Introduction to TensorFlow Internals

Guangcong Liu

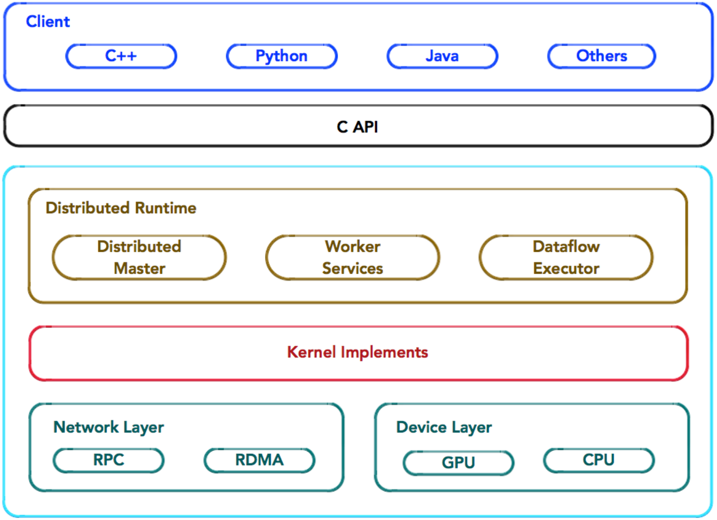
Software Architect@ZTE

2017-11-16

Contents

Architecture Overview

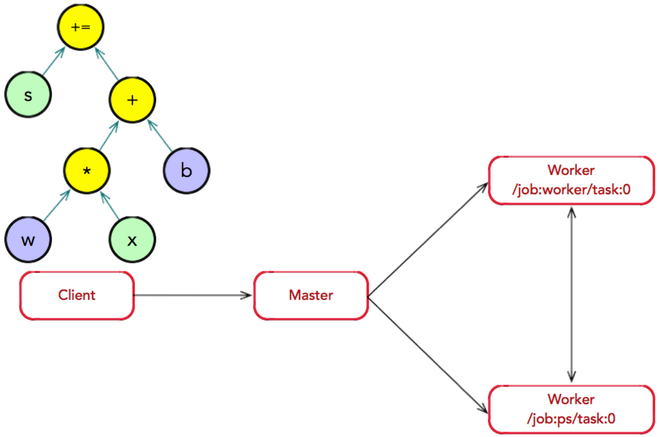
System Architecture



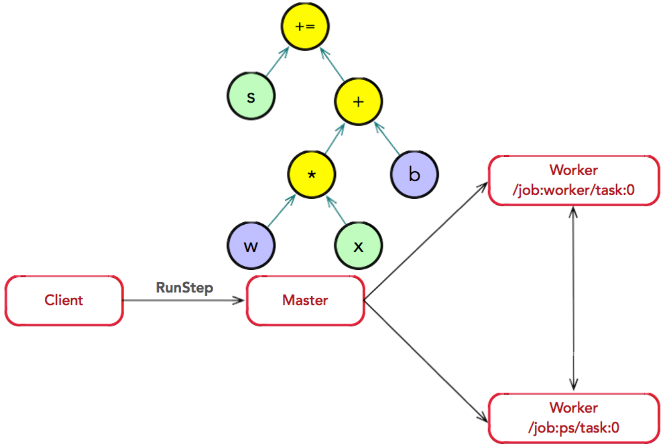
Design Principles

- Deferred Execution： The construction and execution of graph are separated, and the graph execution is delayed.
- Primitive OP： OP is the basic computation unit.
- Abstract Accelerator： Support CPU, GPU, and ASIC.

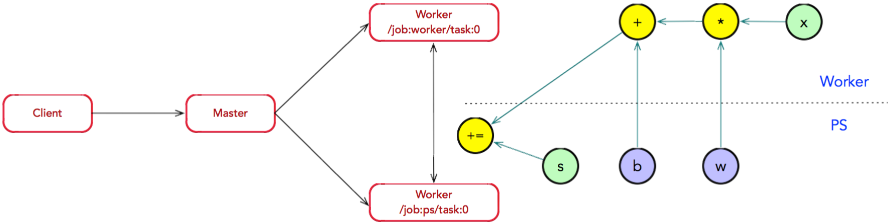
Graph Construction



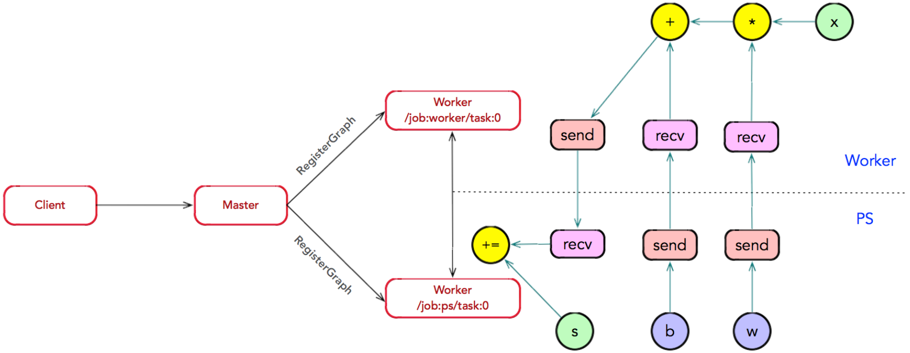
Graph Exection



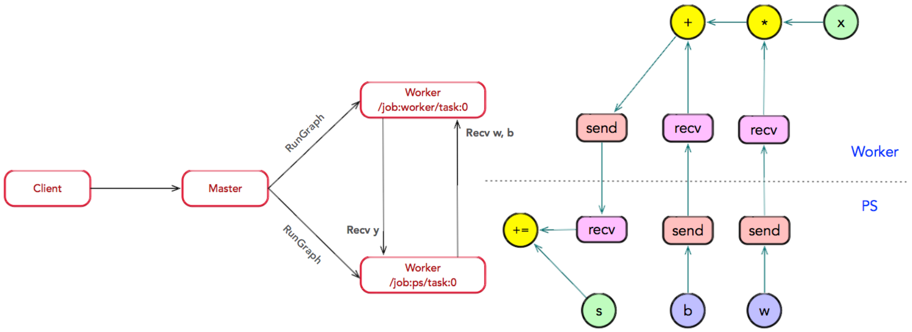
Split Graph


Client
Master
Worker
/job:worker/task:0
Worker
/job:ps/task:0
+
*
x
Worker
PS
+=
s
b
w

Register Graph



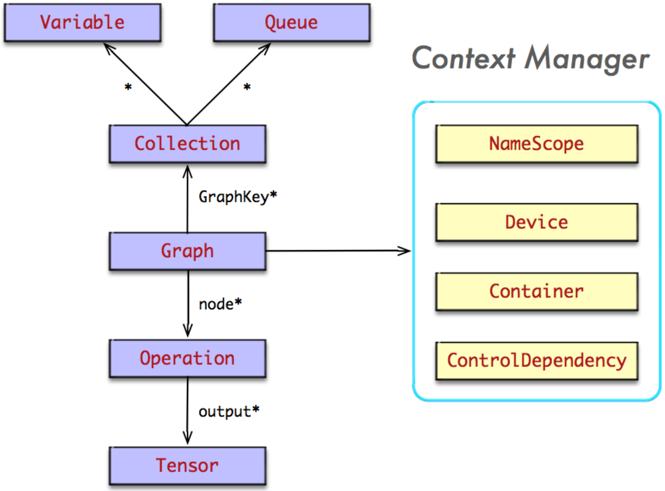
Run Graph


Client
Master
RunGraph
RunGraph
Worker /job:worker/task:0
Worker /job:ps/task:0
Recv w, b
Recv y
+
*
x
send
recv
recv
Worker
PS
+=
recv
send
send
s
b
w

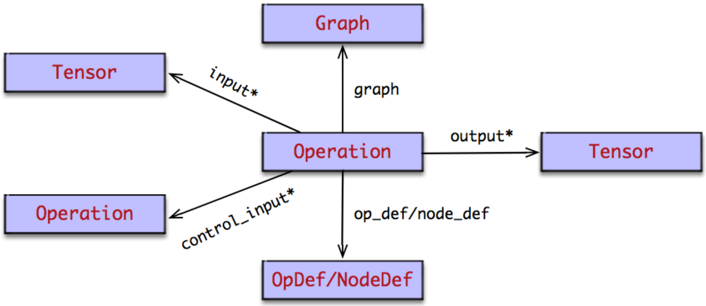
Programming Model

1. Dataflow Graph
2. Variable
3. Session
4. Graph Construction & Exection

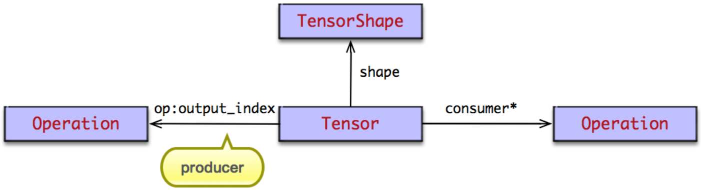
$Graph = Set\{OP\} + Set\{Tensor\}$



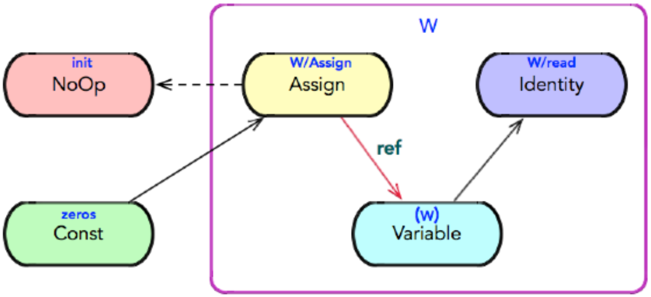
OP: Abstract Computation



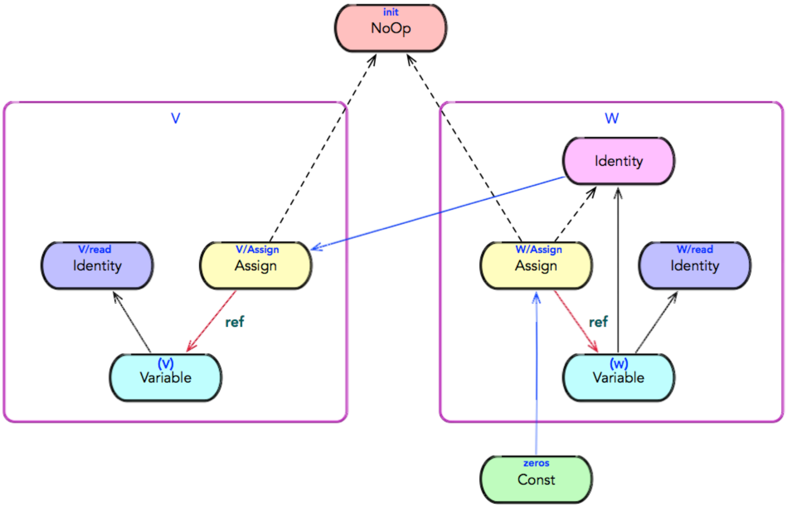
Tensor: Dataflow



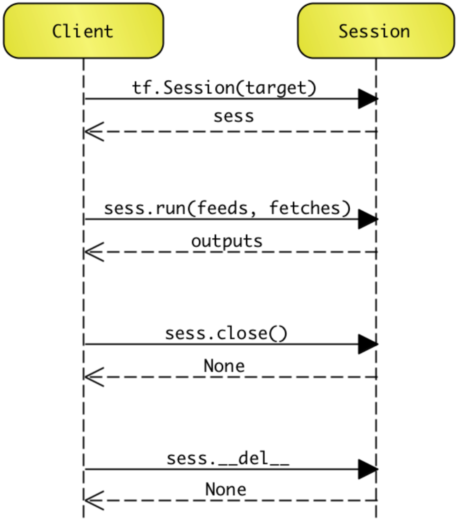
Initialization Model



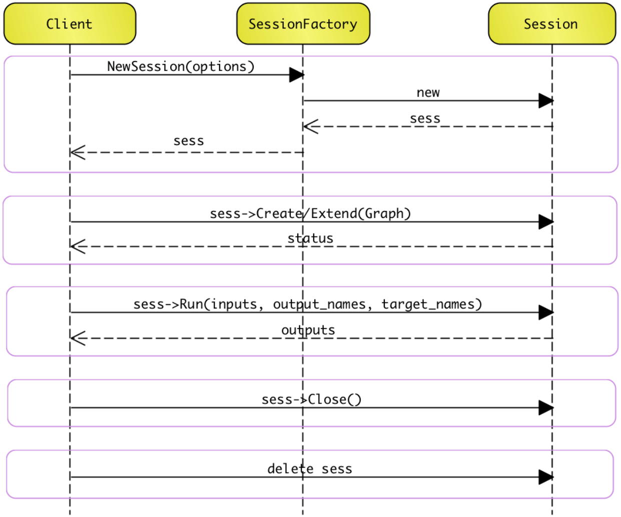
Initialization Dependency



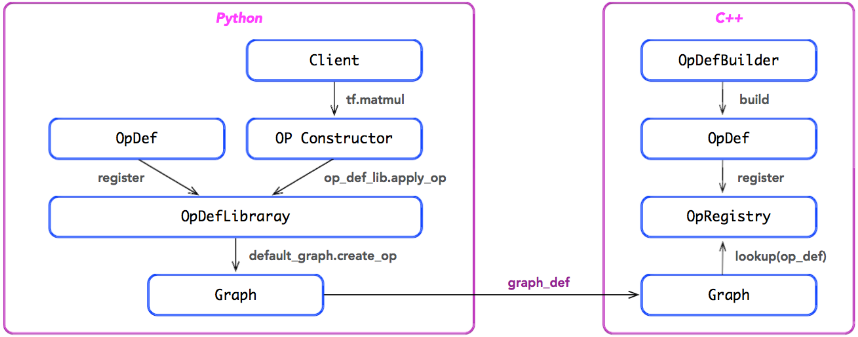
Life Cycle: Python



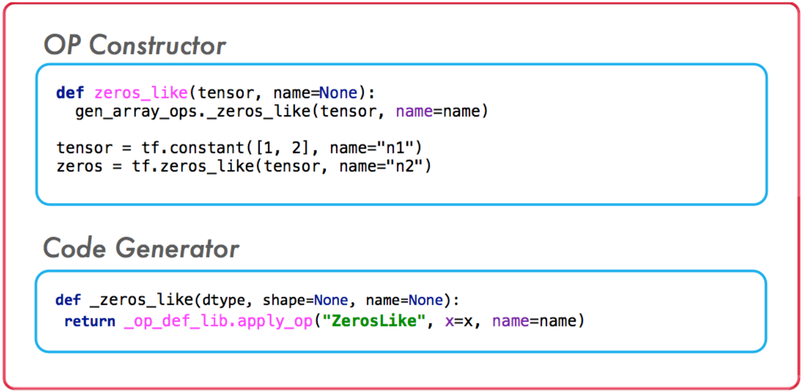
Life Cycle: C++


Client
SessionFactory
Session
NewSession(options)
new
sess
sess
sess->Create/Extend(Graph)
status
sess->Run(inputs, output_names, target_names)
outputs
sess->Close()
delete sess

Graph Construction & Serialization



Example: OP Constructor



OP Constructor

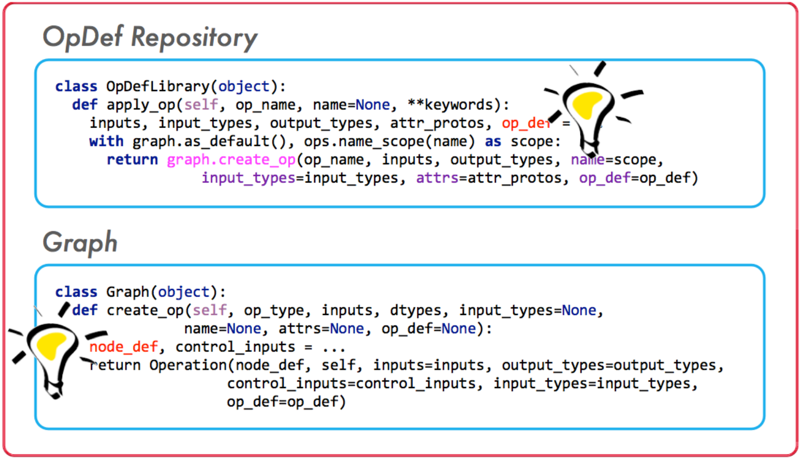
```
def zeros_like(tensor, name=None):
  gen_array_ops._zeros_like(tensor, name=name)

tensor = tf.constant([1, 2], name="n1")
zeros = tf.zeros_like(tensor, name="n2")
```

Code Generator

```
def _zeros_like(dtype, shape=None, name=None):
 return _op_def_lib.apply_op("ZerosLike", x=x, name=name)
```

Example: Create OP



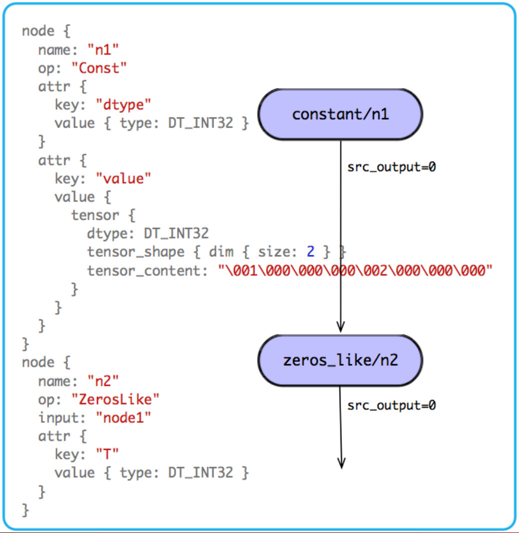
OpDef Repository

```
class OpDefLibrary(object):
  def apply_op(self, op_name, name=None, **keywords):
    inputs, input_types, output_types, attr_protos, op_def =
    with graph.as_default(), ops.name_scope(name) as scope:
      return graph.create_op(op_name, inputs, output_types, name=scope,
                  input_types=input_types, attrs=attr_protos, op_def=op_def)
```

Graph

```
class Graph(object):
  def create_op(self, op_type, inputs, dtypes, input_types=None,
                name=None, attrs=None, op_def=None):
    node_def, control_inputs = ...
    return Operation(node_def, self, inputs=inputs, output_types=output_types,
                     control_inputs=control_inputs, input_types=input_types,
                     op_def=op_def)
```

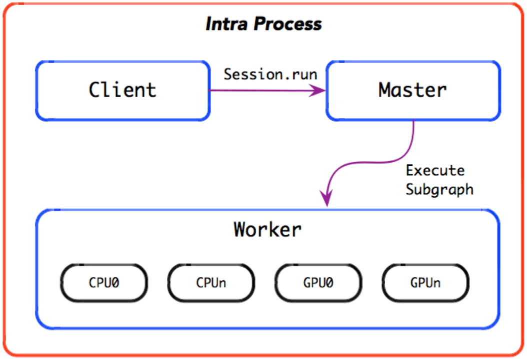
Example: Graph Construction



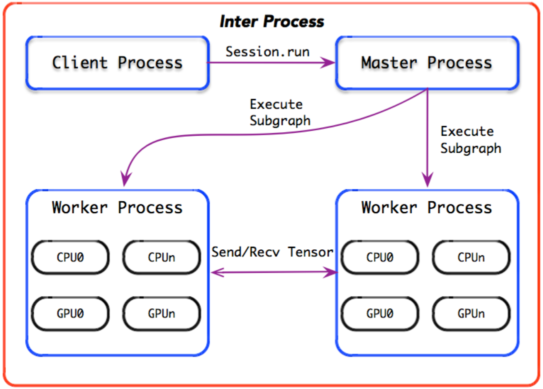
```
node {
  name: "n1"
  op: "Const"
  attr {
    key: "dtype"
    value { type: DT_INT32 }
  }
  attr {
    key: "value"
    value {
      tensor {
        dtype: DT_INT32
        tensor_shape { dim { size: 2 } }
        tensor_content: "\001\000\000\000\002\000\000\000"
      }
    }
  }
}
node {
  name: "n2"
  op: "ZerosLike"
  input: "node1"
  attr {
    key: "T"
    value { type: DT_INT32 }
  }
}
```

Execution Model

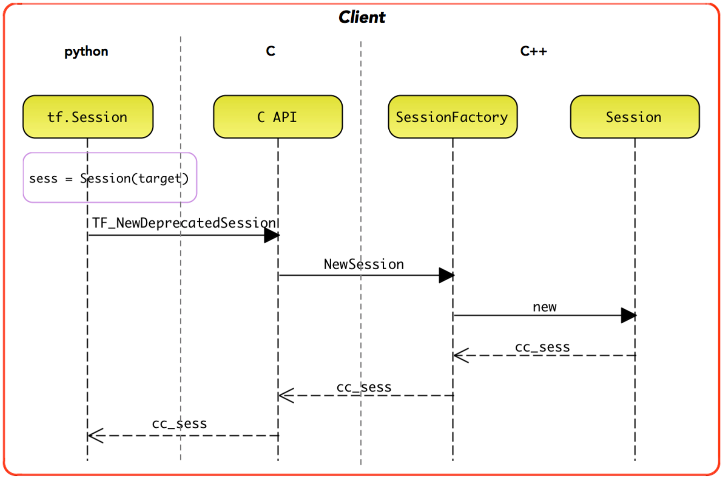
Local Runtime



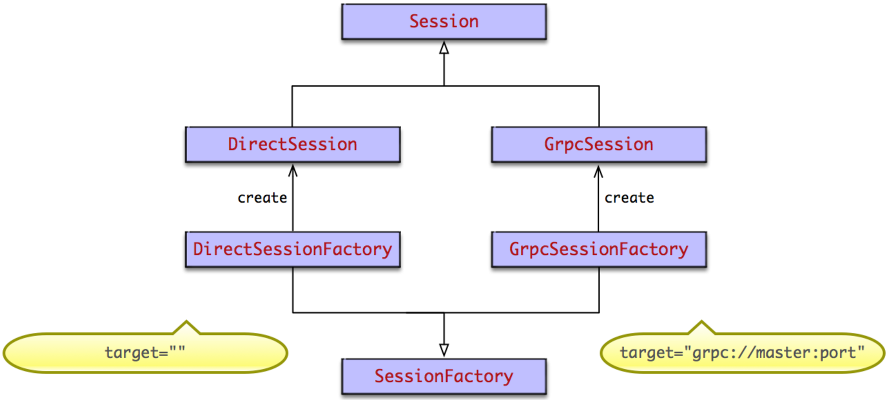
Distributed Runtime



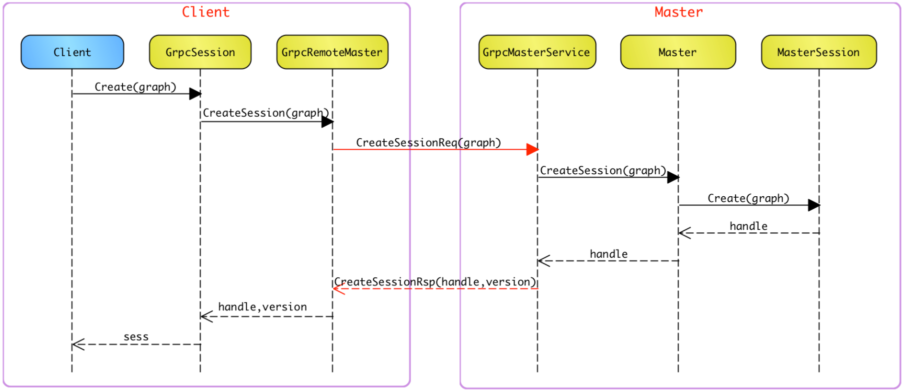
Create ClientSession



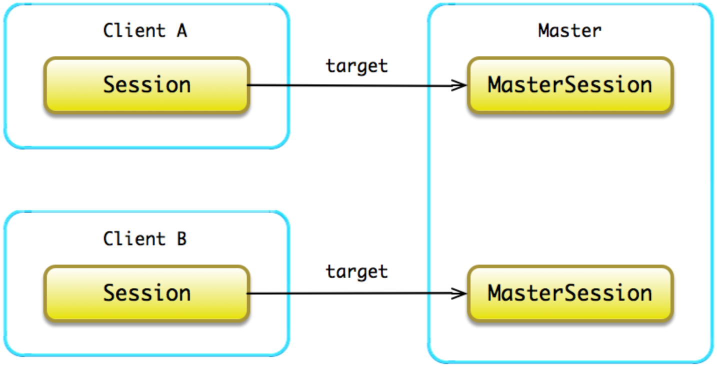
Polymorphism Creation



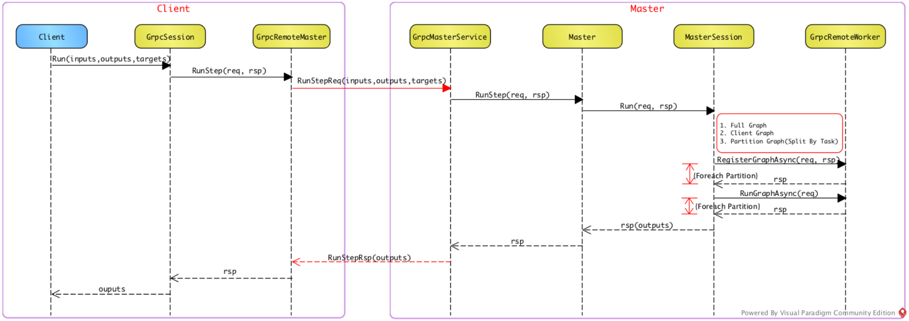
Create MasterSession



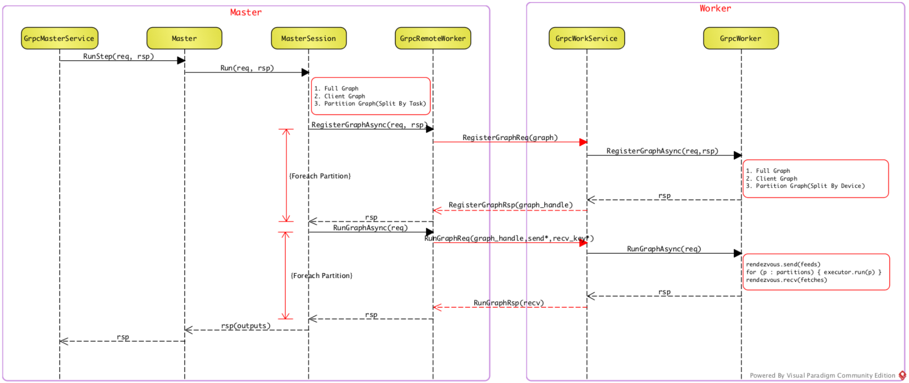
MasterSession Model



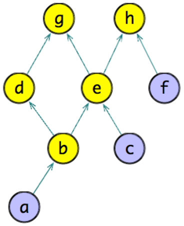
Split Graph by Task)

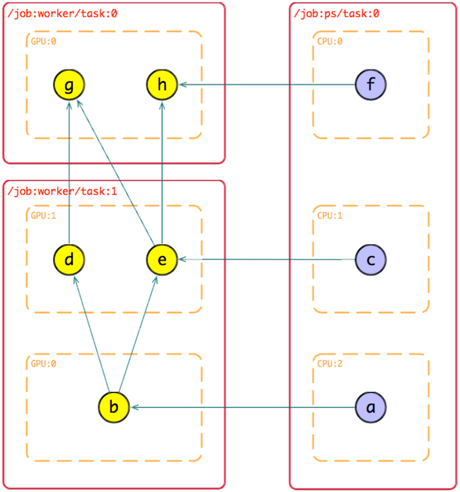


Split Graph by Device

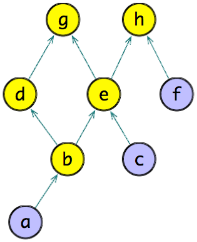


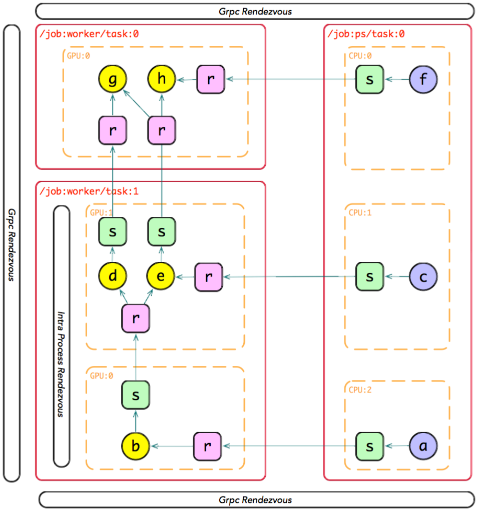
Split Graph


g
h
d
e
f
b
c
a


/job:worker/task:0
GPU:0
g
h
/job:worker/task:1
GPU:1
d
e
GPU:0
b
/job:ps/task:0
CPU:0
f
CPU:1
c
CPU:2
a

Receive Tensor





Training Model

1. Compute Gradients
2. Apply Gradients
3. Training Workflow

Optimizer: Compute Gradients

```
class Optimizer(object):
  def minimize(self, loss, var_list=None, global_step=None):
    grads_and_vars = self.compute_gradients(
      loss, var_list=var_list)
    return self.apply_gradients(
      grads_and_vars,
      global_step=global_step)

  def compute_gradients(loss, var_list):
    grads = gradients(loss, var_list, grad)
    return list(zip(grads, var_list))

  def gradients(loss, var_list, grads=1):
    ops_and_grads = {}
    for op in reversed_graph(loss).topological_sort():
      grad = op.grad_fn(grad)
      ops_and_grads[op] = grad
    return [ops_and_grads.get(var) for var in var_list]
```

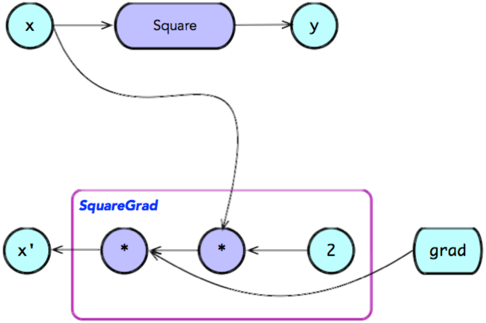
Gradient Function

```
@ops.RegisterGradient("op_name")
def grad_func(op, grad):
  """construct gradient subgraph for an op type.
  Returns:
    A list of gradients, one per each input of op.
  """
  return cons_grad_subgraph(op, grad)
```

$$(y_1, y_2, ..., y_m) = f(x_1, x_2, ..., x_n)$$
$$\left(\partial L/\partial x_1, \partial L/\partial x_2, ..., \partial L/\partial x_n\right) = g\left(x_1, x_2, ..., x_n; \partial L/\partial y_1, \partial L/\partial y_2, ..., \partial L/\partial y_n\right)$$

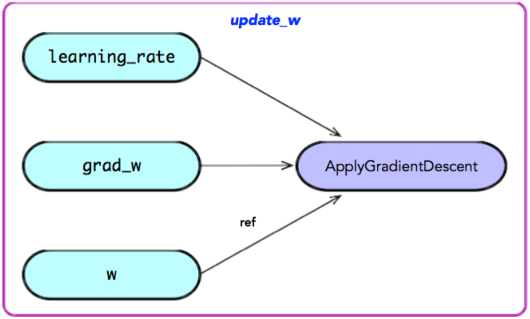
Example: Square

```
@ops.RegisterGradient("Square")
def SquareGrad(op, grad):
  x = op.inputs[0]
  with ops.control_dependencies([grad.op]):
    return grad * (2.0 * x)
```

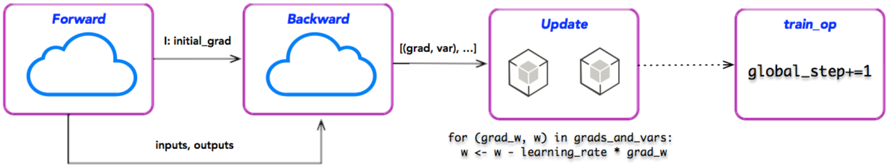


Apply Gradients

```
def apply_gradients(grads_and_vars, learning_rate):
  for (grad, var) in grads_and_vars:
    apply_gradient_descent(learning_rate, grad, var)
```

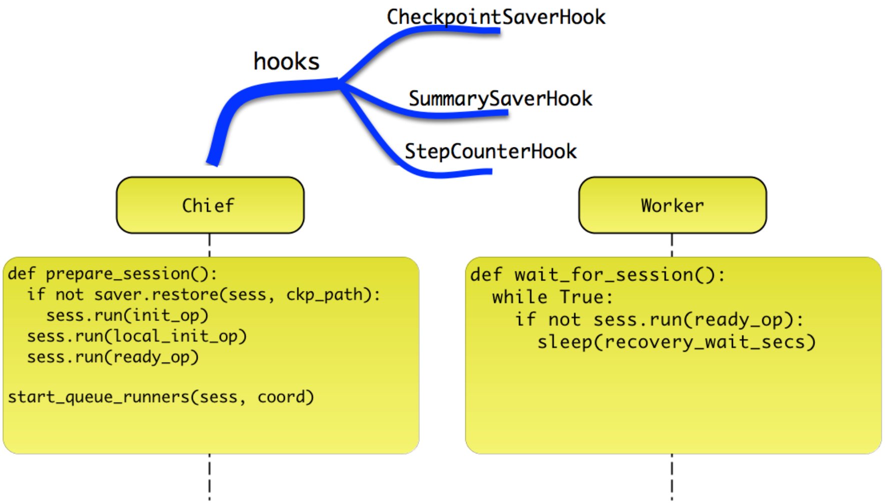


Critical Path: RunStep



```
for (grad_w, w) in grads_and_vars:
  w <- w - learning_rate * grad_w
```

Distributed Initialization



```
def prepare_session():
  if not saver.restore(sess, ckp_path):
    sess.run(init_op)
  sess.run(local_init_op)
  sess.run(ready_op)

start_queue_runners(sess, coord)
```

```
def wait_for_session():
  while True:
    if not sess.run(ready_op):
      sleep(recovery_wait_secs)
```

Bibliography

Papers

- TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, Google Inc.
- TensorFlow: A System for Large-Scale Machine Learning, Google Inc.

Q&A

Thanks