Operator name: StatefulPartitionedCall

Doc link: NA

Understanding: A partitioned call is a function that executes a function across multiple devices. Functions that execute within the same address space can be executed. As arguments, it takes a function to be executed, the function’s arguments, an execution context. It returns a list of output tensors obtained by invoking f(args) on the device. If the function does not return anything, it will return None.

If the context is in eager mode, the stateful\_partitioned\_call() method is called; else, the partitioned\_call() method is called. ([https://github.com/tensorflow/tensorflow/blob/f3c0a3e0e4cba2727710b023757677f4dc331e48/tensorflow/python/ops/functional\_ops.py#L1084](https://github.com/tensorflow/tensorflow/blob/f3c0a3e0e4cba2727710b023757677f4dc331e48/tensorflow/python/ops/functional_ops.py" \l "L1084))

Upon inspecting gen\_functional\_ops.py, stateful\_partitioned\_call() accepts the following arguments:

Tensor inputs, a list of output tensor datatypes, and the function to the invoked. Quoting the documentation:

“The function body of *f* will be placed and partitioned across devices, setting this op apart from the regular Call op. This op is stateful.”

This operator is registered in Tensorflow as below:

REGISTER\_OP("StatefulPartitionedCall")

.Input("args: Tin")

.Output("output: Tout")

.Attr("Tin: list(type) >= 0")

.Attr("Tout: list(type) >= 0")

.Attr("f: func")

.Attr("config: string = ''") // Deprecated in favor of config\_proto

.Attr("config\_proto: string = ''")

.Attr("executor\_type: string = ''")

.SetIsStateful()

.SetShapeFn(shape\_inference::UnknownShape);

From the C++ backend implementation of Tensorflow, this operator class is defined as follows:

// A `PartitionedCallOp` asynchronously executes a function, potentially across

// multiple devices : public AsyncOpKernel {

but within a single process. The kernel places and

// partitions a given function's underlying graph, and executes each of the

// partitioned subgraphs as a function.

//

// TODO(akshayka): Support distributed execution.

class PartitionedCallOp: public AsyncOpKernel {

...

}

To summarize:

We could define a function, say, a = (a\*x) + (b\*y) as a graph, where a, x, b, and y are nodes, and multiplies and adds are performed on separate devices.

The sub-graphs, i.e., a\*x and b\*y will be performed on different devices and merged onto one with the addition operator.

Notes: We believe that this operator is never directly used by a data scientist and some features that it uses are “experimental”. Also how the graph is partitioned(subgraphs creation, distribution to multiple devices in same address space, consolidating outputs) and how the statefulness is utilized is not very clear.

We are now trying to understand if it makes sense to map/implement this operator in TVM and if so, then what shall be the right approach.