will be dequeued from the Dataset every step).

This method can be used for several purposes. For example, whereexperimental\_distribute\_dataset is unable to shard the input files, this method might be used to manually shard the dataset (avoiding the slow fallback behavior in experimental\_distribute\_dataset). In cases where the dataset is infinite, this sharding can be done by creating dataset replicas that differ only in their random seed.

The dataset\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance where information about batching and input replication can be accessed:

def dataset\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(  
      input\_context.num\_input\_pipelines, input\_context.input\_pipeline\_id)  
  
inputs = strategy.experimental\_distribute\_datasets\_from\_function(dataset\_fn)  
  
for batch in inputs:  
  replica\_results = strategy.experimental\_run\_v2(replica\_fn, args=(batch,))

IMPORTANT: The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by dataset\_fn should have a per-replica batch size, unlike experimental\_distribute\_dataset, which uses the global batch size. This may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **dataset\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_local\_results

experimental\_local\_results(value)

Returns the list of all local per-replica values contained in value.

**Note:** This only returns values on the workers initiated by this client. When using a [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy)like [**tf.distribute.experimental.MultiWorkerMirroredStrategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/MultiWorkerMirroredStrategy), each worker will be its own client, and this function will only return values computed on that worker.

#### Args:

* **value**: A value returned by experimental\_run(), experimental\_run\_v2(),extended.call\_for\_each\_replica(), or a variable created in scope.

#### Returns:

A tuple of values contained in value. If value represents a single value, this returns (value,).

### experimental\_make\_numpy\_dataset

experimental\_make\_numpy\_dataset(  
    numpy\_input,  
    session=None  
)

Makes a dataset for input provided via a numpy array.

This avoids adding numpy\_input as a large constant in the graph, and copies the data to the machine or machines that will be processing the input.

#### Args:

* **numpy\_input**: A nest of NumPy input arrays that will be distributed evenly across all replicas. Note that lists of Numpy arrays are stacked, as that is normal [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) behavior.
* **session**: (TensorFlow v1.x graph execution only) A session used for initialization.

#### Returns:

A [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) representing numpy\_input.

### experimental\_run

experimental\_run(  
    fn,  
    input\_iterator=None  
)

Runs ops in fn on each replica, with inputs from input\_iterator.

DEPRECATED: This method is not available in TF 2.x. Please switch to using experimental\_run\_v2instead.

When eager execution is enabled, executes ops specified by fn on each replica. Otherwise, builds a graph to execute the ops on each replica.

Each replica will take a single, different input from the inputs provided by one get\_next call on the input iterator.

fn may call tf.distribute.get\_replica\_context() to access members such as replica\_id\_in\_sync\_group.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The inputs to the function must match the outputs of input\_iterator.get\_next(). The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **input\_iterator**: (Optional) input iterator from which the inputs are taken.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be PerReplica (if the values are unsynchronized), Mirrored (if the values are kept in sync), or Tensor (if running on a single replica).

### experimental\_run\_v2

experimental\_run\_v2(  
    fn,  
    args=(),  
    kwargs=None  
)

Runs ops in fn on each replica, with the given arguments.

Executes ops specified by fn on each replica. If args or kwargs have "per-replica" values, such as those produced by a "distributed Dataset", when fn is executed on a particular replica, it will be executed with the component of those "per-replica" values that corresponds to that replica.

fn may call tf.distribute.get\_replica\_context() to access members such as all\_reduce.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **args**: (Optional) Positional arguments to fn.
* **kwargs**: (Optional) Keyword arguments to fn.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be "per-replica" Tensor objects or Tensors (for example, if running on a single replica).

### make\_dataset\_iterator

make\_dataset\_iterator(dataset)

Makes an iterator for input provided via dataset.

DEPRECATED: This method is not available in TF 2.x.

Data from the given dataset will be distributed evenly across all the compute replicas. We will assume that the input dataset is batched by the global batch size. With this assumption, we will make a best effort to divide each batch across all the replicas (one or more workers). If this effort fails, an error will be thrown, and the user should instead use make\_input\_fn\_iterator which provides more control to the user, and does not try to divide a batch across replicas.

The user could also use make\_input\_fn\_iterator if they want to customize which input is fed to which replica/worker etc.

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be distributed evenly across all replicas.

#### Returns:

An tf.distribute.InputIterator which returns inputs for each step of the computation. User should call initialize on the returned iterator.

### make\_input\_fn\_iterator

make\_input\_fn\_iterator(  
    input\_fn,  
    replication\_mode=tf.distribute.InputReplicationMode.PER\_WORKER  
)

Returns an iterator split across replicas created from an input function.

DEPRECATED: This method is not available in TF 2.x.

The input\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object where information about batching and input sharding can be accessed:

def input\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(input\_context.num\_input\_pipelines,  
                 input\_context.input\_pipeline\_id)  
with strategy.scope():  
  iterator = strategy.make\_input\_fn\_iterator(input\_fn)  
  replica\_results = strategy.experimental\_run(replica\_fn, iterator)

The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by input\_fn should have a per-replica batch size, which may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **input\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).
* **replication\_mode**: an enum value of [tf.distribute.InputReplicationMode](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputReplicationMode). Only PER\_WORKER is supported currently, which means there will be a single call to input\_fn per worker. Replicas will dequeue from the local [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) on their worker.

#### Returns:

An iterator object that should first be .initialize()-ed. It may then either be passed to strategy.experimental\_run() or you can iterator.get\_next() to get the next value to pass tostrategy.extended.call\_for\_each\_replica().

### reduce

reduce(  
    reduce\_op,  
    value,  
    axis=None  
)

Reduce value across replicas.

Given a per-replica value returned by experimental\_run\_v2, say a per-example loss, the batch will be divided across all the replicas. This function allows you to aggregate across replicas and optionally also across batch elements. For example, if you have a global batch size of 8 and 2 replicas, values for examples [0, 1, 2, 3] will be on replica 0 and [4, 5, 6, 7] will be on replica 1. By default, reduce will just aggregate across replicas, returning [0+4, 1+5, 2+6, 3+7]. This is useful when each replica is computing a scalar or some other value that doesn't have a "batch" dimension (like a gradient). More often you will want to aggregate across the global batch, which you can get by specifying the batch dimension as the axis, typically axis=0. In this case it would return a scalar 0+1+2+3+4+5+6+7.

If there is a last partial batch, you will need to specify an axis so that the resulting shape is consistent across replicas. So if the last batch has size 6 and it is divided into [0, 1, 2, 3] and [4, 5], you would get a shape mismatch unless you specify axis=0. If you specify [tf.distribute.ReduceOp.MEAN](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp#MEAN), using axis=0 will use the correct denominator of 6. Contrast this with computing reduce\_mean to get a scalar value on each replica and this function to average those means, which will weigh some values 1/8 and others 1/4.

#### Args:

* **reduce\_op**: A [tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) value specifying how values should be combined.
* **value**: A "per replica" value, e.g. returned by experimental\_run\_v2 to be combined into a single tensor.
* **axis**: Specifies the dimension to reduce along within each replica's tensor. Should typically be set to the batch dimension, or None to only reduce across replicas (e.g. if the tensor has no batch dimension).

#### Returns:

A Tensor.

### scope

scope()

Returns a context manager selecting this Strategy as current.

Inside a with strategy.scope(): code block, this thread will use a variable creator set by strategy, and will enter its "cross-replica context".

#### Returns:

A context manager.

### update\_config\_proto

update\_config\_proto(config\_proto)

Returns a copy of config\_proto modified for use with this strategy.

DEPRECATED: This method is not available in TF 2.x.

The updated config has something needed to run a strategy, e.g. configuration to run collective ops, or device filters to improve distributed training performance.

#### Args:

* **config\_proto**: a tf.ConfigProto object.

#### Returns:

The updated copy of the config\_proto.

# tf.compat.v1.distribute.OneDeviceStrategy

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/OneDeviceStrategy#top_of_page)
* [Class OneDeviceStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/OneDeviceStrategy#class_onedevicestrategy)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/OneDeviceStrategy#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/OneDeviceStrategy#properties)
  + [extended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/OneDeviceStrategy#extended)

## Class OneDeviceStrategy

A distribution strategy for running on a single device.

Inherits From: [Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy)

Defined in [python/distribute/one\_device\_strategy.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/distribute/one_device_strategy.py).

## \_\_init\_\_

\_\_init\_\_(device)

## Properties

### extended

[tf.distribute.StrategyExtended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended) with additional methods.

### num\_replicas\_in\_sync

Returns number of replicas over which gradients are aggregated.

## Methods

### experimental\_distribute\_dataset

experimental\_distribute\_dataset(dataset)

Distributes a tf.data.Dataset instance provided via dataset.

In a multi-worker setting, we will first attempt to distribute the dataset by attempting to detect whether the dataset is being created out of ReaderDatasets (e.g. TFRecordDataset, TextLineDataset, etc.) and if so, attempting to shard the input files. Note that there has to be at least one input file per worker. If you have less than one input file per worker, we suggest that you should disable distributing your dataset using the method below.

If that attempt is unsuccessful (e.g. the dataset is created from a Dataset.range), we will shard the dataset evenly at the end by appending a .shard operation to the end of the processing pipeline. This will cause the entire preprocessing pipeline for all the data to be run on every worker, and each worker will do redundant work. We will print a warning if this method of sharding is selected. In this case, consider using experimental\_distribute\_datasets\_from\_function instead.

You can disable dataset distribution using the auto\_shard option in[tf.data.experimental.DistributeOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/experimental/DistributeOptions).

Within each host, we will also split the data among all the worker devices (if more than one a present), and this will happen even if multi-worker sharding is disabled using the method above.

The following is an example:

strategy = tf.distribute.MirroredStrategy()  
  
# Create a dataset  
dataset = dataset\_ops.Dataset.TFRecordDataset([  
  "/a/1.tfr", "/a/2.tfr", "/a/3.tfr", /a/4.tfr"])  
  
# Distribute that dataset  
dist\_dataset = strategy.experimental\_distribute\_dataset(dataset)  
# Iterate over the distributed dataset  
for x in dist\_dataset:  
  # process dataset elements  
  strategy.experimental\_run\_v2(train\_step, args=(x,))

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be sharded across all replicas using the rules stated above.

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_distribute\_datasets\_from\_function

experimental\_distribute\_datasets\_from\_function(dataset\_fn)

Distributes [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) instances created by calls to dataset\_fn.

dataset\_fn will be called once for each worker in the strategy. Each replica on that worker will dequeue one batch of inputs from the local Dataset (i.e. if a worker has two replicas, two batches will be dequeued from the Dataset every step).

This method can be used for several purposes. For example, whereexperimental\_distribute\_dataset is unable to shard the input files, this method might be used to manually shard the dataset (avoiding the slow fallback behavior in experimental\_distribute\_dataset). In cases where the dataset is infinite, this sharding can be done by creating dataset replicas that differ only in their random seed.

The dataset\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance where information about batching and input replication can be accessed:

def dataset\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(  
      input\_context.num\_input\_pipelines, input\_context.input\_pipeline\_id)  
  
inputs = strategy.experimental\_distribute\_datasets\_from\_function(dataset\_fn)  
  
for batch in inputs:  
  replica\_results = strategy.experimental\_run\_v2(replica\_fn, args=(batch,))

IMPORTANT: The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by dataset\_fn should have a per-replica batch size, unlike experimental\_distribute\_dataset, which uses the global batch size. This may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **dataset\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_local\_results

experimental\_local\_results(value)

Returns the list of all local per-replica values contained in value.

**Note:** This only returns values on the workers initiated by this client. When using a [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy)like [**tf.distribute.experimental.MultiWorkerMirroredStrategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/MultiWorkerMirroredStrategy), each worker will be its own client, and this function will only return values computed on that worker.

#### Args:

* **value**: A value returned by experimental\_run(), experimental\_run\_v2(),extended.call\_for\_each\_replica(), or a variable created in scope.

#### Returns:

A tuple of values contained in value. If value represents a single value, this returns (value,).

### experimental\_make\_numpy\_dataset

experimental\_make\_numpy\_dataset(  
    numpy\_input,  
    session=None  
)

Makes a dataset for input provided via a numpy array.

This avoids adding numpy\_input as a large constant in the graph, and copies the data to the machine or machines that will be processing the input.

#### Args:

* **numpy\_input**: A nest of NumPy input arrays that will be distributed evenly across all replicas. Note that lists of Numpy arrays are stacked, as that is normal [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) behavior.
* **session**: (TensorFlow v1.x graph execution only) A session used for initialization.

#### Returns:

A [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) representing numpy\_input.

### experimental\_run

experimental\_run(  
    fn,  
    input\_iterator=None  
)

Runs ops in fn on each replica, with inputs from input\_iterator.

DEPRECATED: This method is not available in TF 2.x. Please switch to using experimental\_run\_v2instead.

When eager execution is enabled, executes ops specified by fn on each replica. Otherwise, builds a graph to execute the ops on each replica.

Each replica will take a single, different input from the inputs provided by one get\_next call on the input iterator.

fn may call tf.distribute.get\_replica\_context() to access members such as replica\_id\_in\_sync\_group.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The inputs to the function must match the outputs of input\_iterator.get\_next(). The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **input\_iterator**: (Optional) input iterator from which the inputs are taken.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be PerReplica (if the values are unsynchronized), Mirrored (if the values are kept in sync), or Tensor (if running on a single replica).

### experimental\_run\_v2

experimental\_run\_v2(  
    fn,  
    args=(),  
    kwargs=None  
)

Runs ops in fn on each replica, with the given arguments.

Executes ops specified by fn on each replica. If args or kwargs have "per-replica" values, such as those produced by a "distributed Dataset", when fn is executed on a particular replica, it will be executed with the component of those "per-replica" values that corresponds to that replica.

fn may call tf.distribute.get\_replica\_context() to access members such as all\_reduce.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **args**: (Optional) Positional arguments to fn.
* **kwargs**: (Optional) Keyword arguments to fn.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be "per-replica" Tensor objects or Tensors (for example, if running on a single replica).

### make\_dataset\_iterator

make\_dataset\_iterator(dataset)

Makes an iterator for input provided via dataset.

DEPRECATED: This method is not available in TF 2.x.

Data from the given dataset will be distributed evenly across all the compute replicas. We will assume that the input dataset is batched by the global batch size. With this assumption, we will make a best effort to divide each batch across all the replicas (one or more workers). If this effort fails, an error will be thrown, and the user should instead use make\_input\_fn\_iterator which provides more control to the user, and does not try to divide a batch across replicas.

The user could also use make\_input\_fn\_iterator if they want to customize which input is fed to which replica/worker etc.

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be distributed evenly across all replicas.

#### Returns:

An tf.distribute.InputIterator which returns inputs for each step of the computation. User should call initialize on the returned iterator.

### make\_input\_fn\_iterator

make\_input\_fn\_iterator(  
    input\_fn,  
    replication\_mode=tf.distribute.InputReplicationMode.PER\_WORKER  
)

Returns an iterator split across replicas created from an input function.

DEPRECATED: This method is not available in TF 2.x.

The input\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object where information about batching and input sharding can be accessed:

def input\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(input\_context.num\_input\_pipelines,  
                 input\_context.input\_pipeline\_id)  
with strategy.scope():  
  iterator = strategy.make\_input\_fn\_iterator(input\_fn)  
  replica\_results = strategy.experimental\_run(replica\_fn, iterator)

The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by input\_fn should have a per-replica batch size, which may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **input\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).
* **replication\_mode**: an enum value of [tf.distribute.InputReplicationMode](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputReplicationMode). Only PER\_WORKER is supported currently, which means there will be a single call to input\_fn per worker. Replicas will dequeue from the local [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) on their worker.

#### Returns:

An iterator object that should first be .initialize()-ed. It may then either be passed to strategy.experimental\_run() or you can iterator.get\_next() to get the next value to pass tostrategy.extended.call\_for\_each\_replica().

### reduce

reduce(  
    reduce\_op,  
    value,  
    axis=None  
)

Reduce value across replicas.

Given a per-replica value returned by experimental\_run\_v2, say a per-example loss, the batch will be divided across all the replicas. This function allows you to aggregate across replicas and optionally also across batch elements. For example, if you have a global batch size of 8 and 2 replicas, values for examples [0, 1, 2, 3] will be on replica 0 and [4, 5, 6, 7] will be on replica 1. By default, reduce will just aggregate across replicas, returning [0+4, 1+5, 2+6, 3+7]. This is useful when each replica is computing a scalar or some other value that doesn't have a "batch" dimension (like a gradient). More often you will want to aggregate across the global batch, which you can get by specifying the batch dimension as the axis, typically axis=0. In this case it would return a scalar 0+1+2+3+4+5+6+7.

If there is a last partial batch, you will need to specify an axis so that the resulting shape is consistent across replicas. So if the last batch has size 6 and it is divided into [0, 1, 2, 3] and [4, 5], you would get a shape mismatch unless you specify axis=0. If you specify [tf.distribute.ReduceOp.MEAN](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp#MEAN), using axis=0 will use the correct denominator of 6. Contrast this with computing reduce\_mean to get a scalar value on each replica and this function to average those means, which will weigh some values 1/8 and others 1/4.

#### Args:

* **reduce\_op**: A [tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) value specifying how values should be combined.
* **value**: A "per replica" value, e.g. returned by experimental\_run\_v2 to be combined into a single tensor.
* **axis**: Specifies the dimension to reduce along within each replica's tensor. Should typically be set to the batch dimension, or None to only reduce across replicas (e.g. if the tensor has no batch dimension).

#### Returns:

A Tensor.

### scope

scope()

Returns a context manager selecting this Strategy as current.

Inside a with strategy.scope(): code block, this thread will use a variable creator set by strategy, and will enter its "cross-replica context".

#### Returns:

A context manager.

### update\_config\_proto

update\_config\_proto(config\_proto)

Returns a copy of config\_proto modified for use with this strategy.

DEPRECATED: This method is not available in TF 2.x.

The updated config has something needed to run a strategy, e.g. configuration to run collective ops, or device filters to improve distributed training performance.

#### Args:

* **config\_proto**: a tf.ConfigProto object.

#### Returns:

The updated copy of the config\_proto.

# tf.compat.v1.distribute.Strategy

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy#top_of_page)
* [Class Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy#class_strategy)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy#properties)
  + [extended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy#extended)

## Class Strategy

A list of devices with a state & compute distribution policy.

Inherits From: [Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy)

Defined in [python/distribute/distribute\_lib.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/distribute/distribute_lib.py).

See [the guide](https://www.tensorflow.org/guide/distribute_strategy) for overview and examples.

**Note:** Not all [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementations currently support TensorFlow's partitioned variables (where a single variable is split across multiple devices) at this time.

## \_\_init\_\_

\_\_init\_\_(extended)

## Properties

### extended

[tf.distribute.StrategyExtended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended) with additional methods.

### num\_replicas\_in\_sync

Returns number of replicas over which gradients are aggregated.

## Methods

### experimental\_distribute\_dataset

experimental\_distribute\_dataset(dataset)

Distributes a tf.data.Dataset instance provided via dataset.

In a multi-worker setting, we will first attempt to distribute the dataset by attempting to detect whether the dataset is being created out of ReaderDatasets (e.g. TFRecordDataset, TextLineDataset, etc.) and if so, attempting to shard the input files. Note that there has to be at least one input file per worker. If you have less than one input file per worker, we suggest that you should disable distributing your dataset using the method below.

If that attempt is unsuccessful (e.g. the dataset is created from a Dataset.range), we will shard the dataset evenly at the end by appending a .shard operation to the end of the processing pipeline. This will cause the entire preprocessing pipeline for all the data to be run on every worker, and each worker will do redundant work. We will print a warning if this method of sharding is selected. In this case, consider using experimental\_distribute\_datasets\_from\_function instead.

You can disable dataset distribution using the auto\_shard option in[tf.data.experimental.DistributeOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/experimental/DistributeOptions).

Within each host, we will also split the data among all the worker devices (if more than one a present), and this will happen even if multi-worker sharding is disabled using the method above.

The following is an example:

strategy = tf.distribute.MirroredStrategy()  
  
# Create a dataset  
dataset = dataset\_ops.Dataset.TFRecordDataset([  
  "/a/1.tfr", "/a/2.tfr", "/a/3.tfr", /a/4.tfr"])  
  
# Distribute that dataset  
dist\_dataset = strategy.experimental\_distribute\_dataset(dataset)  
# Iterate over the distributed dataset  
for x in dist\_dataset:  
  # process dataset elements  
  strategy.experimental\_run\_v2(train\_step, args=(x,))

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be sharded across all replicas using the rules stated above.

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_distribute\_datasets\_from\_function

experimental\_distribute\_datasets\_from\_function(dataset\_fn)

Distributes [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) instances created by calls to dataset\_fn.

dataset\_fn will be called once for each worker in the strategy. Each replica on that worker will dequeue one batch of inputs from the local Dataset (i.e. if a worker has two replicas, two batches will be dequeued from the Dataset every step).

This method can be used for several purposes. For example, whereexperimental\_distribute\_dataset is unable to shard the input files, this method might be used to manually shard the dataset (avoiding the slow fallback behavior in experimental\_distribute\_dataset). In cases where the dataset is infinite, this sharding can be done by creating dataset replicas that differ only in their random seed.

The dataset\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance where information about batching and input replication can be accessed:

def dataset\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(  
      input\_context.num\_input\_pipelines, input\_context.input\_pipeline\_id)  
  
inputs = strategy.experimental\_distribute\_datasets\_from\_function(dataset\_fn)  
  
for batch in inputs:  
  replica\_results = strategy.experimental\_run\_v2(replica\_fn, args=(batch,))

IMPORTANT: The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by dataset\_fn should have a per-replica batch size, unlike experimental\_distribute\_dataset, which uses the global batch size. This may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **dataset\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_local\_results

experimental\_local\_results(value)

Returns the list of all local per-replica values contained in value.

**Note:** This only returns values on the workers initiated by this client. When using a [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy)like [**tf.distribute.experimental.MultiWorkerMirroredStrategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/MultiWorkerMirroredStrategy), each worker will be its own client, and this function will only return values computed on that worker.

#### Args:

* **value**: A value returned by experimental\_run(), experimental\_run\_v2(),extended.call\_for\_each\_replica(), or a variable created in scope.

#### Returns:

A tuple of values contained in value. If value represents a single value, this returns (value,).

### experimental\_make\_numpy\_dataset

experimental\_make\_numpy\_dataset(  
    numpy\_input,  
    session=None  
)

Makes a dataset for input provided via a numpy array.

This avoids adding numpy\_input as a large constant in the graph, and copies the data to the machine or machines that will be processing the input.

#### Args:

* **numpy\_input**: A nest of NumPy input arrays that will be distributed evenly across all replicas. Note that lists of Numpy arrays are stacked, as that is normal [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) behavior.
* **session**: (TensorFlow v1.x graph execution only) A session used for initialization.

#### Returns:

A [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) representing numpy\_input.

### experimental\_run

experimental\_run(  
    fn,  
    input\_iterator=None  
)

Runs ops in fn on each replica, with inputs from input\_iterator.

DEPRECATED: This method is not available in TF 2.x. Please switch to using experimental\_run\_v2instead.

When eager execution is enabled, executes ops specified by fn on each replica. Otherwise, builds a graph to execute the ops on each replica.

Each replica will take a single, different input from the inputs provided by one get\_next call on the input iterator.

fn may call tf.distribute.get\_replica\_context() to access members such as replica\_id\_in\_sync\_group.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The inputs to the function must match the outputs of input\_iterator.get\_next(). The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **input\_iterator**: (Optional) input iterator from which the inputs are taken.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be PerReplica (if the values are unsynchronized), Mirrored (if the values are kept in sync), or Tensor (if running on a single replica).

### experimental\_run\_v2

experimental\_run\_v2(  
    fn,  
    args=(),  
    kwargs=None  
)

Runs ops in fn on each replica, with the given arguments.

Executes ops specified by fn on each replica. If args or kwargs have "per-replica" values, such as those produced by a "distributed Dataset", when fn is executed on a particular replica, it will be executed with the component of those "per-replica" values that corresponds to that replica.

fn may call tf.distribute.get\_replica\_context() to access members such as all\_reduce.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **args**: (Optional) Positional arguments to fn.
* **kwargs**: (Optional) Keyword arguments to fn.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be "per-replica" Tensor objects or Tensors (for example, if running on a single replica).

### make\_dataset\_iterator

make\_dataset\_iterator(dataset)

Makes an iterator for input provided via dataset.

DEPRECATED: This method is not available in TF 2.x.

Data from the given dataset will be distributed evenly across all the compute replicas. We will assume that the input dataset is batched by the global batch size. With this assumption, we will make a best effort to divide each batch across all the replicas (one or more workers). If this effort fails, an error will be thrown, and the user should instead use make\_input\_fn\_iterator which provides more control to the user, and does not try to divide a batch across replicas.

The user could also use make\_input\_fn\_iterator if they want to customize which input is fed to which replica/worker etc.

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be distributed evenly across all replicas.

#### Returns:

An tf.distribute.InputIterator which returns inputs for each step of the computation. User should call initialize on the returned iterator.

### make\_input\_fn\_iterator

make\_input\_fn\_iterator(  
    input\_fn,  
    replication\_mode=tf.distribute.InputReplicationMode.PER\_WORKER  
)

Returns an iterator split across replicas created from an input function.

DEPRECATED: This method is not available in TF 2.x.

The input\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object where information about batching and input sharding can be accessed:

def input\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(input\_context.num\_input\_pipelines,  
                 input\_context.input\_pipeline\_id)  
with strategy.scope():  
  iterator = strategy.make\_input\_fn\_iterator(input\_fn)  
  replica\_results = strategy.experimental\_run(replica\_fn, iterator)

The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by input\_fn should have a per-replica batch size, which may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **input\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).
* **replication\_mode**: an enum value of [tf.distribute.InputReplicationMode](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputReplicationMode). Only PER\_WORKER is supported currently, which means there will be a single call to input\_fn per worker. Replicas will dequeue from the local [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) on their worker.

#### Returns:

An iterator object that should first be .initialize()-ed. It may then either be passed to strategy.experimental\_run() or you can iterator.get\_next() to get the next value to pass tostrategy.extended.call\_for\_each\_replica().

### reduce

reduce(  
    reduce\_op,  
    value,  
    axis=None  
)

Reduce value across replicas.

Given a per-replica value returned by experimental\_run\_v2, say a per-example loss, the batch will be divided across all the replicas. This function allows you to aggregate across replicas and optionally also across batch elements. For example, if you have a global batch size of 8 and 2 replicas, values for examples [0, 1, 2, 3] will be on replica 0 and [4, 5, 6, 7] will be on replica 1. By default, reduce will just aggregate across replicas, returning [0+4, 1+5, 2+6, 3+7]. This is useful when each replica is computing a scalar or some other value that doesn't have a "batch" dimension (like a gradient). More often you will want to aggregate across the global batch, which you can get by specifying the batch dimension as the axis, typically axis=0. In this case it would return a scalar 0+1+2+3+4+5+6+7.

If there is a last partial batch, you will need to specify an axis so that the resulting shape is consistent across replicas. So if the last batch has size 6 and it is divided into [0, 1, 2, 3] and [4, 5], you would get a shape mismatch unless you specify axis=0. If you specify [tf.distribute.ReduceOp.MEAN](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp#MEAN), using axis=0 will use the correct denominator of 6. Contrast this with computing reduce\_mean to get a scalar value on each replica and this function to average those means, which will weigh some values 1/8 and others 1/4.

#### Args:

* **reduce\_op**: A [tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) value specifying how values should be combined.
* **value**: A "per replica" value, e.g. returned by experimental\_run\_v2 to be combined into a single tensor.
* **axis**: Specifies the dimension to reduce along within each replica's tensor. Should typically be set to the batch dimension, or None to only reduce across replicas (e.g. if the tensor has no batch dimension).

#### Returns:

A Tensor.

### scope

scope()

Returns a context manager selecting this Strategy as current.

Inside a with strategy.scope(): code block, this thread will use a variable creator set by strategy, and will enter its "cross-replica context".

#### Returns:

A context manager.

### update\_config\_proto

update\_config\_proto(config\_proto)

Returns a copy of config\_proto modified for use with this strategy.

DEPRECATED: This method is not available in TF 2.x.

The updated config has something needed to run a strategy, e.g. configuration to run collective ops, or device filters to improve distributed training performance.

#### Args:

* **config\_proto**: a tf.ConfigProto object.

#### Returns:

The updated copy of the config\_proto.

# tf.compat.v1.distribute.StrategyExtended

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/StrategyExtended#top_of_page)
* [Class StrategyExtended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/StrategyExtended#class_strategyextended)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/StrategyExtended#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/StrategyExtended#properties)
  + [experimental\_between\_graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/StrategyExtended#experimental_between_graph)

## Class StrategyExtended

Additional APIs for algorithms that need to be distribution-aware.

Inherits From: [StrategyExtended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended)

Defined in [python/distribute/distribute\_lib.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/distribute/distribute_lib.py).

**Note:** For most usage of [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy), there should be no need to call these methods, since TensorFlow libraries (such as optimizers) already call these methods when needed on your behalf.

Lower-level concepts:

* Wrapped values: In order to represent values parallel across devices (either replicas or the devices associated with a particular value), we wrap them in a "PerReplica" or "Mirrored" object that contains a map from replica id to values. "PerReplica" is used when the value may be different across replicas, and "Mirrored" when the value are the same.
* Unwrapping and merging: Consider calling a function fn on multiple replicas, like experimental\_run\_v2(fn, args=[w]) with an argument w that is a wrapped value. This means w will have a map taking replica id 0 to w0, replica id 11 to w1, etc.experimental\_run\_v2() unwraps w before calling fn, so it calls fn(w0) on d0, fn(w1) on d1, etc. It then merges the return values from fn(), which can possibly result in wrapped values. For example, let's say fn() returns a tuple with three components: (x, a, v0) from replica 0, (x, b, v1) on replica 1, etc. If the first component is the same object x from every replica, then the first component of the merged result will also be x. If the second component is different (a, b, ...) from each replica, then the merged value will have a wrapped map from replica device to the different values. If the third component is the members of a mirrored variable (v maps d0 to v0, d1 to v1, etc.), then the merged result will be that mirrored variable (v).
* Worker devices vs. parameter devices: Most replica computations will happen on worker devices. Since we don't yet support model parallelism, there will be one worker device per replica. When using parameter servers or central storage, the set of devices holding variables may be different, otherwise the parameter devices might match the worker devices.

Replica context vs. Cross-replica context

replica context is when we are in some function that is being called once for each replica. Otherwise we are in cross-replica context, which is useful for calling [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) methods which operate across the replicas (like reduce\_to()). By default you start in a replica context (the "default single replica context") and then some methods can switch you back and forth. There is a third mode you can be in called update context used when updating variables.

* [tf.distribute.Strategy.scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy#scope): enters cross-replica context when no other strategy is in scope.
* [tf.distribute.Strategy.experimental\_run\_v2](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy#experimental_run_v2): calls a function in replica context.
* [tf.distribute.ReplicaContext.merge\_call](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReplicaContext#merge_call): transitions from replica context to cross-replica context.
* [tf.distribute.StrategyExtended.update](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended#update): calls a function in an update context from a cross-replica context.

In a replica context, you may freely read the values of variables, but you may only update their value if they specify a way to aggregate the update using the aggregation parameter in the variable's constructor. In a cross-replica context, you may read or write variables (writes may need to be broadcast to all copies of the variable if it is mirrored).

Sync on read variables

In some cases, such as a metric, we want to accumulate a bunch of updates on each replica independently and only aggregate when reading. This can be a big performance win when the value is read only rarely (maybe the value is only read at the end of an epoch or when checkpointing). These are variables created by passing synchronization=ON\_READ to the variable's constructor (and some value for aggregation).

The strategy may choose to put the variable on multiple devices, like mirrored variables, but unlike mirrored variables we don't synchronize the updates to them to make sure they have the same value. Instead, the synchronization is performed when reading in cross-replica context. In a replica context, reads and writes are performed on the local copy (we allow reads so you can write code like v = 0.9\*v + 0.1\*update). We don't allow operations like v.assign\_add in a cross-replica context for sync on read variables; right now we don't have a use case for such updates and depending on the aggregation mode such updates may not be sensible.

Locality

Depending on how a value is produced, it will have a type that will determine how it may be used.

"Per-replica" values exist on the worker devices, with a different value for each replica. They are produced by iterating through a "distributed Dataset" returned by [tf.distribute.Strategy.experimental\_distribute\_dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy#experimental_distribute_dataset) and[tf.distribute.Strategy.experimental\_distribute\_datasets\_from\_function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy#experimental_distribute_datasets_from_function). They are also the typical result returned by [tf.distribute.Strategy.experimental\_run\_v2](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy#experimental_run_v2). You typically can't use a per-replica value directly in a cross-replica context, without first resolving how to aggregate the values across replicas, for instance by using [tf.distribute.Strategy.reduce](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy#reduce).

"Mirrored" values are like per-replica values, except we know that the value on all replicas are the same. We can safely read a mirrored value in a cross-replica context by using the value on any replica. You can convert a per-replica value into a mirrored value by using[tf.distribute.ReplicaContext.all\_reduce](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReplicaContext#all_reduce).

Values can also have the same locality as a variable, which is a mirrored value but residing on the same devices as the variable (as opposed to the compute devices). Such values may be passed to a call to [tf.distribute.StrategyExtended.update](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended#update) to update the value of a variable. You may use [tf.distribute.StrategyExtended.colocate\_vars\_with](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended#colocate_vars_with) to give a variable the same locality as another variable. This is useful, for example, for "slot" variables used by an optimizer for keeping track of statistics used to update a primary/model variable. You may convert a per-replica value to a variable's locality by using [tf.distribute.StrategyExtended.reduce\_to](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended#reduce_to) or[tf.distribute.StrategyExtended.batch\_reduce\_to](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended#batch_reduce_to).

In addition to slot variables which should be colocated with their primary variables, optimizers also define non-slot variables. These can be things like "number of step updates performed" or "beta1^t" and "beta2^t". Each strategy has some policy for which devices those variables should be copied too, called the "non-slot devices" (some subset of the parameter devices). We require that all non-slot variables are allocated on the same device, or mirrored across the same set of devices. You can use[tf.distribute.StrategyExtended.non\_slot\_devices](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended#non_slot_devices) to pick a consistent set of devices to pass to both [tf.distribute.StrategyExtended.colocate\_vars\_with](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended#colocate_vars_with) and [tf.distribute.StrategyExtended.update\_non\_slot](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended#update_non_slot).

How to update a variable

The standard pattern for updating variables is to:

1. In your function passed to [tf.distribute.Strategy.experimental\_run\_v2](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy#experimental_run_v2), compute a list of (update, variable) pairs. For example, the update might be a the gradient of the loss with respect to the variable.
2. Switch to cross-replica mode by callingtf.distribute.get\_replica\_context().merge\_call() with the updates and variables as arguments.
3. Call tf.distribute.StrategyExtended.reduce\_to(VariableAggregation.SUM, t, v) (for one variable) or [tf.distribute.StrategyExtended.batch\_reduce\_to](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended#batch_reduce_to) (for a list of variables) to sum the updates. and broadcast the result to the variable's devices.
4. Call tf.distribute.StrategyExtended.update(v) for each variable to update its value.

Steps 2 through 4 are done automatically by class [tf.keras.optimizers.Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/optimizers/Optimizer) if you call its[tf.keras.optimizers.Optimizer.apply\_gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/optimizers/Optimizer#apply_gradients) method in a replica context. They are also done automatically if you call an assign\* method on a (non sync-on-read) variable that was constructed with an aggregation method (which is used to determine the reduction used in step 3).

Distribute-aware layers

Layers are generally called in a replica context, except when defining a functional model. [tf.distribute.in\_cross\_replica\_context](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/in_cross_replica_context) will let you determine which case you are in. If in a replica context, the [tf.distribute.get\_replica\_context](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/get_replica_context) function will return a[tf.distribute.ReplicaContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReplicaContext) object. The ReplicaContext object has an all\_reduce method for aggregating across all replicas. Alternatively, you can update variables following steps 2-4 above.

**Note:** For new [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementations, please put all logic in a subclass of [**tf.distribute.StrategyExtended**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended). The only code needed for the [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) subclass is for instantiating your subclass of [**tf.distribute.StrategyExtended**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended) in the **\_\_init\_\_** method.

## \_\_init\_\_

\_\_init\_\_(container\_strategy)

## Properties

### experimental\_between\_graph

Whether the strategy uses between-graph replication or not.

This is expected to return a constant value that will not be changed throughout its life cycle.

### experimental\_require\_static\_shapes

### experimental\_should\_init

Whether initialization is needed.

### parameter\_devices

Returns the tuple of all devices used to place variables.

### should\_checkpoint

Whether checkpointing is needed.

### should\_save\_summary

Whether saving summaries is needed.

### worker\_devices

Returns the tuple of all devices used to for compute replica execution.

## Methods

### batch\_reduce\_to

batch\_reduce\_to(  
    reduce\_op,  
    value\_destination\_pairs  
)

Combine multiple reduce\_to calls into one for faster execution.

#### Args:

* **reduce\_op**: Reduction type, an instance of [tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) enum.
* **value\_destination\_pairs**: A sequence of (value, destinations) pairs. See reduce\_to() for a description.

#### Returns:

A list of mirrored values, one per pair in value\_destination\_pairs.

### broadcast\_to

broadcast\_to(  
    tensor,  
    destinations  
)

Mirror a tensor on one device to all worker devices.

#### Args:

* **tensor**: A Tensor value to broadcast.
* **destinations**: A mirrored variable or device string specifying the destination devices to copy tensor to.

#### Returns:

A value mirrored to destinations devices.

### call\_for\_each\_replica

call\_for\_each\_replica(  
    fn,  
    args=(),  
    kwargs=None  
)

Run fn once per replica.

fn may call tf.get\_replica\_context() to access methods such as replica\_id\_in\_sync\_groupand merge\_call().

merge\_call() is used to communicate between the replicas and re-enter the cross-replica context. All replicas pause their execution having encountered a merge\_call() call. After that the merge\_fn-function is executed. Its results are then unwrapped and given back to each replica call. After that execution resumes until fn is complete or encounters another merge\_call(). Example:

# Called once in "cross-replica" context.  
def merge\_fn(distribution, three\_plus\_replica\_id):  
  # sum the values across replicas  
  return sum(distribution.experimental\_local\_results(three\_plus\_replica\_id))  
  
# Called once per replica in `distribution`, in a "replica" context.  
def fn(three):  
  replica\_ctx = tf.get\_replica\_context()  
  v = three + replica\_ctx.replica\_id\_in\_sync\_group  
  # Computes the sum of the `v` values across all replicas.  
  s = replica\_ctx.merge\_call(merge\_fn, args=(v,))  
  return s + v  
  
with distribution.scope():  
  # in "cross-replica" context  
  ...  
  merged\_results = distribution.experimental\_run\_v2(fn, args=[3])  
  # merged\_results has the values from every replica execution of `fn`.  
  # This statement prints a list:  
  print(distribution.experimental\_local\_results(merged\_results))

#### Args:

* **fn**: function to run (will be run once per replica).
* **args**: Tuple or list with positional arguments for fn.
* **kwargs**: Dict with keyword arguments for fn.

#### Returns:

Merged return value of fn across all replicas.

### colocate\_vars\_with

colocate\_vars\_with(colocate\_with\_variable)

Scope that controls which devices variables will be created on.

No operations should be added to the graph inside this scope, it should only be used when creating variables (some implementations work by changing variable creation, others work by using a tf.compat.v1.colocate\_with() scope).

This may only be used inside self.scope().

#### Example usage:

with strategy.scope():  
  var1 = tf.Variable(...)  
  with strategy.extended.colocate\_vars\_with(var1):  
    # var2 and var3 will be created on the same device(s) as var1  
    var2 = tf.Variable(...)  
    var3 = tf.Variable(...)  
  
  def fn(v1, v2, v3):  
    # operates on v1 from var1, v2 from var2, and v3 from var3  
  
  # `fn` runs on every device `var1` is on, `var2` and `var3` will be there  
  # too.  
  strategy.extended.update(var1, fn, args=(var2, var3))

#### Args:

* **colocate\_with\_variable**: A variable created in this strategy's scope(). Variables created while in the returned context manager will be on the same set of devices as colocate\_with\_variable.

#### Returns:

A context manager.

### experimental\_make\_numpy\_dataset

experimental\_make\_numpy\_dataset(  
    numpy\_input,  
    session=None  
)

Makes a dataset for input provided via a numpy array.

This avoids adding numpy\_input as a large constant in the graph, and copies the data to the machine or machines that will be processing the input.

#### Args:

* **numpy\_input**: A nest of NumPy input arrays that will be distributed evenly across all replicas. Note that lists of Numpy arrays are stacked, as that is normal [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) behavior.
* **session**: (TensorFlow v1.x graph execution only) A session used for initialization.

#### Returns:

A [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) representing numpy\_input.

### experimental\_run\_steps\_on\_iterator

experimental\_run\_steps\_on\_iterator(  
    fn,  
    iterator,  
    iterations=1,  
    initial\_loop\_values=None  
)

Run fn with input from iterator for iterations times.

This method can be used to run a step function for training a number of times using input from a dataset.

#### Args:

* **fn**: function to run using this distribution strategy. The function must have the following signature: def fn(context, inputs). context is an instance of MultiStepContext that will be passed when fn is run. context can be used to specify the outputs to be returned from fn by calling context.set\_last\_step\_output. It can also be used to capture non tensor outputs by context.set\_non\_tensor\_output. See MultiStepContext documentation for more information. inputs will have same type/structure as iterator.get\_next(). Typically, fn will use call\_for\_each\_replica method of the strategy to distribute the computation over multiple replicas.
* **iterator**: Iterator of a dataset that represents the input for fn. The caller is responsible for initializing the iterator as needed.
* **iterations**: (Optional) Number of iterations that fn should be run. Defaults to 1.
* **initial\_loop\_values**: (Optional) Initial values to be passed into the loop that runs fn. Defaults to None. # TODO(priyag): Remove initial\_loop\_values argument when we have a mechanism to infer the outputs of fn.

#### Returns:

Returns the MultiStepContext object which has the following properties, among other things: - run\_op: An op that runs fn iterations times. - last\_step\_outputs: A dictionary containing tensors set using context.set\_last\_step\_output. Evaluating this returns the value of the tensors after the last iteration. - non\_tensor\_outputs: A dictionatry containing anything that was set by fn by calling context.set\_non\_tensor\_output.

### non\_slot\_devices

non\_slot\_devices(var\_list)

Device(s) for non-slot variables.

Create variables on these devices in a with colocate\_vars\_with(non\_slot\_devices(...)):block. Update those using update\_non\_slot().

#### Args:

* **var\_list**: The list of variables being optimized, needed with the default [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy).

### read\_var

read\_var(v)

Reads the value of a variable.

Returns the aggregate value of a replica-local variable, or the (read-only) value of any other variable.

#### Args:

* **v**: A variable allocated within the scope of this [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy).

#### Returns:

A tensor representing the value of v, aggregated across replicas if necessary.

### reduce\_to

reduce\_to(  
    reduce\_op,  
    value,  
    destinations  
)

Combine (via e.g. sum or mean) values across replicas.

#### Args:

* **reduce\_op**: Reduction type, an instance of [tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) enum.
* **value**: A per-replica value with one value per replica.
* **destinations**: A mirrored variable, a per-replica tensor, or a device string. The return value will be copied to all destination devices (or all the devices where the destinations value resides). To perform an all-reduction, pass value to destinations.

#### Returns:

A value mirrored to destinations.

### update

update(  
    var,  
    fn,  
    args=(),  
    kwargs=None,  
    group=True  
)

Run fn to update var using inputs mirrored to the same devices.

If var is mirrored across multiple devices, then this implements logic like:

results = {}  
for device, v in var:  
  with tf.device(device):  
    # args and kwargs will be unwrapped if they are mirrored.  
    results[device] = fn(v, \*args, \*\*kwargs)  
return merged(results)

Otherwise this returns fn(var, \*args, \*\*kwargs) colocated with var.

Neither args nor kwargs may contain per-replica values. If they contain mirrored values, they will be unwrapped before calling fn.

#### Args:

* **var**: Variable, possibly mirrored to multiple devices, to operate on.
* **fn**: Function to call. Should take the variable as the first argument.
* **args**: Tuple or list. Additional positional arguments to pass to fn().
* **kwargs**: Dict with keyword arguments to pass to fn().
* **group**: Boolean. Defaults to True. If False, the return value will be unwrapped.

#### Returns:

By default, the merged return value of fn across all replicas. The merged result has dependencies to make sure that if it is evaluated at all, the side effects (updates) will happen on every replica. If instead "group=False" is specified, this function will return a nest of lists where each list has an element per replica, and the caller is responsible for ensuring all elements are executed.

### update\_non\_slot

update\_non\_slot(  
    colocate\_with,  
    fn,  
    args=(),  
    kwargs=None,  
    group=True  
)

Runs fn(\*args, \*\*kwargs) on colocate\_with devices.

#### Args:

* **colocate\_with**: The return value of non\_slot\_devices().
* **fn**: Function to execute.
* **args**: Tuple or list. Positional arguments to pass to fn().
* **kwargs**: Dict with keyword arguments to pass to fn().
* **group**: Boolean. Defaults to True. If False, the return value will be unwrapped.

#### Returns:

Return value of fn, possibly merged across devices.

### value\_container

value\_container(value)

Returns the container that this per-replica value belongs to.

#### Args:

* **value**: A value returned by experimental\_run\_v2() or a variable created in scope().

#### Returns:

A container that value belongs to. If value does not belong to any container (including the case of container having been destroyed), returns the value itself. value in experimental\_local\_results(value\_container(value)) will always be true.

### variable\_created\_in\_scope

variable\_created\_in\_scope(v)

Tests whether v was created while this strategy scope was active.

Variables created inside the strategy scope are "owned" by it:

with strategy.scope(): ... v = tf.Variable(1.) strategy.variable\_created\_in\_scope(v) True

Variables created outside the strategy are not owned by it:

v = tf.Variable(1.) strategy.variable\_created\_in\_scope(v) False

#### Args:

* **v**: A [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) instance.

#### Returns:

True if v was created inside the scope, False if not.

Module: tf.compat.v1.distribute.cluster\_resolver

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/cluster_resolver#top_of_page)
* [Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/cluster_resolver#classes)

Library Imports for Cluster Resolvers.

Classes

[class ClusterResolver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/cluster_resolver/ClusterResolver): Abstract class for all implementations of ClusterResolvers.

[class GCEClusterResolver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/cluster_resolver/GCEClusterResolver): Cluster Resolver for Google Compute Engine.

[class KubernetesClusterResolver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/cluster_resolver/KubernetesClusterResolver): Cluster Resolver for Kubernetes.

[class SimpleClusterResolver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/cluster_resolver/SimpleClusterResolver): Simple implementation of ClusterResolver that accepts a ClusterSpec.

[class SlurmClusterResolver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/cluster_resolver/SlurmClusterResolver): Cluster Resolver for system with Slurm workload manager.

[class TFConfigClusterResolver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/cluster_resolver/TFConfigClusterResolver): Implementation of a ClusterResolver which reads the TF\_CONFIG EnvVar.

[class TPUClusterResolver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/cluster_resolver/TPUClusterResolver): Cluster Resolver for Google Cloud TPUs.

[class UnionResolver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/cluster_resolver/UnionResolver): Performs a union on underlying ClusterResolvers.

Module: tf.compat.v1.distribute.experimental

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental#top_of_page)
* [Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental#classes)

Experimental Distribution Strategy library.

Classes

[class CentralStorageStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/CentralStorageStrategy): A one-machine strategy that puts all variables on a single device.

[class CollectiveCommunication](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/CollectiveCommunication): Communication choices for CollectiveOps.

[class MultiWorkerMirroredStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/MultiWorkerMirroredStrategy): Distribution strategy that uses collective ops for all-reduce.

[class ParameterServerStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/ParameterServerStrategy): An asynchronous multi-worker parameter server DistributionStrategy.

[class TPUStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/TPUStrategy): TPU distribution strategy implementation.

# tf.compat.v1.distribute.experimental.CentralStorageStrategy

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/CentralStorageStrategy#top_of_page)
* [Class CentralStorageStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/CentralStorageStrategy#class_centralstoragestrategy)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/CentralStorageStrategy#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/CentralStorageStrategy#properties)
  + [extended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/CentralStorageStrategy#extended)

## Class CentralStorageStrategy

A one-machine strategy that puts all variables on a single device.

Inherits From: [Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy)

Defined in [python/distribute/central\_storage\_strategy.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/distribute/central_storage_strategy.py).

Variables are assigned to local CPU or the only GPU. If there is more than one GPU, compute operations (other than variable update operations) will be replicated across all GPUs.

#### Args:

* **compute\_devices**: an optional list of strings for device to replicate models on. If this is not provided, all local GPUs will be used; if there is no GPU, local CPU will be used.
* **parameter\_device**: an optional device string for which device to put variables on. The default one is CPU or GPU if there is only one.

## \_\_init\_\_

\_\_init\_\_(  
    compute\_devices=None,  
    parameter\_device=None  
)

Initializes this strategy with default TFConfigClusterResolver.

## Properties

### extended

[tf.distribute.StrategyExtended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended) with additional methods.

### num\_replicas\_in\_sync

Returns number of replicas over which gradients are aggregated.

## Methods

### experimental\_distribute\_dataset

experimental\_distribute\_dataset(dataset)

Distributes a tf.data.Dataset instance provided via dataset.

In a multi-worker setting, we will first attempt to distribute the dataset by attempting to detect whether the dataset is being created out of ReaderDatasets (e.g. TFRecordDataset, TextLineDataset, etc.) and if so, attempting to shard the input files. Note that there has to be at least one input file per worker. If you have less than one input file per worker, we suggest that you should disable distributing your dataset using the method below.

If that attempt is unsuccessful (e.g. the dataset is created from a Dataset.range), we will shard the dataset evenly at the end by appending a .shard operation to the end of the processing pipeline. This will cause the entire preprocessing pipeline for all the data to be run on every worker, and each worker will do redundant work. We will print a warning if this method of sharding is selected. In this case, consider using experimental\_distribute\_datasets\_from\_function instead.

You can disable dataset distribution using the auto\_shard option in[tf.data.experimental.DistributeOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/experimental/DistributeOptions).

Within each host, we will also split the data among all the worker devices (if more than one a present), and this will happen even if multi-worker sharding is disabled using the method above.

The following is an example:

strategy = tf.distribute.MirroredStrategy()  
  
# Create a dataset  
dataset = dataset\_ops.Dataset.TFRecordDataset([  
  "/a/1.tfr", "/a/2.tfr", "/a/3.tfr", /a/4.tfr"])  
  
# Distribute that dataset  
dist\_dataset = strategy.experimental\_distribute\_dataset(dataset)  
# Iterate over the distributed dataset  
for x in dist\_dataset:  
  # process dataset elements  
  strategy.experimental\_run\_v2(train\_step, args=(x,))

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be sharded across all replicas using the rules stated above.

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_distribute\_datasets\_from\_function

experimental\_distribute\_datasets\_from\_function(dataset\_fn)

Distributes [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) instances created by calls to dataset\_fn.

dataset\_fn will be called once for each worker in the strategy. Each replica on that worker will dequeue one batch of inputs from the local Dataset (i.e. if a worker has two replicas, two batches will be dequeued from the Dataset every step).

This method can be used for several purposes. For example, whereexperimental\_distribute\_dataset is unable to shard the input files, this method might be used to manually shard the dataset (avoiding the slow fallback behavior in experimental\_distribute\_dataset). In cases where the dataset is infinite, this sharding can be done by creating dataset replicas that differ only in their random seed.

The dataset\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance where information about batching and input replication can be accessed:

def dataset\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(  
      input\_context.num\_input\_pipelines, input\_context.input\_pipeline\_id)  
  
inputs = strategy.experimental\_distribute\_datasets\_from\_function(dataset\_fn)  
  
for batch in inputs:  
  replica\_results = strategy.experimental\_run\_v2(replica\_fn, args=(batch,))

IMPORTANT: The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by dataset\_fn should have a per-replica batch size, unlike experimental\_distribute\_dataset, which uses the global batch size. This may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **dataset\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_local\_results

experimental\_local\_results(value)

Returns the list of all local per-replica values contained in value.

**Note:** This only returns values on the workers initiated by this client. When using a [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy)like [**tf.distribute.experimental.MultiWorkerMirroredStrategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/MultiWorkerMirroredStrategy), each worker will be its own client, and this function will only return values computed on that worker.

#### Args:

* **value**: A value returned by experimental\_run(), experimental\_run\_v2(),extended.call\_for\_each\_replica(), or a variable created in scope.

#### Returns:

A tuple of values contained in value. If value represents a single value, this returns (value,).

### experimental\_make\_numpy\_dataset

experimental\_make\_numpy\_dataset(  
    numpy\_input,  
    session=None  
)

Makes a dataset for input provided via a numpy array.

This avoids adding numpy\_input as a large constant in the graph, and copies the data to the machine or machines that will be processing the input.

#### Args:

* **numpy\_input**: A nest of NumPy input arrays that will be distributed evenly across all replicas. Note that lists of Numpy arrays are stacked, as that is normal [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) behavior.
* **session**: (TensorFlow v1.x graph execution only) A session used for initialization.

#### Returns:

A [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) representing numpy\_input.

### experimental\_run

experimental\_run(  
    fn,  
    input\_iterator=None  
)

Runs ops in fn on each replica, with inputs from input\_iterator.

DEPRECATED: This method is not available in TF 2.x. Please switch to using experimental\_run\_v2instead.

When eager execution is enabled, executes ops specified by fn on each replica. Otherwise, builds a graph to execute the ops on each replica.

Each replica will take a single, different input from the inputs provided by one get\_next call on the input iterator.

fn may call tf.distribute.get\_replica\_context() to access members such as replica\_id\_in\_sync\_group.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The inputs to the function must match the outputs of input\_iterator.get\_next(). The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **input\_iterator**: (Optional) input iterator from which the inputs are taken.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be PerReplica (if the values are unsynchronized), Mirrored (if the values are kept in sync), or Tensor (if running on a single replica).

### experimental\_run\_v2

experimental\_run\_v2(  
    fn,  
    args=(),  
    kwargs=None  
)

Runs ops in fn on each replica, with the given arguments.

Executes ops specified by fn on each replica. If args or kwargs have "per-replica" values, such as those produced by a "distributed Dataset", when fn is executed on a particular replica, it will be executed with the component of those "per-replica" values that corresponds to that replica.

fn may call tf.distribute.get\_replica\_context() to access members such as all\_reduce.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **args**: (Optional) Positional arguments to fn.
* **kwargs**: (Optional) Keyword arguments to fn.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be "per-replica" Tensor objects or Tensors (for example, if running on a single replica).

### make\_dataset\_iterator

make\_dataset\_iterator(dataset)

Makes an iterator for input provided via dataset.

DEPRECATED: This method is not available in TF 2.x.

Data from the given dataset will be distributed evenly across all the compute replicas. We will assume that the input dataset is batched by the global batch size. With this assumption, we will make a best effort to divide each batch across all the replicas (one or more workers). If this effort fails, an error will be thrown, and the user should instead use make\_input\_fn\_iterator which provides more control to the user, and does not try to divide a batch across replicas.

The user could also use make\_input\_fn\_iterator if they want to customize which input is fed to which replica/worker etc.

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be distributed evenly across all replicas.

#### Returns:

An tf.distribute.InputIterator which returns inputs for each step of the computation. User should call initialize on the returned iterator.

### make\_input\_fn\_iterator

make\_input\_fn\_iterator(  
    input\_fn,  
    replication\_mode=tf.distribute.InputReplicationMode.PER\_WORKER  
)

Returns an iterator split across replicas created from an input function.

DEPRECATED: This method is not available in TF 2.x.

The input\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object where information about batching and input sharding can be accessed:

def input\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(input\_context.num\_input\_pipelines,  
                 input\_context.input\_pipeline\_id)  
with strategy.scope():  
  iterator = strategy.make\_input\_fn\_iterator(input\_fn)  
  replica\_results = strategy.experimental\_run(replica\_fn, iterator)

The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by input\_fn should have a per-replica batch size, which may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **input\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).
* **replication\_mode**: an enum value of [tf.distribute.InputReplicationMode](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputReplicationMode). Only PER\_WORKER is supported currently, which means there will be a single call to input\_fn per worker. Replicas will dequeue from the local [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) on their worker.

#### Returns:

An iterator object that should first be .initialize()-ed. It may then either be passed to strategy.experimental\_run() or you can iterator.get\_next() to get the next value to pass tostrategy.extended.call\_for\_each\_replica().

### reduce

reduce(  
    reduce\_op,  
    value,  
    axis=None  
)

Reduce value across replicas.

Given a per-replica value returned by experimental\_run\_v2, say a per-example loss, the batch will be divided across all the replicas. This function allows you to aggregate across replicas and optionally also across batch elements. For example, if you have a global batch size of 8 and 2 replicas, values for examples [0, 1, 2, 3] will be on replica 0 and [4, 5, 6, 7] will be on replica 1. By default, reduce will just aggregate across replicas, returning [0+4, 1+5, 2+6, 3+7]. This is useful when each replica is computing a scalar or some other value that doesn't have a "batch" dimension (like a gradient). More often you will want to aggregate across the global batch, which you can get by specifying the batch dimension as the axis, typically axis=0. In this case it would return a scalar 0+1+2+3+4+5+6+7.

If there is a last partial batch, you will need to specify an axis so that the resulting shape is consistent across replicas. So if the last batch has size 6 and it is divided into [0, 1, 2, 3] and [4, 5], you would get a shape mismatch unless you specify axis=0. If you specify [tf.distribute.ReduceOp.MEAN](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp#MEAN), using axis=0 will use the correct denominator of 6. Contrast this with computing reduce\_mean to get a scalar value on each replica and this function to average those means, which will weigh some values 1/8 and others 1/4.

#### Args:

* **reduce\_op**: A [tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) value specifying how values should be combined.
* **value**: A "per replica" value, e.g. returned by experimental\_run\_v2 to be combined into a single tensor.
* **axis**: Specifies the dimension to reduce along within each replica's tensor. Should typically be set to the batch dimension, or None to only reduce across replicas (e.g. if the tensor has no batch dimension).

#### Returns:

A Tensor.

### scope

scope()

Returns a context manager selecting this Strategy as current.

Inside a with strategy.scope(): code block, this thread will use a variable creator set by strategy, and will enter its "cross-replica context".

#### Returns:

A context manager.

### update\_config\_proto

update\_config\_proto(config\_proto)

Returns a copy of config\_proto modified for use with this strategy.

DEPRECATED: This method is not available in TF 2.x.

The updated config has something needed to run a strategy, e.g. configuration to run collective ops, or device filters to improve distributed training performance.

#### Args:

* **config\_proto**: a tf.ConfigProto object.

#### Returns:

The updated copy of the config\_proto.

# tf.compat.v1.distribute.experimental.MultiWorkerMirroredStrategy

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/MultiWorkerMirroredStrategy#top_of_page)
* [Class MultiWorkerMirroredStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/MultiWorkerMirroredStrategy#class_multiworkermirroredstrategy)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/MultiWorkerMirroredStrategy#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/MultiWorkerMirroredStrategy#properties)
  + [extended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/MultiWorkerMirroredStrategy#extended)

## Class MultiWorkerMirroredStrategy

Distribution strategy that uses collective ops for all-reduce.

Inherits From: [Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy)

Defined in [python/distribute/collective\_all\_reduce\_strategy.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/distribute/collective_all_reduce_strategy.py).

It is similar to MirroredStrategy but it uses collective ops for reduction.

By default it uses all local GPUs or CPU for single-worker training.

When 'TF\_CONFIG' environment variable is given, it parses cluster\_spec, task\_type and task\_id from 'TF\_CONFIG' and turns into a multi-worker strategy which mirrores models on GPUs of all machines in a cluster. In the current implementation, it uses all GPUs in a cluster and it assumes all workers have the same number of GPUs.

It supports both eager mode and graph mode. However, for eager mode, it has to set up the eager context in its constructor and therefore all ops in eager mode have to run after the strategy object is created.

#### Args:

* **communication**: optional Enum of typedistribute.experimental.CollectiveCommunication. This provides a way for the user to override the choice of collective op communication. Possible values include AUTO, RING, and NCCL.

## \_\_init\_\_

\_\_init\_\_(communication=tf.distribute.experimental.CollectiveCommunication.AUTO)

Initializes the object.

## Properties

### extended

[tf.distribute.StrategyExtended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended) with additional methods.

### num\_replicas\_in\_sync

Returns number of replicas over which gradients are aggregated.

## Methods

### experimental\_distribute\_dataset

experimental\_distribute\_dataset(dataset)

Distributes a tf.data.Dataset instance provided via dataset.

In a multi-worker setting, we will first attempt to distribute the dataset by attempting to detect whether the dataset is being created out of ReaderDatasets (e.g. TFRecordDataset, TextLineDataset, etc.) and if so, attempting to shard the input files. Note that there has to be at least one input file per worker. If you have less than one input file per worker, we suggest that you should disable distributing your dataset using the method below.

If that attempt is unsuccessful (e.g. the dataset is created from a Dataset.range), we will shard the dataset evenly at the end by appending a .shard operation to the end of the processing pipeline. This will cause the entire preprocessing pipeline for all the data to be run on every worker, and each worker will do redundant work. We will print a warning if this method of sharding is selected. In this case, consider using experimental\_distribute\_datasets\_from\_function instead.

You can disable dataset distribution using the auto\_shard option in[tf.data.experimental.DistributeOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/experimental/DistributeOptions).

Within each host, we will also split the data among all the worker devices (if more than one a present), and this will happen even if multi-worker sharding is disabled using the method above.

The following is an example:

strategy = tf.distribute.MirroredStrategy()  
  
# Create a dataset  
dataset = dataset\_ops.Dataset.TFRecordDataset([  
  "/a/1.tfr", "/a/2.tfr", "/a/3.tfr", /a/4.tfr"])  
  
# Distribute that dataset  
dist\_dataset = strategy.experimental\_distribute\_dataset(dataset)  
# Iterate over the distributed dataset  
for x in dist\_dataset:  
  # process dataset elements  
  strategy.experimental\_run\_v2(train\_step, args=(x,))

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be sharded across all replicas using the rules stated above.

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_distribute\_datasets\_from\_function

experimental\_distribute\_datasets\_from\_function(dataset\_fn)

Distributes [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) instances created by calls to dataset\_fn.

dataset\_fn will be called once for each worker in the strategy. Each replica on that worker will dequeue one batch of inputs from the local Dataset (i.e. if a worker has two replicas, two batches will be dequeued from the Dataset every step).

This method can be used for several purposes. For example, whereexperimental\_distribute\_dataset is unable to shard the input files, this method might be used to manually shard the dataset (avoiding the slow fallback behavior in experimental\_distribute\_dataset). In cases where the dataset is infinite, this sharding can be done by creating dataset replicas that differ only in their random seed.

The dataset\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance where information about batching and input replication can be accessed:

def dataset\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(  
      input\_context.num\_input\_pipelines, input\_context.input\_pipeline\_id)  
  
inputs = strategy.experimental\_distribute\_datasets\_from\_function(dataset\_fn)  
  
for batch in inputs:  
  replica\_results = strategy.experimental\_run\_v2(replica\_fn, args=(batch,))

IMPORTANT: The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by dataset\_fn should have a per-replica batch size, unlike experimental\_distribute\_dataset, which uses the global batch size. This may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **dataset\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_local\_results

experimental\_local\_results(value)

Returns the list of all local per-replica values contained in value.

**Note:** This only returns values on the workers initiated by this client. When using a [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy)like [**tf.distribute.experimental.MultiWorkerMirroredStrategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/MultiWorkerMirroredStrategy), each worker will be its own client, and this function will only return values computed on that worker.

#### Args:

* **value**: A value returned by experimental\_run(), experimental\_run\_v2(),extended.call\_for\_each\_replica(), or a variable created in scope.

#### Returns:

A tuple of values contained in value. If value represents a single value, this returns (value,).

### experimental\_make\_numpy\_dataset

experimental\_make\_numpy\_dataset(  
    numpy\_input,  
    session=None  
)

Makes a dataset for input provided via a numpy array.

This avoids adding numpy\_input as a large constant in the graph, and copies the data to the machine or machines that will be processing the input.

#### Args:

* **numpy\_input**: A nest of NumPy input arrays that will be distributed evenly across all replicas. Note that lists of Numpy arrays are stacked, as that is normal [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) behavior.
* **session**: (TensorFlow v1.x graph execution only) A session used for initialization.

#### Returns:

A [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) representing numpy\_input.

### experimental\_run

experimental\_run(  
    fn,  
    input\_iterator=None  
)

Runs ops in fn on each replica, with inputs from input\_iterator.

DEPRECATED: This method is not available in TF 2.x. Please switch to using experimental\_run\_v2instead.

When eager execution is enabled, executes ops specified by fn on each replica. Otherwise, builds a graph to execute the ops on each replica.

Each replica will take a single, different input from the inputs provided by one get\_next call on the input iterator.

fn may call tf.distribute.get\_replica\_context() to access members such as replica\_id\_in\_sync\_group.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The inputs to the function must match the outputs of input\_iterator.get\_next(). The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **input\_iterator**: (Optional) input iterator from which the inputs are taken.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be PerReplica (if the values are unsynchronized), Mirrored (if the values are kept in sync), or Tensor (if running on a single replica).

### experimental\_run\_v2

experimental\_run\_v2(  
    fn,  
    args=(),  
    kwargs=None  
)

Runs ops in fn on each replica, with the given arguments.

Executes ops specified by fn on each replica. If args or kwargs have "per-replica" values, such as those produced by a "distributed Dataset", when fn is executed on a particular replica, it will be executed with the component of those "per-replica" values that corresponds to that replica.

fn may call tf.distribute.get\_replica\_context() to access members such as all\_reduce.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **args**: (Optional) Positional arguments to fn.
* **kwargs**: (Optional) Keyword arguments to fn.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be "per-replica" Tensor objects or Tensors (for example, if running on a single replica).

### make\_dataset\_iterator

make\_dataset\_iterator(dataset)

Makes an iterator for input provided via dataset.

DEPRECATED: This method is not available in TF 2.x.

Data from the given dataset will be distributed evenly across all the compute replicas. We will assume that the input dataset is batched by the global batch size. With this assumption, we will make a best effort to divide each batch across all the replicas (one or more workers). If this effort fails, an error will be thrown, and the user should instead use make\_input\_fn\_iterator which provides more control to the user, and does not try to divide a batch across replicas.

The user could also use make\_input\_fn\_iterator if they want to customize which input is fed to which replica/worker etc.

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be distributed evenly across all replicas.

#### Returns:

An tf.distribute.InputIterator which returns inputs for each step of the computation. User should call initialize on the returned iterator.

### make\_input\_fn\_iterator

make\_input\_fn\_iterator(  
    input\_fn,  
    replication\_mode=tf.distribute.InputReplicationMode.PER\_WORKER  
)

Returns an iterator split across replicas created from an input function.

DEPRECATED: This method is not available in TF 2.x.

The input\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object where information about batching and input sharding can be accessed:

def input\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(input\_context.num\_input\_pipelines,  
                 input\_context.input\_pipeline\_id)  
with strategy.scope():  
  iterator = strategy.make\_input\_fn\_iterator(input\_fn)  
  replica\_results = strategy.experimental\_run(replica\_fn, iterator)

The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by input\_fn should have a per-replica batch size, which may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **input\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).
* **replication\_mode**: an enum value of [tf.distribute.InputReplicationMode](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputReplicationMode). Only PER\_WORKER is supported currently, which means there will be a single call to input\_fn per worker. Replicas will dequeue from the local [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) on their worker.

#### Returns:

An iterator object that should first be .initialize()-ed. It may then either be passed to strategy.experimental\_run() or you can iterator.get\_next() to get the next value to pass tostrategy.extended.call\_for\_each\_replica().

### reduce

reduce(  
    reduce\_op,  
    value,  
    axis=None  
)

Reduce value across replicas.

Given a per-replica value returned by experimental\_run\_v2, say a per-example loss, the batch will be divided across all the replicas. This function allows you to aggregate across replicas and optionally also across batch elements. For example, if you have a global batch size of 8 and 2 replicas, values for examples [0, 1, 2, 3] will be on replica 0 and [4, 5, 6, 7] will be on replica 1. By default, reduce will just aggregate across replicas, returning [0+4, 1+5, 2+6, 3+7]. This is useful when each replica is computing a scalar or some other value that doesn't have a "batch" dimension (like a gradient). More often you will want to aggregate across the global batch, which you can get by specifying the batch dimension as the axis, typically axis=0. In this case it would return a scalar 0+1+2+3+4+5+6+7.

If there is a last partial batch, you will need to specify an axis so that the resulting shape is consistent across replicas. So if the last batch has size 6 and it is divided into [0, 1, 2, 3] and [4, 5], you would get a shape mismatch unless you specify axis=0. If you specify [tf.distribute.ReduceOp.MEAN](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp#MEAN), using axis=0 will use the correct denominator of 6. Contrast this with computing reduce\_mean to get a scalar value on each replica and this function to average those means, which will weigh some values 1/8 and others 1/4.

#### Args:

* **reduce\_op**: A [tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) value specifying how values should be combined.
* **value**: A "per replica" value, e.g. returned by experimental\_run\_v2 to be combined into a single tensor.
* **axis**: Specifies the dimension to reduce along within each replica's tensor. Should typically be set to the batch dimension, or None to only reduce across replicas (e.g. if the tensor has no batch dimension).

#### Returns:

A Tensor.

### scope

scope()

Returns a context manager selecting this Strategy as current.

Inside a with strategy.scope(): code block, this thread will use a variable creator set by strategy, and will enter its "cross-replica context".

#### Returns:

A context manager.

### update\_config\_proto

update\_config\_proto(config\_proto)

Returns a copy of config\_proto modified for use with this strategy.

DEPRECATED: This method is not available in TF 2.x.

The updated config has something needed to run a strategy, e.g. configuration to run collective ops, or device filters to improve distributed training performance.

#### Args:

* **config\_proto**: a tf.ConfigProto object.

#### Returns:

The updated copy of the config\_proto.

# tf.compat.v1.distribute.experimental.ParameterServerStrategy

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/ParameterServerStrategy#top_of_page)
* [Class ParameterServerStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/ParameterServerStrategy#class_parameterserverstrategy)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/ParameterServerStrategy#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/ParameterServerStrategy#properties)
  + [extended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/ParameterServerStrategy#extended)

## Class ParameterServerStrategy

An asynchronous multi-worker parameter server DistributionStrategy.

Inherits From: [Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy)

Defined in [python/distribute/parameter\_server\_strategy.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/distribute/parameter_server_strategy.py).

This strategy requires two jobs: workers and parameter servers. Variables and updates to those variables will be assigned to parameter servers and other operations are assigned to workers.

When each worker has more than one GPU, operations will be replicated on these GPUs. Even though operations may be replicated, variables are not and each worker shares a common view for which parameter server a variable is assigned to.

By default it uses TFConfigClusterResolver to detect configurations for multi-worker training. This requires a 'TF\_CONFIG' environment variable and the 'TF\_CONFIG' must have a cluster spec.

This class assumes each worker is running the same code independently, but parameter servers are running a standard server. This means that while each worker will synchronously compute a single gradient update across all GPUs, updates between workers proceed asynchronously. Operations that occur only on the first replica (such as incrementing the global step), will occur on the first replica of every worker.

It is expected to call call\_for\_each\_replica(fn, ...) for any operations which potentially can be replicated across replicas (i.e. multiple GPUs) even if there is only CPU or one GPU. When defining the fn, extra caution needs to be taken:

1) It is generally not recommended to open a device scope under the strategy's scope. A device scope (i.e. calling [tf.device](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/device)) will be merged with or override the device for operations but will not change the device for variables.

2) It is also not recommended to open a colocation scope (i.e. calling[tf.compat.v1.colocate\_with](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/colocate_with)) under the strategy's scope. For colocating variables, use strategy.extended.colocate\_vars\_with instead. Colocation of ops will possibly create conflicts of device assignment.

## \_\_init\_\_

\_\_init\_\_(cluster\_resolver=None)

Initializes this strategy.

## Properties

### extended

[tf.distribute.StrategyExtended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended) with additional methods.

### num\_replicas\_in\_sync

Returns number of replicas over which gradients are aggregated.

## Methods

### experimental\_distribute\_dataset

experimental\_distribute\_dataset(dataset)

Distributes a tf.data.Dataset instance provided via dataset.

In a multi-worker setting, we will first attempt to distribute the dataset by attempting to detect whether the dataset is being created out of ReaderDatasets (e.g. TFRecordDataset, TextLineDataset, etc.) and if so, attempting to shard the input files. Note that there has to be at least one input file per worker. If you have less than one input file per worker, we suggest that you should disable distributing your dataset using the method below.

If that attempt is unsuccessful (e.g. the dataset is created from a Dataset.range), we will shard the dataset evenly at the end by appending a .shard operation to the end of the processing pipeline. This will cause the entire preprocessing pipeline for all the data to be run on every worker, and each worker will do redundant work. We will print a warning if this method of sharding is selected. In this case, consider using experimental\_distribute\_datasets\_from\_function instead.

You can disable dataset distribution using the auto\_shard option in[tf.data.experimental.DistributeOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/experimental/DistributeOptions).

Within each host, we will also split the data among all the worker devices (if more than one a present), and this will happen even if multi-worker sharding is disabled using the method above.

The following is an example:

strategy = tf.distribute.MirroredStrategy()  
  
# Create a dataset  
dataset = dataset\_ops.Dataset.TFRecordDataset([  
  "/a/1.tfr", "/a/2.tfr", "/a/3.tfr", /a/4.tfr"])  
  
# Distribute that dataset  
dist\_dataset = strategy.experimental\_distribute\_dataset(dataset)  
# Iterate over the distributed dataset  
for x in dist\_dataset:  
  # process dataset elements  
  strategy.experimental\_run\_v2(train\_step, args=(x,))

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be sharded across all replicas using the rules stated above.

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_distribute\_datasets\_from\_function

experimental\_distribute\_datasets\_from\_function(dataset\_fn)

Distributes [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) instances created by calls to dataset\_fn.

dataset\_fn will be called once for each worker in the strategy. Each replica on that worker will dequeue one batch of inputs from the local Dataset (i.e. if a worker has two replicas, two batches will be dequeued from the Dataset every step).

This method can be used for several purposes. For example, whereexperimental\_distribute\_dataset is unable to shard the input files, this method might be used to manually shard the dataset (avoiding the slow fallback behavior in experimental\_distribute\_dataset). In cases where the dataset is infinite, this sharding can be done by creating dataset replicas that differ only in their random seed.

The dataset\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance where information about batching and input replication can be accessed:

def dataset\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(  
      input\_context.num\_input\_pipelines, input\_context.input\_pipeline\_id)  
  
inputs = strategy.experimental\_distribute\_datasets\_from\_function(dataset\_fn)  
  
for batch in inputs:  
  replica\_results = strategy.experimental\_run\_v2(replica\_fn, args=(batch,))

IMPORTANT: The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by dataset\_fn should have a per-replica batch size, unlike experimental\_distribute\_dataset, which uses the global batch size. This may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **dataset\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_local\_results

experimental\_local\_results(value)

Returns the list of all local per-replica values contained in value.

**Note:** This only returns values on the workers initiated by this client. When using a [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy)like [**tf.distribute.experimental.MultiWorkerMirroredStrategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/MultiWorkerMirroredStrategy), each worker will be its own client, and this function will only return values computed on that worker.

#### Args:

* **value**: A value returned by experimental\_run(), experimental\_run\_v2(),extended.call\_for\_each\_replica(), or a variable created in scope.

#### Returns:

A tuple of values contained in value. If value represents a single value, this returns (value,).

### experimental\_make\_numpy\_dataset

experimental\_make\_numpy\_dataset(  
    numpy\_input,  
    session=None  
)

Makes a dataset for input provided via a numpy array.

This avoids adding numpy\_input as a large constant in the graph, and copies the data to the machine or machines that will be processing the input.

#### Args:

* **numpy\_input**: A nest of NumPy input arrays that will be distributed evenly across all replicas. Note that lists of Numpy arrays are stacked, as that is normal [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) behavior.
* **session**: (TensorFlow v1.x graph execution only) A session used for initialization.

#### Returns:

A [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) representing numpy\_input.

### experimental\_run

experimental\_run(  
    fn,  
    input\_iterator=None  
)

Runs ops in fn on each replica, with inputs from input\_iterator.

DEPRECATED: This method is not available in TF 2.x. Please switch to using experimental\_run\_v2instead.

When eager execution is enabled, executes ops specified by fn on each replica. Otherwise, builds a graph to execute the ops on each replica.

Each replica will take a single, different input from the inputs provided by one get\_next call on the input iterator.

fn may call tf.distribute.get\_replica\_context() to access members such as replica\_id\_in\_sync\_group.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The inputs to the function must match the outputs of input\_iterator.get\_next(). The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **input\_iterator**: (Optional) input iterator from which the inputs are taken.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be PerReplica (if the values are unsynchronized), Mirrored (if the values are kept in sync), or Tensor (if running on a single replica).

### experimental\_run\_v2

experimental\_run\_v2(  
    fn,  
    args=(),  
    kwargs=None  
)

Runs ops in fn on each replica, with the given arguments.

Executes ops specified by fn on each replica. If args or kwargs have "per-replica" values, such as those produced by a "distributed Dataset", when fn is executed on a particular replica, it will be executed with the component of those "per-replica" values that corresponds to that replica.

fn may call tf.distribute.get\_replica\_context() to access members such as all\_reduce.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **args**: (Optional) Positional arguments to fn.
* **kwargs**: (Optional) Keyword arguments to fn.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be "per-replica" Tensor objects or Tensors (for example, if running on a single replica).

### make\_dataset\_iterator

make\_dataset\_iterator(dataset)

Makes an iterator for input provided via dataset.

DEPRECATED: This method is not available in TF 2.x.

Data from the given dataset will be distributed evenly across all the compute replicas. We will assume that the input dataset is batched by the global batch size. With this assumption, we will make a best effort to divide each batch across all the replicas (one or more workers). If this effort fails, an error will be thrown, and the user should instead use make\_input\_fn\_iterator which provides more control to the user, and does not try to divide a batch across replicas.

The user could also use make\_input\_fn\_iterator if they want to customize which input is fed to which replica/worker etc.

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be distributed evenly across all replicas.

#### Returns:

An tf.distribute.InputIterator which returns inputs for each step of the computation. User should call initialize on the returned iterator.

### make\_input\_fn\_iterator

make\_input\_fn\_iterator(  
    input\_fn,  
    replication\_mode=tf.distribute.InputReplicationMode.PER\_WORKER  
)

Returns an iterator split across replicas created from an input function.

DEPRECATED: This method is not available in TF 2.x.

The input\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object where information about batching and input sharding can be accessed:

def input\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(input\_context.num\_input\_pipelines,  
                 input\_context.input\_pipeline\_id)  
with strategy.scope():  
  iterator = strategy.make\_input\_fn\_iterator(input\_fn)  
  replica\_results = strategy.experimental\_run(replica\_fn, iterator)

The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by input\_fn should have a per-replica batch size, which may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **input\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).
* **replication\_mode**: an enum value of [tf.distribute.InputReplicationMode](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputReplicationMode). Only PER\_WORKER is supported currently, which means there will be a single call to input\_fn per worker. Replicas will dequeue from the local [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) on their worker.

#### Returns:

An iterator object that should first be .initialize()-ed. It may then either be passed to strategy.experimental\_run() or you can iterator.get\_next() to get the next value to pass tostrategy.extended.call\_for\_each\_replica().

### reduce

reduce(  
    reduce\_op,  
    value,  
    axis=None  
)

Reduce value across replicas.

Given a per-replica value returned by experimental\_run\_v2, say a per-example loss, the batch will be divided across all the replicas. This function allows you to aggregate across replicas and optionally also across batch elements. For example, if you have a global batch size of 8 and 2 replicas, values for examples [0, 1, 2, 3] will be on replica 0 and [4, 5, 6, 7] will be on replica 1. By default, reduce will just aggregate across replicas, returning [0+4, 1+5, 2+6, 3+7]. This is useful when each replica is computing a scalar or some other value that doesn't have a "batch" dimension (like a gradient). More often you will want to aggregate across the global batch, which you can get by specifying the batch dimension as the axis, typically axis=0. In this case it would return a scalar 0+1+2+3+4+5+6+7.

If there is a last partial batch, you will need to specify an axis so that the resulting shape is consistent across replicas. So if the last batch has size 6 and it is divided into [0, 1, 2, 3] and [4, 5], you would get a shape mismatch unless you specify axis=0. If you specify [tf.distribute.ReduceOp.MEAN](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp#MEAN), using axis=0 will use the correct denominator of 6. Contrast this with computing reduce\_mean to get a scalar value on each replica and this function to average those means, which will weigh some values 1/8 and others 1/4.

#### Args:

* **reduce\_op**: A [tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) value specifying how values should be combined.
* **value**: A "per replica" value, e.g. returned by experimental\_run\_v2 to be combined into a single tensor.
* **axis**: Specifies the dimension to reduce along within each replica's tensor. Should typically be set to the batch dimension, or None to only reduce across replicas (e.g. if the tensor has no batch dimension).

#### Returns:

A Tensor.

### scope

scope()

Returns a context manager selecting this Strategy as current.

Inside a with strategy.scope(): code block, this thread will use a variable creator set by strategy, and will enter its "cross-replica context".

#### Returns:

A context manager.

### update\_config\_proto

update\_config\_proto(config\_proto)

Returns a copy of config\_proto modified for use with this strategy.

DEPRECATED: This method is not available in TF 2.x.

The updated config has something needed to run a strategy, e.g. configuration to run collective ops, or device filters to improve distributed training performance.

#### Args:

* **config\_proto**: a tf.ConfigProto object.

#### Returns:

The updated copy of the config\_proto.

# tf.compat.v1.distribute.experimental.TPUStrategy

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/TPUStrategy#top_of_page)
* [Class TPUStrategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/TPUStrategy#class_tpustrategy)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/TPUStrategy#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/TPUStrategy#properties)
  + [extended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/experimental/TPUStrategy#extended)

## Class TPUStrategy

TPU distribution strategy implementation.

Inherits From: [Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute/Strategy)

Defined in [python/distribute/tpu\_strategy.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/distribute/tpu_strategy.py).

## \_\_init\_\_

\_\_init\_\_(  
    tpu\_cluster\_resolver=None,  
    steps\_per\_run=None,  
    device\_assignment=None  
)

Initializes the TPUStrategy object.

#### Args:

* **tpu\_cluster\_resolver**: A tf.distribute.cluster\_resolver.TPUClusterResolver, which provides information about the TPU cluster.
* **steps\_per\_run**: Number of steps to run on device before returning to the host. Note that this can have side-effects on performance, hooks, metrics, summaries etc. This parameter is only used when Distribution Strategy is used with estimator or keras.
* **device\_assignment**: Optional tf.contrib.tpu.DeviceAssignment to specify the placement of replicas on the TPU cluster. Currently only supports the usecase of using a single core within a TPU cluster.

## Properties

### extended

[tf.distribute.StrategyExtended](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/StrategyExtended) with additional methods.

### num\_replicas\_in\_sync

Returns number of replicas over which gradients are aggregated.

### steps\_per\_run

DEPRECATED: use .extended.steps\_per\_run instead.

## Methods

### experimental\_distribute\_dataset

experimental\_distribute\_dataset(dataset)

Distributes a tf.data.Dataset instance provided via dataset.

In a multi-worker setting, we will first attempt to distribute the dataset by attempting to detect whether the dataset is being created out of ReaderDatasets (e.g. TFRecordDataset, TextLineDataset, etc.) and if so, attempting to shard the input files. Note that there has to be at least one input file per worker. If you have less than one input file per worker, we suggest that you should disable distributing your dataset using the method below.

If that attempt is unsuccessful (e.g. the dataset is created from a Dataset.range), we will shard the dataset evenly at the end by appending a .shard operation to the end of the processing pipeline. This will cause the entire preprocessing pipeline for all the data to be run on every worker, and each worker will do redundant work. We will print a warning if this method of sharding is selected. In this case, consider using experimental\_distribute\_datasets\_from\_function instead.

You can disable dataset distribution using the auto\_shard option in[tf.data.experimental.DistributeOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/experimental/DistributeOptions).

Within each host, we will also split the data among all the worker devices (if more than one a present), and this will happen even if multi-worker sharding is disabled using the method above.

The following is an example:

strategy = tf.distribute.MirroredStrategy()  
  
# Create a dataset  
dataset = dataset\_ops.Dataset.TFRecordDataset([  
  "/a/1.tfr", "/a/2.tfr", "/a/3.tfr", /a/4.tfr"])  
  
# Distribute that dataset  
dist\_dataset = strategy.experimental\_distribute\_dataset(dataset)  
# Iterate over the distributed dataset  
for x in dist\_dataset:  
  # process dataset elements  
  strategy.experimental\_run\_v2(train\_step, args=(x,))

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be sharded across all replicas using the rules stated above.

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_distribute\_datasets\_from\_function

experimental\_distribute\_datasets\_from\_function(dataset\_fn)

Distributes [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) instances created by calls to dataset\_fn.

dataset\_fn will be called once for each worker in the strategy. Each replica on that worker will dequeue one batch of inputs from the local Dataset (i.e. if a worker has two replicas, two batches will be dequeued from the Dataset every step).

This method can be used for several purposes. For example, whereexperimental\_distribute\_dataset is unable to shard the input files, this method might be used to manually shard the dataset (avoiding the slow fallback behavior in experimental\_distribute\_dataset). In cases where the dataset is infinite, this sharding can be done by creating dataset replicas that differ only in their random seed.

The dataset\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance where information about batching and input replication can be accessed:

def dataset\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(  
      input\_context.num\_input\_pipelines, input\_context.input\_pipeline\_id)  
  
inputs = strategy.experimental\_distribute\_datasets\_from\_function(dataset\_fn)  
  
for batch in inputs:  
  replica\_results = strategy.experimental\_run\_v2(replica\_fn, args=(batch,))

IMPORTANT: The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by dataset\_fn should have a per-replica batch size, unlike experimental\_distribute\_dataset, which uses the global batch size. This may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **dataset\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) instance and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).

#### Returns:

A "distributed Dataset", which acts like a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) except it produces "per-replica" values.

### experimental\_local\_results

experimental\_local\_results(value)

Returns the list of all local per-replica values contained in value.

**Note:** This only returns values on the workers initiated by this client. When using a [**tf.distribute.Strategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy)like [**tf.distribute.experimental.MultiWorkerMirroredStrategy**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/experimental/MultiWorkerMirroredStrategy), each worker will be its own client, and this function will only return values computed on that worker.

#### Args:

* **value**: A value returned by experimental\_run(), experimental\_run\_v2(),extended.call\_for\_each\_replica(), or a variable created in scope.

#### Returns:

A tuple of values contained in value. If value represents a single value, this returns (value,).

### experimental\_make\_numpy\_dataset

experimental\_make\_numpy\_dataset(  
    numpy\_input,  
    session=None  
)

Makes a dataset for input provided via a numpy array.

This avoids adding numpy\_input as a large constant in the graph, and copies the data to the machine or machines that will be processing the input.

#### Args:

* **numpy\_input**: A nest of NumPy input arrays that will be distributed evenly across all replicas. Note that lists of Numpy arrays are stacked, as that is normal [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) behavior.
* **session**: (TensorFlow v1.x graph execution only) A session used for initialization.

#### Returns:

A [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) representing numpy\_input.

### experimental\_run

experimental\_run(  
    fn,  
    input\_iterator=None  
)

Runs ops in fn on each replica, with inputs from input\_iterator.

DEPRECATED: This method is not available in TF 2.x. Please switch to using experimental\_run\_v2instead.

When eager execution is enabled, executes ops specified by fn on each replica. Otherwise, builds a graph to execute the ops on each replica.

Each replica will take a single, different input from the inputs provided by one get\_next call on the input iterator.

fn may call tf.distribute.get\_replica\_context() to access members such as replica\_id\_in\_sync\_group.

IMPORTANT: Depending on the [tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) implementation being used, and whether eager execution is enabled, fn may be called one or more times (once for each replica).

#### Args:

* **fn**: The function to run. The inputs to the function must match the outputs of input\_iterator.get\_next(). The output must be a [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) of Tensors.
* **input\_iterator**: (Optional) input iterator from which the inputs are taken.

#### Returns:

Merged return value of fn across replicas. The structure of the return value is the same as the return value from fn. Each element in the structure can either be PerReplica (if the values are unsynchronized), Mirrored (if the values are kept in sync), or Tensor (if running on a single replica).

### experimental\_run\_v2

experimental\_run\_v2(  
    fn,  
    args=(),  
    kwargs=None  
)

See base class.

### make\_dataset\_iterator

make\_dataset\_iterator(dataset)

Makes an iterator for input provided via dataset.

DEPRECATED: This method is not available in TF 2.x.

Data from the given dataset will be distributed evenly across all the compute replicas. We will assume that the input dataset is batched by the global batch size. With this assumption, we will make a best effort to divide each batch across all the replicas (one or more workers). If this effort fails, an error will be thrown, and the user should instead use make\_input\_fn\_iterator which provides more control to the user, and does not try to divide a batch across replicas.

The user could also use make\_input\_fn\_iterator if they want to customize which input is fed to which replica/worker etc.

#### Args:

* **dataset**: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) that will be distributed evenly across all replicas.

#### Returns:

An tf.distribute.InputIterator which returns inputs for each step of the computation. User should call initialize on the returned iterator.

### make\_input\_fn\_iterator

make\_input\_fn\_iterator(  
    input\_fn,  
    replication\_mode=tf.distribute.InputReplicationMode.PER\_WORKER  
)

Returns an iterator split across replicas created from an input function.

DEPRECATED: This method is not available in TF 2.x.

The input\_fn should take an [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object where information about batching and input sharding can be accessed:

def input\_fn(input\_context):  
  batch\_size = input\_context.get\_per\_replica\_batch\_size(global\_batch\_size)  
  d = tf.data.Dataset.from\_tensors([[1.]]).repeat().batch(batch\_size)  
  return d.shard(input\_context.num\_input\_pipelines,  
                 input\_context.input\_pipeline\_id)  
with strategy.scope():  
  iterator = strategy.make\_input\_fn\_iterator(input\_fn)  
  replica\_results = strategy.experimental\_run(replica\_fn, iterator)

The [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) returned by input\_fn should have a per-replica batch size, which may be computed using input\_context.get\_per\_replica\_batch\_size.

#### Args:

* **input\_fn**: A function taking a [tf.distribute.InputContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputContext) object and returning a [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset).
* **replication\_mode**: an enum value of [tf.distribute.InputReplicationMode](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/InputReplicationMode). Only PER\_WORKER is supported currently, which means there will be a single call to input\_fn per worker. Replicas will dequeue from the local [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) on their worker.

#### Returns:

An iterator object that should first be .initialize()-ed. It may then either be passed to strategy.experimental\_run() or you can iterator.get\_next() to get the next value to pass tostrategy.extended.call\_for\_each\_replica().

### reduce

reduce(  
    reduce\_op,  
    value,  
    axis=None  
)

Reduce value across replicas.

Given a per-replica value returned by experimental\_run\_v2, say a per-example loss, the batch will be divided across all the replicas. This function allows you to aggregate across replicas and optionally also across batch elements. For example, if you have a global batch size of 8 and 2 replicas, values for examples [0, 1, 2, 3] will be on replica 0 and [4, 5, 6, 7] will be on replica 1. By default, reduce will just aggregate across replicas, returning [0+4, 1+5, 2+6, 3+7]. This is useful when each replica is computing a scalar or some other value that doesn't have a "batch" dimension (like a gradient). More often you will want to aggregate across the global batch, which you can get by specifying the batch dimension as the axis, typically axis=0. In this case it would return a scalar 0+1+2+3+4+5+6+7.

If there is a last partial batch, you will need to specify an axis so that the resulting shape is consistent across replicas. So if the last batch has size 6 and it is divided into [0, 1, 2, 3] and [4, 5], you would get a shape mismatch unless you specify axis=0. If you specify [tf.distribute.ReduceOp.MEAN](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp#MEAN), using axis=0 will use the correct denominator of 6. Contrast this with computing reduce\_mean to get a scalar value on each replica and this function to average those means, which will weigh some values 1/8 and others 1/4.

#### Args:

* **reduce\_op**: A [tf.distribute.ReduceOp](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/ReduceOp) value specifying how values should be combined.
* **value**: A "per replica" value, e.g. returned by experimental\_run\_v2 to be combined into a single tensor.
* **axis**: Specifies the dimension to reduce along within each replica's tensor. Should typically be set to the batch dimension, or None to only reduce across replicas (e.g. if the tensor has no batch dimension).

#### Returns:

A Tensor.

### scope

scope()

Returns a context manager selecting this Strategy as current.

Inside a with strategy.scope(): code block, this thread will use a variable creator set by strategy, and will enter its "cross-replica context".

#### Returns:

A context manager.

### update\_config\_proto

update\_config\_proto(config\_proto)

Returns a copy of config\_proto modified for use with this strategy.

DEPRECATED: This method is not available in TF 2.x.

The updated config has something needed to run a strategy, e.g. configuration to run collective ops, or device filters to improve distributed training performance.

#### Args:

* **config\_proto**: a tf.ConfigProto object.

#### Returns:

The updated copy of the config\_proto.