

# Lab4报告

---

x-WowKiddy小组lab4实验报告。

本次实验选择Ray进行部署、测试、分析及评价

## 1 测试任务选定

---

本次实验模拟数据库同时高并发查找，同时对数组产生100000次查找请求。通过对比顺序查找，使用ray分布式计算查找测试相关指标。

## 2 Ray性能指标

---

- 资源使用率（如内存占用，CPU占用率等）
- 吞吐率（单位时间处理的任务数）
- 响应时间（任务从提交到完成的时间）
- 任务提交延迟（任务提交的延迟）
- 计算集群节点个数（由于Ray Launcher会自动对节点进行调整，故该指标也是较为重要的性能指标）

在测试任务中，我们将对响应时间、吞吐率、任务提交延迟进行测试及优化。

## 3 Ray单机版部署及代码编写

---

### 3.1 Ray的安装

Ubuntu20.04下的安装步骤

1.检查python版本：尽量安装python最新版本，同时确保header和worker的python版本相同。

2.如有pip，跳过此步，否则按如下安装pip：

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
sudo python3 get-pip.py
```

3.更新

```
sudo apt update
```

4.安装ray最新发行版

```
pip install -U ray
pip install 'ray[default]' #美化cli界面
```

## 3.2 Ray Cluster搭建

使用命令：

```
ray start --head --port=6379
```

创建head节点。

如果要使用ray dashboard并在其他主机上查看，请使用如下命令启动ray：

```
ray start --head --port=6379 --include-dashboard=true --dashboard-host=0.0.0.0 --
dashboard-port=8265
```

输入header节点创立后，命令行提示中Next steps下面的第二行，

```
ray start --address='172.30.239.56:6379' --redis-password='5241590000000000' #视实际情况
修改address
```

如果要退出集群，只需

```
ray stop
```

如果在启动ray头节点时出现问题，请先使用如下命令：

```
sudo service redis stop
```

之后再使用

```
ray start --head --port=6379
```

至此，Ray单机版部署完成。

通过Web端访问Ray Dashboard界面如下：

Ray Dashboard

TRY EXPERIMENTAL DASHBOARD

MACHINE VIEWLOGICAL VIEWMEMORYRAY CONFIG

☒ Group by host

	Host	PID	Uptime (s)	CPU	RAM	Plasma	Disk	Sent	Received	Logs	Errors
—	MacBook-Pro.local (127.0.0.1)	0 workers / 16 cores	3d 00h 18m 02s	5.4%	7.2 GiB / 16.0 GiB (45%)	0.0 MiB / 2048.0 MiB	627.8 GiB / 931.5 GiB (67%)	0.0 MiB/s	0.0 MiB/s	No logs	No errors
🔌	Totals (1 host)	0 workers / 16 cores		5.4%	7.2 GiB / 16.0 GiB (45%)	0.0 MiB / 2048.0 MiB	627.8 GiB / 931.5 GiB (67%)	0.0 MiB/s	0.0 MiB/s	No logs	No errors

Last updated: 2022/6/28 下午8:44:42

### 3.3 基于Ray API的Python测试代码编写及初步测试

这里我们对三个指标进行了测试，分别是：

- 响应时间
- 吞吐率
- 任务提交延迟

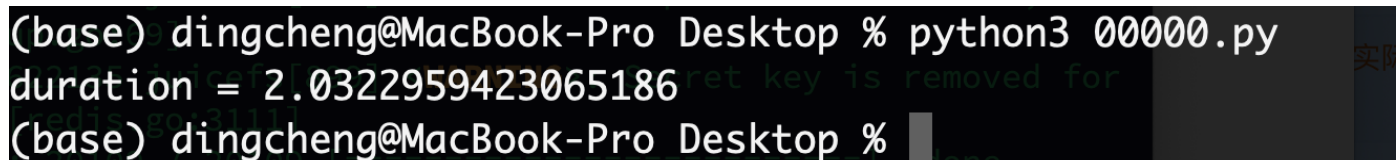
首先放出我们用于对比的顺序查找代码：

```
import ray
import time
import numpy as np
import random
import string
from tqdm import tqdm

def search(array, n):
    res = -1
    for i in range(len(array)):
        if array[i] == n:
            res = i
            break
    return res

if __name__ == '__main__':
    array = np.random.randint(0,1000,size=100)
    array = list(array)
    tic = time.time()
    results = [search(array, random.randint(0,1000)) for x in range(100000)]
    duration = (time.time() - tic)
    print("duration = "+ str(duration))
```

这个函数模拟对100000条数据的查找。在命令行运行该程序，可以得到对应的时间：



```
(base) dingcheng@MacBook-Pro Desktop % python3 00000.py
duration = 2.0322959423065186
(base) dingcheng@MacBook-Pro Desktop %
```

下面是我们基于Ray API编写的程序：

```
import ray
import time
import numpy as np
import random
import string
```

```

from tqdm import tqdm
import sys

@ray.remote
def search(array, n):
    res = -1
    for i in range(len(array)):
        if array[i] == n:
            res = i
            break
    return res

if __name__ == '__main__':
    ray.init(num_cpus=10, ignore_reinit_error=True)
    array = np.random.randint(0, 1000, size=100)
    array = list(array)
    start = time.time()
    result_ids = [search.remote(array, random.randint(0, 1000)) for x in
range(int(sys.argv[1]))]
    results = ray.get(result_ids)
    print("duration = " + str(time.time()-start))

```

该程序同样模拟100000次对数据的查找，与不使用Ray的顺序查找不同，我们使用ray.remote修饰查找函数，让ray自身创建100000个worker进行并行查找。

在命令行运行程序可以得到：

```

(base) dingcheng@MacBook-Pro Desktop % python3 11111.py 100000
2022-06-28 20:59:31,979 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265
duration = 99.82111620903015
(base) dingcheng@MacBook-Pro Desktop %

```

可以发现，基于Ray API的分布式计算代码成功运行。

我们同样做了对吞吐率的测试，通过让基于Ray的python代码执行1000、10000、100000次查找请求，可以得到其响应时间：

```

(base) dingcheng@MacBook-Pro Desktop % python3 11111.py 1000
2022-06-28 21:31:54,645 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265
duration = 0.9705557823181152
(base) dingcheng@MacBook-Pro Desktop % python3 11111.py 10000
2022-06-28 21:32:10,245 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265
duration = 9.88011121749878
(base) dingcheng@MacBook-Pro Desktop % python3 11111.py 100000
2022-06-28 21:32:35,960 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265
duration = 97.25204205513
(base) dingcheng@MacBook-Pro Desktop %

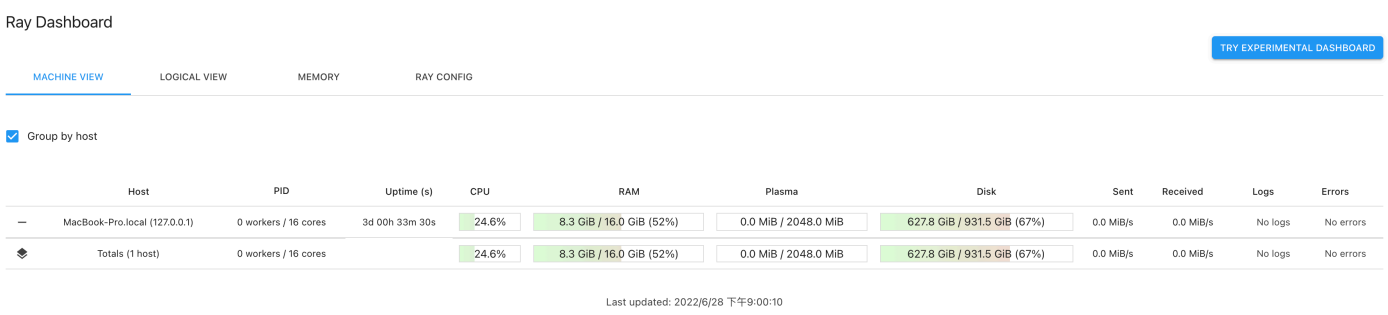
```

可以看到，代码运行时间与查找请求执行次数大致成线性关系，说明在代码结构不变的情况下，吞吐率基本恒定不变。

然后我们对该代码创建不同数目的远程任务的延迟进行了测试，即调用remote()方法所消耗的时间，分别测试了1000、10000、100000次任务提交：

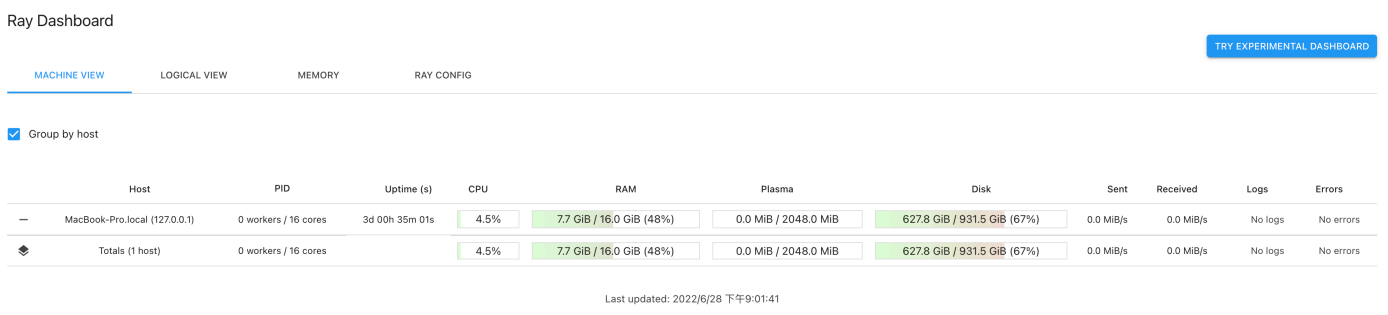
```
(base) dingcheng@MacBook-Pro Desktop % python3 11111.py 1000
2022-06-28 21:45:25,512 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265
Task submit duration = 0.9623451232910156
(base) dingcheng@MacBook-Pro Desktop % python3 11111.py 10000
2022-06-28 21:45:44,456 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265
Task submit duration = 9.653314113616943
(base) dingcheng@MacBook-Pro Desktop % python3 11111.py 100000
2022-06-28 21:46:13,144 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265
Task submit duration = 95.31101059913635
(base) dingcheng@MacBook-Pro Desktop %
```

在代码运行过程中打开Ray Dashboard，可以实时监测到计算节点的资源占用率：



可以看到，在大量任务并行时，CPU占用率显著提升。

在计算完成后，CPU占用率回到正常水平：



然而，我们发现分布式计算代码的时间却相比顺序运行慢了数十倍，这与我们的认知相违背，因此我们进入分析测试与优化部分——

## 4 Ray单机版分析、测试、优化

调用Ray API对可以并行的任务进行分布式计算，理论上应该会极大地提升速度，降低响应时间。但是经过我们上面的测试可以发现，事实上其响应时间是顺序运行的数十倍。

在Ray的工作过程中，它需要对每一个有着@ray.remote函数修饰符的函数创建一个远程任务，同时通过其内置的调度模块将这些远程任务分配给worker。

这里的问题是每个任务调用都有一个无法忽略的开销（例如，调度，进程间通信，更新系统状态），这个开销占据了执行任务所需的实际时间。

在上述测试过程中我们也可以发现，创建远程任务，即调用remote()方法实际上占去了运行的大部分时间。

而为了解决此问题，我们需要改变程序架构，将很多耗时很小的任务聚合成一个较大的任务，这样可以降低创建远程任务时的开销，从而提升计算速度。

事实上，经过测试，使用这种方法可以非常显著地提升运算速度。

以下是我们经过重构过的代码：

```
import ray
import time
import numpy as np
import random
import string
from tqdm import tqdm

ray.init(num_cpus = 4)

def tiny_work(array, n):
    res = -1
    for i in range(len(array)):
        if array[i] == n:
            res = i
            break
    return res

@ray.remote
def mega_work(array, n, start, end):

    return [tiny_work(array, n) for x in range(start, end)]

array = np.random.randint(0,1000,size=100)
array = list(array)
start = time.time()

result_ids = []

[result_ids.append(mega_work.remote(array,random.randint(0,1000), x*100, (x+1)*100))
 for x in range(1000)]

results = ray.get(result_ids)
```

```
print("duration = " + str(time.time() - start))
```

该程序将每100次查找请求聚合成一个任务，通过1000次并行调用该任务达到100000次查找的效果。

在命令行运行该程序，可以得到：

```
2022-06-28 21:24:11,704 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265  
duration = 1.0609259605407715  
(base) dingcheng@MacBook-Pro Desktop %
```

可以发现此时的响应时间已经极大地降低了，缩短到了顺序查找时间的1/2。

接下来，我们将基于目前的部署进行代码结构的调整，以提升其性能。

通过调参与测试，我们发现将每1000次查找聚合成一个任务时，响应时间降到最低：

```
import ray
import time
import numpy as np
import random
import string
from tqdm import tqdm

ray.init(num_cpus = 4)

def tiny_work(array, n):
    res = -1
    for i in range(len(array)):
        if array[i] == n:
            res = i
            break
    return res

@ray.remote
def mega_work(array, n, start, end):

    return [tiny_work(array, n) for x in range(start, end)]

array = np.random.randint(0,1000,size=100)
array = list(array)
start = time.time()

result_ids = []

[result_ids.append(mega_work.remote(array,random.randint(0,1000), x*1000, (x+1)*1000))
 for x in range(100)]

results = ray.get(result_ids)

print("duration = " + str(time.time() - start))
```

```
(base) dingcheng@MacBook-Pro Desktop % python3 22222.py
2022-06-28 21:26:38,819 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265果。
duration = 0.5344350337982178
(base) dingcheng@MacBook-Pro Desktop %
```

可以看到，响应时间性能相对初始部署提升约50%。

由于在代码结构相同时，吞吐率和响应时间成正比，因此吞吐率相对初始部署也提升了50%。

测试其任务提交延迟：

```
(base) dingcheng@MacBook-Pro Desktop % python3 22222.py
2022-06-28 21:58:10,414 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265
task submit duration = 1.0111627578735352
(base) dingcheng@MacBook-Pro Desktop % python3 22222.py
2022-06-28 21:59:22,406 INFO services.py:1412 -- View the Ray dashboard at http://127.0.0.1:8265
task submit duration = 0.10447406768798828
(base) dingcheng@MacBook-Pro Desktop %
```

对比优化前后的任务提交延迟可以发现，原本对100个查找进行聚合，需要创建1000个远程任务，而优化之后对1000个查找进行聚合，只需创建100个远程任务，任务提交延迟降低到了原来的1/10。

## 4 Ray分布式部署及测试

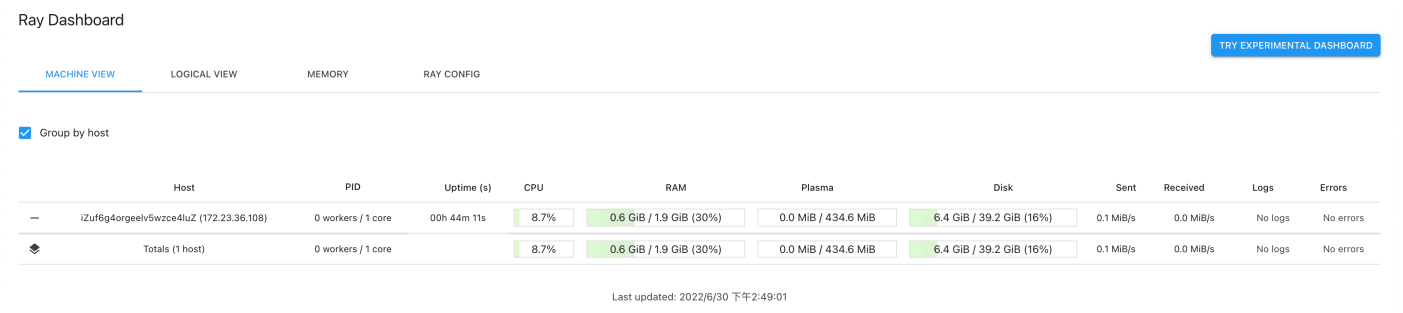
由于Ray的Slave节点需要和Master节点在同一局域网下，这里使用两台阿里云服务器搭建Ray集群。（同一区域的阿里云服务器位于同一局域网下）

首先在一台服务器上运行命令：

```
ray start --head --port=6379 --include-dashboard=true --dashboard-host=0.0.0.0 --
dashboard-port=8265
```

启动Master节点并将dashboard设置为任何ip都可以访问

打开dashboard可以看到：



此时Ray集群内只有ip为172.23.36.108的Master节点。

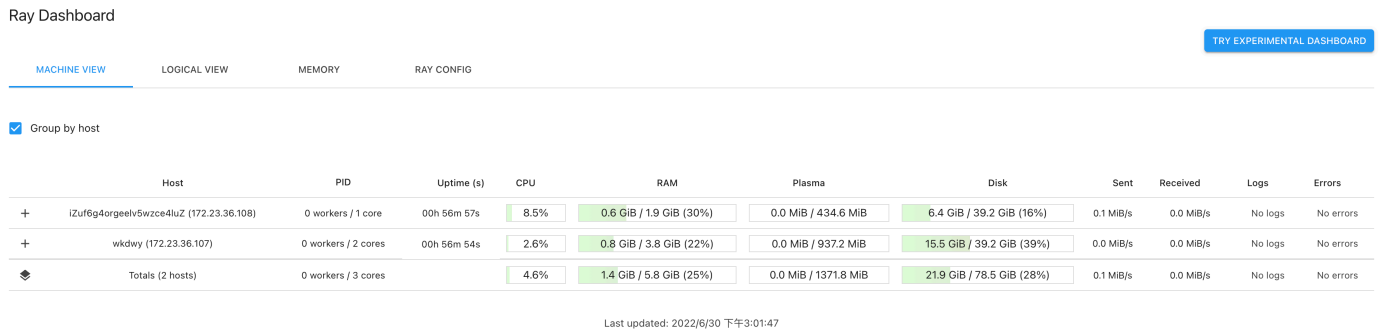
之后在另一个服务器上运行命令：



```
ray start --address='172.23.36.108:6379'
```

连接到Master节点。

之后在Dashboard可以看到：



此时Ray集群中有ip为172.23.36.108的Master节点和ip为172.23.36.107的Slave节点。

至此，Ray分布式部署完毕，下面对分布式的性能进行测试。

首先将原本单机部署的测试代码移植到服务器上，并进行适当修改：

```
import ray
import time
import numpy as np
import random
import string
from tqdm import tqdm
import sys

@ray.remote
def search(array, n):
    res = -1
    for i in range(len(array)):
        if array[i] == n:
            res = i
            break
    return res

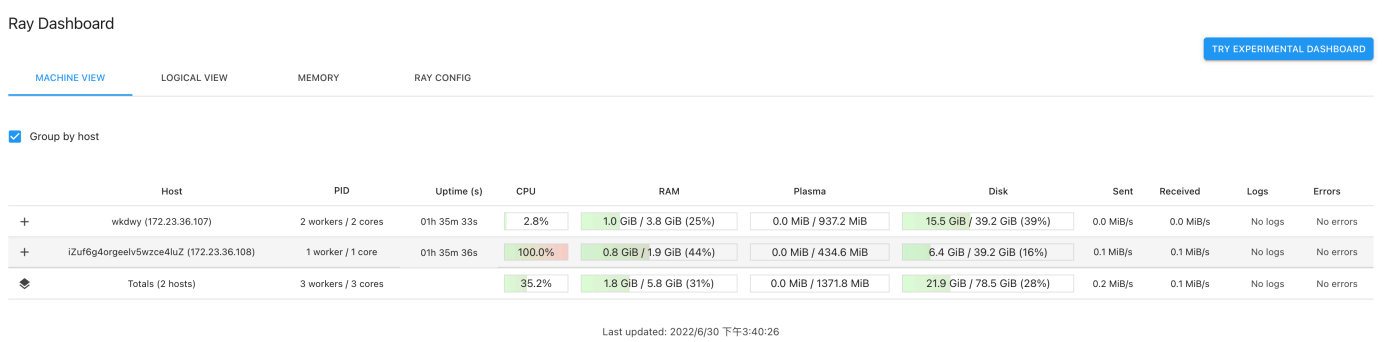
if __name__ == '__main__':
    ray.init(address='auto')
    array = np.random.randint(0,1000,size=100)
    array = list(array)
    start = time.time()
    result_ids = [search.remote(array, random.randint(0,1000)) for x in
range(int(sys.argv[1]))]
    results = ray.get(result_ids)
    print("duration = "+ str(time.time()-start))
```

由于此处使用ray.init连接到已存在的Ray Cluster，因此去掉ray.init中对num\_cpus的指定

之后运行程序，模拟100000次请求，每一个请求作为一个远程任务时的并发。

```
root@iZuf6g4orgeelv5wzce4luZ:~# python3 test.py 100000
```

此时查看Dashboard可以看到Ray集群的资源使用情况



```
root@iZuf6g4orgeelv5wzce4luZ:~# python3 test.py 100000
duration = 241.62525296211243
root@iZuf6g4orgeelv5wzce4luZ:~# python3 test.py 10000
duration = 22.268905639648438
root@iZuf6g4orgeelv5wzce4luZ:~# python3 test.py 1000
duration = 2.238037347793579
root@iZuf6g4orgeelv5wzce4luZ:~#
```

可以看到，响应时间与任务数成正比，在代码结构不变时，吞吐量基本恒定不变。

之后将单机版的优化代码移植到服务器上：

```
import ray
import time
import numpy as np
import random
import string
from tqdm import tqdm

ray.init(address='auto')
def tiny_work(array, n):
    res = -1
    for i in range(len(array)):
        if array[i] == n:
            res = i
            break
    return res

@ray.remote
def mega_work(array, n, start, end):

    return [tiny_work(array, n) for x in range(start, end)]
```

```
array = np.random.randint(0,1000,size=100)
array = list(array)
start = time.time()
result_ids = []
[result_ids.append(mega_work.remote(array,random.randint(0,1000), x*1000, (x+1)*1000))
 for x in range(100)]

results = ray.get(result_ids)

print("duration = " + str(time.time() - start))
```

运行该程序进行测试：

```
root@iZuf6g4orgeelv5wzce4luZ:~# vim test_op.py
root@iZuf6g4orgeelv5wzce4luZ:~# python3 test_op.py
duration = 1.5772817134857178
root@iZuf6g4orgeelv5wzce4luZ:~#
```

可以看到速度相比每一个请求作为一个任务有了极大的提升。

至此，Ray分布式部署及测试完毕。

## 5 基于Docker的Ray部署

基于Docker的Ray部署使用DockerHub中Ray官方发布的源：

<https://hub.docker.com/r/rayproject/ray>

使用如下命令下载Docker镜像：

```
docker pull rayproject/ray
```

可以看到命令行出现如下信息：

```
root@iZuf6g4orgeelv5wzce4luZ:~# docker pull rayproject/ray
Using default tag: latest
latest: Pulling from rayproject/ray
d5fd17ec1767: Pull complete
341eeba1871f: Pull complete
913d7f86391e: Pull complete
2107148d3c4b: Pull complete
d0a777325523: Pull complete
1dc2e272e6b0: Pull complete
df3e0297f017: Pull complete
da37cde4b300: Pull complete
9d617ad85b1c: Pull complete
Digest: sha256:7831c923acc3b761f52ac9cfa8d449d3b8ad02d611cf4615795c08982bc41c9d
Status: Downloaded newer image for rayproject/ray:latest
docker.io/rayproject/ray:latest
```

说明Docker镜像下载完毕，之后使用如下命令启动head节点：

```
root@iZuf6g4orgeelv5wzce4luZ:~# docker run -it rayproject/ray
(base) ray@77d7e9da9366:~$
```

在docker的bash中使用：

```
ray start --head --port=6379 --include-dashboard=true --dashboard-host=0.0.0.0 --
dashboard-port=8265
```

启动ray head节点

```

-----
Ray runtime started.
-----

Next steps
  To connect to this Ray runtime from another node, run
    ray start --address='172.17.0.2:6379'

  Alternatively, use the following Python code:
    import ray
    ray.init(address='auto')

  To connect to this Ray runtime from outside of the cluster, for example to
  connect to a remote cluster from your laptop directly, use the following
  Python code:
    import ray
    ray.init(address='ray://<head_node_ip_address>:10001')

  If connection fails, check your firewall settings and network configuration.

  To terminate the Ray runtime, run
    ray stop
(base) ray@77d7e9da9366:~$ █

```

此时在ip为172.17.0.2的容器启动了头节点。

之后启动另一个Docker，运行如下命令：

```

(base) ray@efb60689fe02:~$ ray start --address='172.17.0.2:6379'
Local node IP: 172.17.0.3
2022-06-30 01:56:33,362 WARNING services.py:2013 -- WARNING: The object store
is using /tmp instead of /dev/shm because /dev/shm has only 67108864 bytes
available. This will harm performance! You may be able to free up space by
deleting files in /dev/shm. If you are inside a Docker container, you can in
crease /dev/shm size by passing '--shm-size=0.61gb' to 'docker run' (or add
it to the run_options list in a Ray cluster config). Make sure to set this t
o more than 30% of available RAM.
[2022-06-30 01:56:33,627 I 24 24] global_state_accessor.cc:357: This node ha
s an IP address of 172.17.0.3, while we can not found the matched Raylet add
ress. This maybe come from when you connect the Ray cluster with a different
IP address or connect a container.

-----
Ray runtime started.
-----

To terminate the Ray runtime, run
  ray stop
(base) ray@efb60689fe02:~$ █

```

可以看到，在另一个ip为172.17.0.3的容器中启动了worker节点。

此时在head节点使用：

```
ray status
```

可以查看Ray Cluster的信息：

```
(base) ray@77d7e9da9366:~$ ray status
===== Autoscaler status: 2022-06-30 01:57:02.994672 =====
Node status
-----
Healthy:
  1 node_885c758ac783c756c0bac3433d2cb623f8852051e0a2bee6269f5014
  1 node_5fddd662444557accc246e13169a0ecf3e288d13330f5d5c4711dba9
Pending:
  (no pending nodes)
Recent failures:
  (no failures)

Resources
-----
Usage:
  0.0/2.0 CPU
  0.00/2.207 GiB memory
  0.00/1.011 GiB object_store_memory

Demands:
  (no resource demands)
(base) ray@77d7e9da9366:~$
```

可以看到，此时Ray Cluster中存在两个节点，说明基于Docker的Ray分布式集群部署完毕。

下面同样对响应时间和吞吐率进行测试：

```
(base) ray@77d7e9da9366:~$ python3 test.py 100000
duration = 329.41372203826904
(base) ray@77d7e9da9366:~$ python3 test.py 10000
duration = 33.99405336380005
(base) ray@77d7e9da9366:~$ python3 test.py 1000
(scheduler +5s) Tip: use `ray status` to view detailed cluster status. To di
sable these messages, set RAY_SCHEDULER_EVENTS=0.
(scheduler +5s) Warning: The following resource request cannot be scheduled
right now: {'CPU': 1.0}. This is likely due to all cluster resources being c
laimed by actors. Consider creating fewer actors or adding more nodes to thi
s Ray cluster.
duration = 5.119900703430176
(base) ray@77d7e9da9366:~$
```

由于服务器计算资源受限，该处ray报了warning，其他情况，响应时间与任务数成正比，在代码结构不变时，吞吐量基本恒定不变。

将优化代码同样移植到docker中并运行：

```
(base) ray@77d7e9da9366:~$ python3 test_op.py  
duration = 5.477735996246338  
(base) ray@77d7e9da9366:~$
```

至此，基于Docker的Ray集群分布式容器化部署及测试完成。