# tf.contrib.tpu.InfeedQueue

Contents



## Class `InfeedQueue`

Defined in `tensorflow/contrib/tpu/python/tpu/tpu_feed.py`.

A helper object to build a device infeed queue.

The InfeedQueue builds the host-side and device-side Ops to enqueue and dequeue elements, respectively, and ensures that their types and shapes match.

## Properties

### `number_of_shards`

Gets the number of shards to use for the InfeedQueue.

#### Returns:

Number of shards or None if the number of shards has not been set.

### `number_of_tuple_elements`

Returns the number of InfeedQueue tuple elements.

### `shard_dimensions`

Gets the shard dimension of each tuple element.

#### Returns:

A list of length number_of_tuple_elements, where each list entry is the shard dimension of that tuple element or None if the shard dimension has not been set.

### `sharding_policies`

Returns the sharding policies of the InfeedQueue tuple elements.

### `tuple_shapes`

Returns the shapes of the InfeedQueue tuple elements.

### `tuple_types`

Returns the types of the InfeedQueue tuple elements.

## Methods

### `__init__`

```
__init__(
    number_of_tuple_elements=None,
    tuple_types=None,
    tuple_shapes=None,
    shard_dimensions=None,
    name=None
)
```

Creates a new InfeedQueue with the given configuration.

The configuration need not be fully specified at creation since it can be modified subsequently by methods that set the values explicitly or infer them from the shapes of inputs.

Args:

- `number_of_tuple_elements`: the number of Tensors fed atomically through the queue, must be present unless it can be inferred from other arguments.
- `tuple_types`: if not None, a list of types of the elements of the queue.
- `tuple_shapes`: if not None, a list of shapes of the elements of the queue.
- `shard_dimensions`: if not None, a list of dimensions on which the elements of the queue should be sharded during automatic parallelization.
- `name`: the name of the queue.

Raises:

- `ValueError`: if number_of_tuple_elements <= 0; or number_of_tuple_arguments, tuple_types, tuple_shapes, and shard_dimensions are all None; or the length of tuple_types, tuple_shapes, or shard_dimensions is not equal to number_of_tuple_elements; or any element of shard_dimensions can't be converted to a Dimension.
- `TypeError`: if any element of tuple_types or tuple_shapes can't be converted to a dtype or TensorShape, respectively.

### `freeze`

```
freeze()
```

Freezes the InfeedQueue so it can no longer be modified.

The configuration is implicitly frozen before any host-side or device-side Ops are generated. The configuration cannot be frozen until the types and shapes of the tuple elements have been set.

Raises:

- `ValueError`: if the types or shapes of the tuple elements have not been set.

### `generate_dequeue_op`

```
generate_dequeue_op()
```

Generates the device-side Op to dequeue a tuple from the queue.

Implicitly freezes the queue configuration if it is not already frozen, which will raise errors if the shapes and types have not been fully specified.

Returns:

A list of Outputs corresponding to a shard of infeed dequeued into XLA, suitable for use within a replicated block.

Raises:

- `ValueError`: if the types or shapes of the tuple elements have not been set; or if a dequeue op has already been generated.

### `generate_enqueue_ops`

```
generate_enqueue_ops(
    sharded_inputs,
    tpu_ordinal_function=None
)
```

Generates the host-side Ops to enqueue the shards of a tuple.

sharded_inputs is a list, one for each shard, of lists of Tensors. sharded_inputs[0] is the tuple of Tensors to use to feed shard 0 if the queue. Returns the host-side Ops that must be run to enqueue the sharded tuple. The Op for shard i is colocated with the inputs for shard i.

Implicitly freezes the queue configuration if it is not already frozen. If the configuration has already been frozen, and is not compatible with the types and shapes of sharded_inputs, an error will be raised.

Args:

- `sharded_inputs`: a list of lists of Tensors. The length of the outer list determines the number of shards. Each inner list indicates the types and shapes of the tuples in the corresponding shard.
- `tpu_ordinal_function`: if not None, a function that takes the shard index as input and returns the ordinal of the TPU device the shard's infeed should be placed on. tpu_ordinal_function must be set if the inputs are placed on CPU devices.

Returns:

A list of host-side Ops, one for each shard, that when executed together will enqueue a full-size element of infeed.

Raises:

- `ValueError`: if the queue configuration has previously been frozen and the shapes of the elements of

sharded_inputs are not compatible with the frozen configuration; or if the shapes of the elements of sharded_inputs don't form a consistent unsharded tuple; or if the elements of a tuple have different device constraints.

- `TypeError`: if the queue configuration has previously been frozen and the types of the elements of sharded_inputs are not compatible with the frozen configuration; or if the types of the elements of sharded_inputs don't form a consistent unsharded tuple.

### `set_configuration_from_input_tensors`

```
set_configuration_from_input_tensors(input_tensors)
```

Sets the shapes and types of the queue tuple elements.

input_tensors is a list of Tensors whose types and shapes are used to set the queue configuration.

Args:

- `input_tensors`: list of Tensors of the same types and shapes as the desired queue Tuple.

Raises:

- `ValueError`: if input_tensors is not a list of length self.number_of_tuple_elements

### `set_configuration_from_sharded_input_tensors`

```
set_configuration_from_sharded_input_tensors(input_tensors)
```

Sets the shapes and types of the queue tuple elements.

input_tensors is a list of lists of Tensors whose types and shapes are used to set the queue configuration. The length of the outer list is the number of shards required, and each inner list is the tuple of Tensors to use to determine the types and shapes of the corresponding shard. This method depends on the shard dimension, and calling it freezes the shard policy.

Args:

- `input_tensors`: list of lists of Tensors. The outer list length corresponds to the desired number of shards, and each inner list is the size and shape of the desired configuration of the corresponding shard.

Raises:

- `ValueError`: if any inner list is not a list of length self.number_of_tuple_elements; or the inner lists do not combine to form a consistent unsharded shape.
- `TypeError`: if the types of the Tensors in the inner lists do not match.

### `set_number_of_shards`

```
set_number_of_shards(number_of_shards)
```

Sets the number of shards to use for the InfeedQueue.

Args:

- `number_of_shards`: number of ways to shard the InfeedQueue.

Raises:

- `ValueError`: if number_of_shards is not > 0; or the policies have been frozen and number_of_shards was already set to something else.

### `set_shard_dimensions`

```
set_shard_dimensions(shard_dimensions)
```

Sets the shard_dimension of each element of the queue.

shard_dimensions must be a list of length self.number_of_tuple_elements, and each element must be convertible to a Dimension compatible with self.tuple_shapes.

Args:

- `shard_dimensions`: the dimensions of each queue element.

Raises:

- `ValueError`: if shard_dimensions is not of length self.number_of_tuple_elements; or an element of shard_dimensions cannot be converted to a Dimension; or an element of shard_dimensions is a Dimension that is out of range for the corresponding tuple element shape.

### `set_tuple_shapes`

```
set_tuple_shapes(tuple_shapes)
```

Sets the shape of each element of the queue.

tuple_shapes must be a list of length self.number_of_tuple_elements, and each element must be convertible to a TensorShape.

Args:

- `tuple_shapes`: the shapes of each queue element.

Raises:

- `ValueError`: if tuple_shapes is not of length self.number_of_tuple_elements.
- `TypeError`: if an element of tuple_shapes cannot be converted to a TensorShape.

### `set_tuple_types`

```
set_tuple_types(tuple_types)
```

Sets the type of each element of the queue.

tuple_types must be a list of length self.number_of_tuple_elements, and each element must be convertible to a dtype.

Args:

- `tuple_types`: the types of each queue element.

Raises:

- `ValueError`: if tuple_types is not of length self.number_of_tuple_elements.
- `TypeError`: if an element of tuple_types cannot be converted to a dtype.

### split_inputs_and_generate_enqueue_ops

```
split_inputs_and_generate_enqueue_ops(
    inputs,
    global_tpu_id=None,
    placement_function=None,
    tpu_ordinal_function=None
)
```

POORLY-PERFORMING ON MULTI-HOST SYSTEMS.

Generates the host-side Ops to enqueue a tuple.

This method performs poorly because it takes an entire input on a single host, splits it, and distributes it to all of the cores. It is present only to simplify tutorial examples.

inputs is a list of Tensors to use to feed the queue. Each input is split into self.number_of_shards shards. Returns an Op for each shard to enqueue the shard. The Op for shard i is placed on device placement_function(i).

Implicitly freezes the queue configuration if it is not already frozen. If the configuration has already been frozen, and is not compatible with the types and shapes of inputs, an error will be raised.

Args:

- `inputs`: a list of Tensors which indicates the types and shapes of the queue tuple. global_tpu_id: if not None, a Numpy 2D array indicating the global id of each TPU device in the system. The outer dimension of the array is host task id, and the inner dimension is device ordinal, so e.g., global_tpu_id[x][y] indicates the global id of device /task:x/device:TPU_NODE:y. If global_tpu_id is not None, but placement_function and ordinal_function are None, then global_tpu_id will be used to place infeed on the TPUs with the first k global ids, where k is the number of shards in the queue.
- `placement_function`: if not None, a function that takes the shard index as input and returns a device string indicating which device the shard's infeed should be placed on. If placement_function and tpu_ordinal_function are None, inputs are sharded round-robin across the devices in the system.
- `tpu_ordinal_function`: if not None, a function that takes the shard index as input and returns the ordinal of the TPU device the shard's infeed should be placed on. If placement_function and tpu_ordinal_function are None, inputs are sharded round-robin across the devices in the system.

Returns:

A list of host-side Ops, one for each shard, that when executed together will enqueue a full-size element of infeed.

Raises:

- `ValueError`: if the queue configuration has previously been frozen and the shapes of the elements of inputs are not

compatible with the frozen configuration.

- `TypeError`: if the queue configuration has previously been frozen and the types of the elements of inputs are not compatible with the frozen configuration.

---



*Last updated November 2, 2017.*