(4) --- 网络基础 & Driver

目录

0x01 引子

在 horovod/runner/launch.py 文件中，_run_static 函数中使用 `driver_service.get_common_interfaces` 来获取路由信息等。

```
def _run_static(args):
    nics = driver_service.get_common_interfaces(settings, all_host_names,
                                                remote_host_names, fn_cache)
```

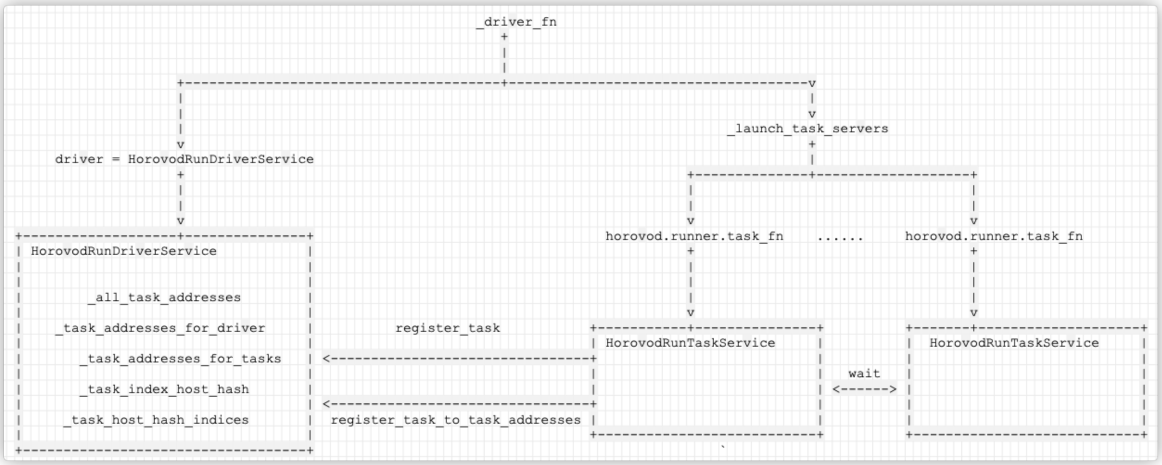
因为这部分比较复杂（ Driver 的概念很类似 Spark 之中 Driver 的概念），所以本文我们单独来分析。

本文的分析问题点是：

- 为什么要知道路由信息？
- 当有多个host时候，horovod如何处理？
- 如何找到路由信息？
- 怎么互相交互？
- （后文会详细分析）SparkDriverService，SparkTaskService，ElasticDriver, Worker 都有什么区别和联系？

本文重点分析 HorovodRunDriverService 和 HorovodRunTaskService 相关。

先给出一个图例，大家可以有些概念。



0x02 总体架构

从注释可知，get_common_interfaces 完成了获得路由信息（所有host之间的共有路由接口集合）的功能，主要是调用 _driver_fn 来完成相关工作。

```
def get_common_interfaces(settings, all_host_names, remote_host_names=None,
fn_cache=None):
    '''
    Find the set of common and routed interfaces on all the hosts.
    '''

    # 得到远端host地址
    if remote_host_names is None:
        remote_host_names = network.filter_local_addresses(all_host_names)

    if len(remote_host_names) > 0:
        if settings.nics: # 如果参数有设定网络接口，就使用
            # If args.nics is provided, we will use those interfaces. All the
workers
            # must have at least one of those interfaces available.
            nics = settings.nics
        else:
            # Find the set of common, routed interfaces on all the hosts (remote
            # and local) and specify it in the args to be used by NCCL. It is
            # expected that the following function will find at least one
interface
            # otherwise, it will raise an exception.

            local_host_names = set(all_host_names) - set(remote_host_names)
            # 获取其他host的网络接口
            nics = _driver_fn(all_host_names, local_host_names, settings,
fn_cache=fn_cache)

    else:
        nics = get_local_interfaces(settings) # 获取本地的网络接口
    return nics
```

2.1 get_local_interfaces

此函数比较简单，目的是获取本地的网络接口。

```
def get_local_interfaces(settings):
    # If all the given hosts are local, find the interfaces with address
    # 127.0.0.1
    nics = set()
    for iface, addrs in net_if_addrs().items():
        if settings.nics and iface not in settings.nics:
            continue
        for addr in addrs:
            if addr.family == AF_INET and addr.address == '127.0.0.1':
                nics.add(iface)
                break

    return nics
```

2.2 _driver_fn

这是本文重点，获取其他host 的网络接口，_driver_fn 的作用是：

- 启动 service 服务；
- 使用 driver.addresses() 获取 Driver 服务的地址（使用`self._addresses = self._get_local_addresses()`完成）；
- 使用 _launch_task_servers（利用 Driver 服务的地址）在每个 worker 之中启动 task 服务，然后 task 服务会在 service 服务中注册；
- 因为是一个环形，每个 worker 会探测 worker index + 1 的所有网络接口；
- 最后 _run_probe 返回一个所有 workers 上的所有路由接口的交集；

代码如下：

这里需要注意的一点是：@cache.use_cache() 的使用：当第一次使用过之后，会把结果放入缓存。

```
@cache.use_cache()
def _driver_fn(all_host_names, local_host_names, settings):
    """
    launches the service service, launches the task service on each worker and
    have them register with the service service. Each worker probes all the
    interfaces of the worker index + 1 (in a ring manner) and only keeps the
    routed interfaces. Function returns the intersection of the set of all the
    routed interfaces on all the workers.
    :param all_host_names: list of addresses. for example,
        ['worker-0','worker-1']
        ['10.11.11.11', '10.11.11.12']
    :type all_host_names: list(string)
    :param local_host_names: host names that resolve into a local addresses.
    :type local_host_names: set
    :param settings: the object that contains the setting for running horovod
    :type settings: horovod.runner.common.util.settings.Settings
    :return: example: ['eth0', 'eth1']
    :rtype: list[string]
    """
    # Launch a TCP server called service service on the host running horovod
    # 启动 service 服务
    num_hosts = len(all_host_names)
    driver = HorovodRunDriverService(num_hosts, settings.key, settings.nics)

    # Have all the workers register themselves with the service service.
    #（利用 Driver 服务的地址）在每个worker之中启动 task 服务，然后task服务会在 service
服务中注册
    _launch_task_servers(all_host_names, local_host_names,
                         driver.addresses(), settings)
    try:
        # 返回一个所有 workers 上的所有路由接口的交集
        return _run_probe(driver, settings, num_hosts)
    finally:
        driver.shutdown()
```

2.3 获取路由接口

我们对 _run_probe 函数做进一步分析。

2.3.1 probe逻辑

_run_probe 函数就是当 所有 task 都启动，注册，probe 环中下一个worker 邻居完成 之后，得到 接口集合。

- 利用 wait_for_initial_registration 等待所有 task 完成注册；
- 对于所有 task，完成 task.notify_initial_registration_complete 通知；
- 利用 driver.wait_for_task_to_task_address_updates 等待 每一个 worker probe 完成；
- 利用 nics.intersection_update 得到接口集合；

```
def _run_probe(driver, settings, num_hosts):
       # wait for all the hosts to register with the service service.

    driver.wait_for_initial_registration(settings.start_timeout)
    tasks = [
        task_service.HorovodRunTaskClient(
            index,
            driver.task_addresses_for_driver(index),
            settings.key,
            settings.verbose) for index in range(
            num_hosts)]
    # Notify all the drivers that the initial registration is complete.
    for task in tasks:
        task.notify_initial_registration_complete()

    # Each worker should probe the interfaces of the next worker in a ring
    # manner and filter only the routed ones -- it should filter out
    # interfaces that are not really connected to any external networks
    # such as lo0 with address 127.0.0.1.
    driver.wait_for_task_to_task_address_updates(settings.start_timeout)

    # Determine a set of common interfaces for task-to-task communication.
    nics = set(driver.task_addresses_for_tasks(0).keys())
    for index in range(1, num_hosts):
        nics.intersection_update(
            driver.task_addresses_for_tasks(index).keys())

    return nics
```

2.3.2 等待函数

probe 利用 wait_for_initial_registration 等待所有 task 完成注册，具体等待函数如下：

```
def wait_for_initial_registration(self, timeout):
    self._wait_cond.acquire()
    try:
        while len(self._all_task_addresses) < self._num_proc:
            self._wait_cond.wait(timeout.remaining())
            timeout.check_time_out_for('tasks to start')
    finally:
        self._wait_cond.release()

def wait_for_task_to_task_address_updates(self, timeout):
    self._wait_cond.acquire()
    try:
        while len(self._task_addresses_for_tasks) < self._num_proc:
            self._wait_cond.wait(timeout.remaining())
            timeout.check_time_out_for(
                'tasks to update task-to-task addresses')
    finally:
        self._wait_cond.release()
```

0x03 基础网络服务

前面提到，Horovod Driver 的概念很类似 Spark 之中 Driver 的概念。Spark应用程序运行时主要分为 Driver 和 Executor，Driver负载总体调度及UI展示，Executor负责Task运行。用户的Spark应用程序运行在Driver上（某种程度上说，用户的程序就是Spark Driver程序），经过Spark调度封装成一个个Task，再将这些Task信息发给Executor执行，Task信息包括代码逻辑以及数据信息，Executor不直接运行用户的代码。

对于 Horovod 来说：

- HorovodRunDriverService 就是 Driver 的实现类。
- HorovodRunTaskService 提供了 Task 部分服务功能，这些 task 需要注册到 HorovodRunDriverService 之中。
- 这套 driver & task 机制的底层由 "基础网络服务" 支撑。

所以我们就仔细分析下基础网络服务。

3.1 继承关系

首先给出继承关系，我们下面讲解的 Driver 服务由 HorovodRunDriverService 提供，Task 服务由 HorovodRunTaskService 提供。

这两个类最终都继承了 network.BasicService。

```
                            network.BasicService

                                  ^    ^
                                  |    |
              +-------------------+    +-------------+
              |                                      |
              +                                      +
driver_service.BasicDriverService       task_service.BasicTaskService
              ^                                      ^
              |                                      |
              |                                      |
              |                                      |
              +                                      +
    HorovodRunDriverService                HorovodRunTaskService
```

3.2 network.BasicService

BasicService 提供了一个网络服务器功能。即通过find_port函数构建了一个`ThreadingTCPServer`，对外提供服务。

```python
class BasicService(object):
    def __init__(self, service_name, key, nics):
        self._service_name = service_name
        self._wire = Wire(key)
        self._nics = nics
        self._server, _ = find_port(
            lambda addr: socketserver.ThreadingTCPServer(
                addr, self._make_handler()))
        self._server._block_on_close = True
        self._port = self._server.socket.getsockname()[1]
        self._addresses = self._get_local_addresses()
        self._thread = in_thread(target=self._server.serve_forever)
```

3.2.1 创建Server

创建服务器代码如下，这里是搜索一个随机端口，然后设置：

```python
def find_port(server_factory):
    min_port = 1024
    max_port = 65536
    num_ports = max_port - min_port
    start_port = random.randrange(0, num_ports)

    for port_offset in range(num_ports):
        try:
            port = min_port + (start_port + port_offset) % num_ports
            addr = ('', port)
            server = server_factory(addr)
            return server, port
        except Exception as e:
            pass

    raise Exception('Unable to find a port to bind to.')
```

3.2.2 Server功能

服务器就是基本的功能，比如获取本server地址，处理 ping，网络交互等。

```
def _make_handler(self):
    server = self

    class _Handler(socketserver.StreamRequestHandler):
        def handle(self):
            try:
                req = server._wire.read(self.rfile)
                resp = server._handle(req, self.client_address)

                # A tuple is the usual response object followed by a utf8 text
stream
                if type(resp) == tuple:
                    (resp, stream) = resp
                    server._wire.write(resp, self.wfile)
                    server._wire.stream(stream, self.wfile)
                else:
                    server._wire.write(resp, self.wfile)
            except (EOFError, BrokenPipeError):
                # Happens when client is abruptly terminated, don't want to
pollute the logs.
                pass

    return _Handler

def _handle(self, req, client_address):
    if isinstance(req, PingRequest):
        return PingResponse(self._service_name, client_address[0])

    raise NotImplementedError(req)

def _get_local_addresses(self):
    result = {}
    for intf, intf_addresses in psutil.net_if_addrs().items():
        if self._nics and intf not in self._nics:
            continue
        for addr in intf_addresses:
            if addr.family == socket.AF_INET:
                if intf not in result:
                    result[intf] = []
                result[intf].append((addr.address, self._port))
    return result

def addresses(self):
    return self._addresses.copy()

def shutdown(self):
    self._server.shutdown()
```

```
        self._server.server_close()
        self._thread.join()

    def get_port(self):
        return self._port
```

3.3 network.BasicClient

HorovodRunDriverClient 和 HorovodRunTaskClient 这两个类都继承了network.BasicClient。

network.BasicClient 的作用就是连接 network.BasicService，与其交互。即 network.BasicClient 是一个操作接口。

```
                             network.BasicClient

                                ^            ^
                                |            |
             +------------------+            +---------------+
             |                                               |
             +                                               |
                                                             +
driver_service.BasicDriverClient               task_service.BasicTaskClient

             ^                                               ^
             |                                               |
             |                                               |
             +                                               +
   HorovodRunDriverClient                           HorovodRunTaskClient
```

两个主要 API 如下：

3.3.1 _probe

_probe 获取 server 的网络接口。

```
def _probe(self, addresses):
    result_queue = queue.Queue()
    threads = []
    for intf, intf_addresses in addresses.items():
        for addr in intf_addresses:
            thread = in_thread(target=self._probe_one, args=(intf, addr,
result_queue))
            threads.append(thread)
    for t in threads:
        t.join()

    result = {}
    while not result_queue.empty():
        intf, addr = result_queue.get()
        if intf not in result:
            result[intf] = []
        result[intf].append(addr)
    return result
```

3.3.2 发送消息

_send 的作用是给server发送消息。

```
def _send(self, req, stream=None):
    """
    Sends the request and returns the response object.
    Streaming data response is transferred to the optional stream parameter.
    """
    # Since all the addresses were vetted, use the first one.
    addr = list(self._addresses.values())[0][0]
    return self._send_one(addr, req, stream)
```

3.4 总结

我们可以看到，network.BasicService 会提供了一个server，这个 Service 都是通过 network.BasicClient 来访问。基于此，Horovod 的HorovodRunDriverService 和 HorovodRunTaskService 这两个类就可以互相交互，进行沟通。

0x04 Driver 服务

Driver 服务由 HorovodRunDriverService 提供，其功能主要是维护维护各种 task 地址以及相应关系。具体各种 task 地址 就是 Task 服务 来注册的。

需要注意的是：HorovodRunDriverService 和 HorovodRunTaskService 都最终继承了 network.BasicService，他们之间可以是异地运行交互。

4.1 HorovodRunDriverService

HorovodRunDriverService 是对 BasicDriverService 的封装。

HorovodRunDriverClient 是 其 访问接口。

```
class HorovodRunDriverService(driver_service.BasicDriverService):
    NAME = 'horovod driver service'

    def __init__(self, num_hosts, key, nics):
        super(HorovodRunDriverService, self).__init__(num_hosts,

 HorovodRunDriverService.NAME,
                                                     key, nics)

class HorovodRunDriverClient(driver_service.BasicDriverClient):
    def __init__(self, driver_addresses, key, verbose, match_intf=False):
        super(HorovodRunDriverClient, self).__init__(
            HorovodRunDriverService.NAME,
            driver_addresses,
            key,
            verbose,
            match_intf=match_intf)
```

4.2 BasicDriverService

BasicDriverService基类 主要就是 维护各种 task 地址以及相应关系。

```
class BasicDriverService(network.BasicService):
    def __init__(self, num_proc, name, key, nics):
        super(BasicDriverService, self).__init__(name, key, nics)
        self._num_proc = num_proc
        self._all_task_addresses = {}
        self._task_addresses_for_driver = {}
        self._task_addresses_for_tasks = {}
        self._task_index_host_hash = {}
        self._task_host_hash_indices = {}
        self._wait_cond = threading.Condition()
```

这里的各种 task 地址就是 Task 服务 注册到 Driver 的数值。

可以看到里面有各种关于地址的变量，为了让大家理解这些变量的作用，对于每一个变量我们举例如下（这里有些变量是专门为 spark 设计，都放到基类里面有点奇怪）：

4.2.1 _all_task_addresses

本变量是记录了所有 task 的地址，变量举例如下：

```
self._all_task_addresses = {
  1: {
    'lo' : [('1.1.1.1', 12345)],
        'eth0' : [('10.10.10.01', 12345)]
    },
  0: {
    'lo' : [('2.2.2.2', 54321)],
        'eth0' : [('10.10.10.02', 54321)]
    }
}
```

本变量由 task 调用 RegisterTaskRequest 来注册。

```
if isinstance(req, RegisterTaskRequest):
    self._wait_cond.acquire()
    try:
        assert 0 <= req.index < self._num_proc
        self._all_task_addresses[req.index] = req.task_addresses
```

使用时候，有几个方式，举例如下：

比如 all_task_addresses 获取 self._all_task_addresses[index].copy() 来决定 ping /check 的下一个跳转。

4.2.2 _task_addresses_for_driver

本变量是记录了所有 task 的地址，但是网卡接口有多种，这里选择与 本 driver 地址匹配的地址。

变量举例如下：

```
self._task_addresses_for_driver = {
  1: {
        'eth0' : [('10.10.10.01', 12345)]
    },
  0: {
        'eth0' : [('10.10.10.02', 54321)]
    }
}
```

本变量由 task 调用 RegisterTaskRequest 来注册。

```
# Just use source address for service for fast probing.
self._task_addresses_for_driver[req.index] = \
    self._filter_by_ip(req.task_addresses, client_address[0])
```

具体使用举例如下：

```
def task_addresses_for_driver(self, index):
    self._wait_cond.acquire()
    try:
        return self._task_addresses_for_driver[index].copy()
    finally:
        self._wait_cond.release()
```

driver用这个地址来生成 其内部 task 变量。

```
tasks = [
    task_service.HorovodRunTaskClient(
        index,
        driver.task_addresses_for_driver(index),
        settings.key,
        settings.verbose) for index in range(
        num_hosts)]
```

4.2.3 _task_addresses_for_tasks

该变量举例如下：

```
self._task_addresses_for_tasks = {
  1: {
        'eth0' : [('10.10.10.01', 12345)]
    },
  0: {
        'eth0' : [('10.10.10.02', 54321)]
    }
}
```

本变量由RegisterTaskToTaskAddressesRequest注册。

```
if isinstance(req, RegisterTaskToTaskAddressesRequest):
    self.register_task_to_task_addresses(req.index, req.task_addresses)
    return network.AckResponse()

def register_task_to_task_addresses(self, index, task_addresses):
    self._wait_cond.acquire()
    try:
        assert 0 <= index < self._num_proc
        self._task_addresses_for_tasks[index] = task_addresses # 这里赋值
    finally:
        self._wait_cond.notify_all()
        self._wait_cond.release()
```

该变量被 task 用来获取 某个 task 的一套网络接口，比如：

```
# Determine a set of common interfaces for task-to-task communication.
nics = set(driver.task_addresses_for_tasks(0).keys())
```

4.2.4 _task_index_host_hash

每一个 task 有一个对应的 host hash，该数值被 MPI 作为 host name 来操作。

```
self._task_index_host_hash = {
  1: {
        'ip-10-10-10-01-dfdsfdsfdsfdsf2'
    },
  0: {
        'ip-10-10-10-02-treterwrtqwer'
    }
}
```

具体使用如下。这个函数是 spark 相关会使用，具体是逐一通知 spark task 进入下一阶段。

```
def task_indices(self):
    self._wait_cond.acquire()
    try:
        return list(self._task_index_host_hash.keys())
    finally:
        self._wait_cond.release()
```

或者使用如下，是为了获取某一个 host 对应的 `host hash name`。

```
def task_index_host_hash(self, index):
    self._wait_cond.acquire()
    try:
        assert 0 <= index < self._num_proc
        return self._task_index_host_hash[index]
    finally:
        self._wait_cond.release()
```

4.2.5 _task_host_hash_indices

该变量举例如下：

```
self._task_host_hash_indices = {
  {
          'ip-10-10-10-01-dfdsfdsfdsfdsf2' : [1]
    },
  {
          'ip-10-10-10-02-treterwrtqwer' : [0]
    }
}
```

具体是在注册 RegisterTaskRequest 时候生成。

```
self._task_host_hash_indices[req.host_hash].append(req.index)
```

使用具体代码是：

```
def task_host_hash_indices(self):
    self._wait_cond.acquire()
    try:
        return self._task_host_hash_indices.copy()
    finally:
        self._wait_cond.release()
```

具体是被 rsh 使用。rsh 就是在某一个 host 上，让某一个 horovod rank 启动。具体逻辑是：

- 获取某一个 host 上所有的 task indices ;
- 利用 task_host_hash_indices 取出本进程 local rank 对应的 task index;
- 取出在 driver 中 task index 对应保持的 task address;
- 最后依据这个 task addresses 生成一个 SparkTaskClient，进行后续操作。

```
driver_client = driver_service.SparkDriverClient(driver_addresses, key,
verbose=verbose)
task_indices = driver_client.task_host_hash_indices(host_hash)
task_index = task_indices[local_rank]
task_addresses = driver_client.all_task_addresses(task_index)
task_client = task_service.SparkTaskClient(task_index, task_addresses, key,
verbose=verbose)
task_client.stream_command_output(stdout, stderr)
task_client.run_command(command, env,
                        capture_stdout=stdout is not None,
                        capture_stderr=stderr is not None,

 prefix_output_with_timestamp=prefix_output_with_timestamp)
```

4.3 总体逻辑

总体逻辑如下：

```
                        network.BasicService

                             ^    ^
```

```
                                     |    |
                 +-------------------+   +-------------+
                 |                                     |
                 +                                     +
   driver_service.BasicDriverService       task_service.BasicTaskService
                 ^                                     ^
                 |                                     |
                 |                                     |
                 |                                     |
                 |                                     +
+----------------+------------------+     HorovodRunTaskService
| HorovodRunDriverService           |
|                                   |
|                                   |
|        _all_task_addresses        |
|                                   |
|    _task_addresses_for_driver     |
|                                   |
|       _task_addresses_for_tasks   |
|                                   |
|       _task_index_host_hash       |
|                                   |
|     _task_host_hash_indices       |
|                                   |
+-----------------------------------+
```

0x05 Task 服务

HorovodRunTaskService 提供了 Task 部分服务功能。整体逻辑是由几个函数共同完成。

5.1 启动具体服务

_launch_task_servers 用来启动具体服务，其主要作用是：多线程运行，在每一个线程中，远程运行 `horovod.runner.task_fn`。

其中：

- 传入参数中，all_host_names 就是程序启动时候配置的所有host，比如 ["1.1.1.1", "2.2.2.2"]；
- 使用了我们之前提到的 safe_shell_exec.execute 完成了安全运行保证；
- 使用我们前文提到的 get_remote_command 完成了远程命令的获取，即在命令之前加上了 `ssh -o PasswordAuthentication=no -o StrictHostKeyChecking=no` 等等配置；
- 最终每个启动的命令举例如下： `ssh -o PasswordAuthentication=no -o StrictHostKeyChecking=no 1.1.1.1 python -m horovod.runner.task_fn xxxxxxx`；
- 使用 execute_function_multithreaded 在每一个 host 上运行，启动 task 服务；

具体代码如下：

```
def _launch_task_servers(all_host_names, local_host_names, driver_addresses,
                         settings):
    """
    Executes the task server and service client task for registration on the
    hosts.
    :param all_host_names: list of addresses. for example,
        ['worker-0','worker-1']
        ['10.11.11.11', '10.11.11.12']
    :type all_host_names: list(string)
    :param local_host_names: names that are resolved to one of the addresses
    of local hosts interfaces. For example,
        set(['localhost', '127.0.0.1'])
    :type local_host_names: set
    :param driver_addresses: map of interfaces and their address and port for
    the service. For example:
        {
            'lo': [('127.0.0.1', 34588)],
            'docker0': [('172.122.10.1', 34588)],
            'eth0': [('11.111.33.73', 34588)]
        }
    :type driver_addresses: map
    :param settings: the object that contains the setting for running horovod
    :type settings: horovod.runner.common.util.settings.Settings
    :return:
    :rtype:
    """

    def _exec_command(command):
        host_output = io.StringIO()
        try:
            # 完成了安全运行保证
            exit_code = safe_shell_exec.execute(command,
                                                stdout=host_output,
                                                stderr=host_output)
        finally:
            host_output.close()
        return exit_code

    args_list = []
    num_hosts = len(all_host_names)
    for index in range(num_hosts):
        host_name = all_host_names[index] # all_host_names 就是程序启动时候配置的所有
host, 比如 ["1.1.1.1", "2.2.2.2"]
        command = \
            '{python} -m horovod.runner.task_fn {index} {num_hosts} ' \
            '{driver_addresses} {settings}' \
            .format(python=sys.executable,
                    index=codec.dumps_base64(index),
                    num_hosts=codec.dumps_base64(num_hosts),
                    driver_addresses=codec.dumps_base64(driver_addresses),
                    settings=codec.dumps_base64(settings))
        if host_name not in local_host_names:
            # 完成了远程命令的获取, 即在命令之前加上了 `ssh -o
PasswordAuthentication=no -o StrictHostKeyChecking=no`等等配置
            command = get_remote_command(command,
                                         host=host_name,
                                         port=settings.ssh_port,

identity_file=settings.ssh_identity_file)

        args_list.append([command])

    # Each thread will use ssh command to launch the server on one task. If an
    # error occurs in one thread, entire process will be terminated. Otherwise,
    # threads will keep running and ssh session -- and the the task server --
    # will be bound to the thread. In case, the horovod process dies, all
    # the ssh sessions and all the task servers will die as well.
    # 使用 execute_function_multithreaded 在每一个 host 上运行，启动 task 服务
    threads.execute_function_multithreaded(_exec_command,
                                           args_list,
                                           block_until_all_done=False)
```

5.2 具体服务逻辑

上段有：`{python} -m horovod.runner.task_fn {index} {num_hosts} {driver_addresses} {settings}`执行具体服务逻辑，所以我们介绍下 `horovod.runner.task_fn`。

`_task_fn` 函数完成了

- 生成了 HorovodRunTaskService 实例，赋值给 task；
- 使用 `HorovodRunDriverClient . register_task` 来向 Driver 服务注册task（自己）的地址；
- 使用 `HorovodRunDriverClient . register_task_to_task_addresses` 来向 Driver 服务注册自己在Ring上 下一个邻居的地址；
- 每一个 task 都做这个操作，最后就得到了在这个 ring cluster 之中的一个路由接口；

具体代码如下：

```
def _task_fn(index, num_hosts, driver_addresses, settings):

    task = task_service.HorovodRunTaskService(index, settings.key,
settings.nics)
    try:
        driver = driver_service.HorovodRunDriverClient(
            driver_addresses, settings.key, settings.verbose)
        # 向 Driver 服务注册task（自己）的地址
        driver.register_task(index,
                             task.addresses(),
                             host_hash.host_hash())
        task.wait_for_initial_registration(settings.start_timeout)
        # Tasks ping each other in a circular fashion to determine interfaces
        # reachable within the cluster.
        next_task_index = (index + 1) % num_hosts
        next_task_addresses = driver.all_task_addresses(next_task_index)
        # We request interface matching to weed out all the NAT'ed interfaces.
        next_task = task_service.HorovodRunTaskClient(
            next_task_index,
            next_task_addresses,
            settings.key,
            settings.verbose,
            match_intf=True,
            attempts=10)
        # 向 Driver 服务注册自己在Ring上 下一个邻居的地址
        driver.register_task_to_task_addresses(next_task_index,
                                               next_task.addresses())
        # Notify the next task that the address checks are completed.
        next_task.task_to_task_address_check_completed()
        # Wait to get a notification from previous task that its address checks
        # are completed as well.

 task.wait_for_task_to_task_address_check_finish_signal(settings.start_timeout)

    finally:
        task.shutdown()
```

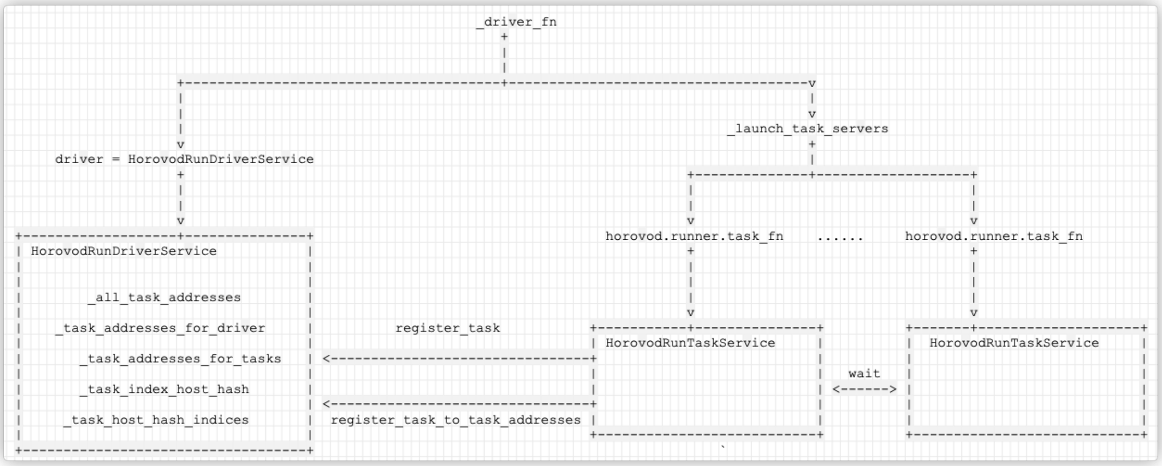
```
if __name__ == '__main__':
    index = codec.loads_base64(sys.argv[1])
    num_hosts = codec.loads_base64(sys.argv[2])
    driver_addresses = codec.loads_base64(sys.argv[3])
    settings = codec.loads_base64(sys.argv[4])

    _task_fn(index, num_hosts, driver_addresses, settings)
```

5.3 HorovodRunTaskService

HorovodRunTaskService 主要的作用是提供了两个等待函数。因为具体路由操作是需要彼此通知，所以需要互相等待。

```
class HorovodRunTaskService(task_service.BasicTaskService):
    NAME_FORMAT = 'horovod task service #%d'

    def __init__(self, index, key, nics):
        super(HorovodRunTaskService, self).__init__(
            HorovodRunTaskService.NAME_FORMAT % index,
            index, key, nics)
        self.index = index
        self._task_to_task_address_check_completed = False

    def _handle(self, req, client_address):

        if isinstance(req, TaskToTaskAddressCheckFinishedSignal):
            self._wait_cond.acquire()
            try:
                self._task_to_task_address_check_completed = True
            finally:
                self._wait_cond.notify_all()
                self._wait_cond.release()

            return TaskToTaskAddressCheckFinishedSignalResponse(self.index)

        return super(HorovodRunTaskService, self)._handle(req, client_address)

    def wait_for_task_to_task_address_check_finish_signal(self, timeout):
        self._wait_cond.acquire()
        try:
            while not self._task_to_task_address_check_completed:
                self._wait_cond.wait(timeout.remaining())
                timeout.check_time_out_for('Task to task address check')
        finally:
            self._wait_cond.release()


class HorovodRunTaskClient(task_service.BasicTaskClient):

    def __init__(self, index, task_addresses, key, verbose, match_intf=False,
attempts=3):
        super(HorovodRunTaskClient, self).__init__(
            HorovodRunTaskService.NAME_FORMAT % index,
            task_addresses, key, verbose,
            match_intf=match_intf,
            attempts=attempts)
        self.index = index

    def task_to_task_address_check_completed(self):
        resp = self._send(TaskToTaskAddressCheckFinishedSignal(self.index))
        return resp.index
```

逻辑如下：



0x06 总结

本文总结如下：

- 因为 Horovod 分布式训练 涉及到多个 hosts，所以如果要彼此访问，需要知道路由信息；
- 当所有 task 都启动，注册，probe 环中下一个worker 邻居完成 之后，DriverService 会得到路由信息（所有host之间的共有路由接口集合），返回给 Horovod 主体部分使用；
- network.BasicService 提供了网络服务功能；
- XXXService 都是通过 XXXClient作为接口才能访问；
- HorovodRunDriverService 和 HorovodRunTaskService 都最终继承了 network.BasicService，他们之间可以是异地运行交互。
- HorovodRunTaskService 提供了 Task 部分服务功能，这些 task 需要注册到 Driver 之中（和Spark思路类似）。
- HorovodRunDriverService 是对 BasicDriverService 的封装。

  BasicDriverService 就是 维护各种 task 地址以及相应关系

  ，比如：

  - _all_task_addresses ： 记录了所有 task 的地址；
  - _task_addresses_for_driver ： 记录了所有 task 的地址，但是因为网卡接口有多种，这里选择与 本driver 地址匹配的地址；
  - _task_addresses_for_tasks ： 用来给某一个 task 分配一个地址，同时获取本 task 的一套网络接口；
  - _task_index_host_hash ： 每一个 task 有一个对应的 host hash。这个函数是 spark 相关会使用，具体是逐一通知 spark task 进入下一阶段。或者是为了获取某一个 host 对应的 host hash name；
  - _task_host_hash_indices ： 具体是被 rsh 使用，由 rank 得到 在 driver 中 task index 对应保持的 task address；
- SparkDriverService，SparkTaskService，ElasticDriver, Worker 都有什么区别和联系？
  - HorovodRunDriverService 这里只是用来得到路由信息，记录各种 Task 地址；

- SparkDriverService 除了记录路由和地址之外，还提交执行任务（Command），因为具体在哪一个Spark Executor启动之后，SparkDriverService 就需要知道 对应 SparkTaskService 的地址，这样才能知道提交到哪里；
- SparkTaskService 负责执行命令（抛弃了Spark Executor的逻辑，自己搞了一套），就是从 SparkDriverService 那里获得训练函数，然后启动 python 进程来执行；
- ElasticDriver 做得更多，因为还有弹性，需要容错；