Docker封装的分布式矩阵运算

这是一个使用Python3.6编写的简单的分布式矩阵计算的测试程序,支持分布式矩阵乘法与分布式矩阵求逆。整个项目使用Docker封装,容易批量配置。矩阵乘法使用分块乘法实现分布式,矩阵求逆使用分块消元算法实现分布式,仅仅实现功能,在网络传输、性能与并行度上依旧有所不足。

- multiprocessing 实现分布式的任务分配
- flask 实现Web控制界面
- h5py 实现结果HDF5格式存储

已发布到Docker Hub: https://hub.docker.com/r/wnjxyk/simple_distributed_matrix, 可以在这个网址拉取镜像并且测试。

Github链接: https://github.com/WNJXYK/Distributed-Matrix-Calculation

博客链接: https://blog.wnjxyk.cn/2018/12/29/docker封装的简易分布式矩阵计算/

使用与测试

Step One: 从镜像新建Docker容器

首先从镜像仓库拉取本镜像。

```
docker pull wnjxyk/simple_distributed_matrix
```

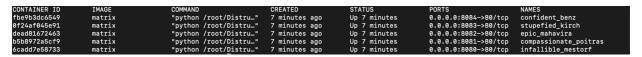
然后从镜像新建了5容器,分别映射端口号为8080、8081~8084。并且将8081~8084端口号容器的 CPU性能限制在1%。这里8080作为主控节点,8081~8084作为分布式计算节点。

限制CPU性能是为了最大化所演示的分布式计算的效果,因为程序比较简单,并未对于网络传输与性能有所优化,所以中心的控制节点会成为性能的瓶颈。我们将控制节点的性能不设限制,计算节点的性能限制在1%,可以展现不考虑中心节点与网络传输限制的理想情况下,分布式计算的效果。

```
docker run -d -it -p 8080:80 matrix python
/root/Distributed_Matrix_Method/Distributed.py
docker run -d -it -p 8081:80 --cpus=0.01 matrix python
/root/Distributed_Matrix_Method/Distributed.py
docker run -d -it -p 8082:80 --cpus=0.01 matrix python
/root/Distributed_Matrix_Method/Distributed.py
docker run -d -it -p 8083:80 --cpus=0.01 matrix python
/root/Distributed_Matrix_Method/Distributed.py
docker run -d -it -p 8084:80 --cpus=0.01 matrix python
/root/Distributed_Matrix_Method/Distributed.py
```

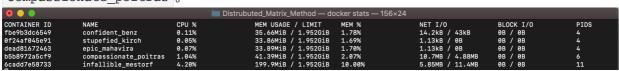
Step Two:测试性能

查明8080端口Docker容器在局域网中的IP,然后在Docker宿主浏览器中打开这五个Docker控制页面,进行分布式计算性能测试。



这里我们的控制节点名字叫做 infallible_mestorf, 四个计算节点的名字叫做 confident_benz 、 stupefied_kirch 、 epic_mahavira 与 compassionate_poitras 。

首先,我们测试一下只使用一个计算节点的情况下做矩阵乘法,这里我们启用计算节点 compassionate poitras 。



在控制页面中,我们可以看到这次矩阵乘法使用了61.56秒。

Host		Worker		
Address: 0.0.0.0 : E	51234	Address: 127.0.0.1 : 51234		
Authenticate: Matrix_Method		Authenticate: Matrix_Method		
Running	Terminate	Stopped	Start	
Task				
Matrix Size: 1024		Block Size: 256		
Inverse Task		Multiply Task		
Type: Mul		Status: Correct(Saved)		
Epoch: 1/1	Unit: 64/64	Time: 61.55	S	
Download HDF5 File				

然后,我们令四个计算节点一起工作计算同样规模的矩阵乘法。

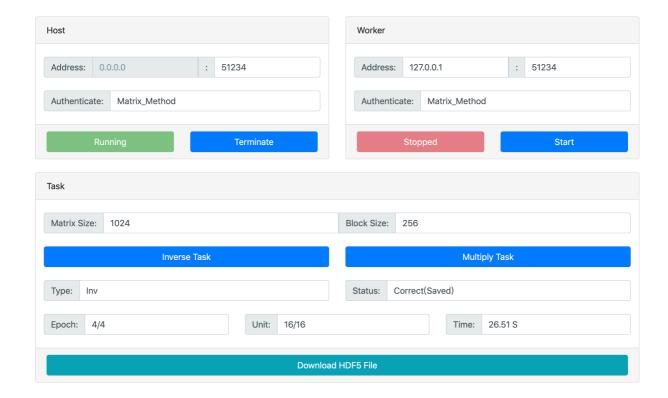


在控制页面中,我们可以看到这次矩阵乘法使用了19.86秒。提升非堂大

Host		Worker		
Address: 0.0.0.0 :	51234	Address: 127.0.0.1	: 51234	
Authenticate: Matrix_Method		Authenticate: Matrix_Method		
Running	Terminate	Stopped	Start	
Matrix Size: 1024		Block Size: 256		
Inverse Task		Multiply Task		
Type: Mul		Status: Correct(Saved)		
Type: Mul Epoch: 1/1	Unit: 64/64	Status: Correct(Saved) Time: 19.86	s	

再测试一下矩阵求逆运算,可以看到单个计算节点需要使用100秒,而四个计算节点只需要使用约26 秒。

Host	Worker			
Address: 0.0.0.0 : 51234	Address: 172.17.0.2 : 51234			
Authenticate: Matrix_Method	Authenticate: Matrix_Method			
Running Terminate	Stopped Start			
Task				
Matrix Size: 1024	Block Size: 256			
Inverse Task	Multiply Task			
Type: Inv	Status: Finished(Saved)			
Epoch: 4/4 Unit: 16/16	Time: 100.07 S			
Download HDF5 File				



Step Three:正确性测试

控制面板会自动测试结果的正确性,如果显示结果为 Correct 就可以判断结果是正确的。同时系统会讲结果存储为一个HDF5格式的文件,可以点击最下面的按钮进行下载,然后使用HDF Viewer进行浏览与检验。

分布式实现

乘法与求逆元都使用了分块的思路,将一个大小为siz的矩阵分成大小为blk的分块 $Block_{i,j}$,那么会得到一个 $blc=\frac{siz}{bl}$ 的方阵,其每个元素都是一个blk的方阵。

分块矩阵乘法

那么原来矩阵乘法 $Ans = A \cdot B$,可以转化成 $BlockAns_{i,j} = \sum_{k=0}^{\frac{sit}{blc}} BlockA_{i,k} \cdot BlockB_{k,j}$ 。可以看出分块之后的乘法之间是完全独立的,所以我们可以把这些耗时的矩阵乘法任务分发给计算节点运算,然后将结果返回加入结果的分块矩阵中就可以了。 假设计算节点有CNT个,我们可以把原本计算矩阵乘法的复杂度从 $O(n^3)$ 变换为 $O(blk^3 \times (\lceil \frac{blc^3}{CNT} \rceil)$ 。

Sequential Block-Based Gauss-Jordan Algorithm

矩阵求逆的方法使用的还是消元,不过使用了一种序列分块版本,对分布式运算更加友好。 这是一种将高斯消元的过程分块的算法,它的好处分块的过程中,块与块之间会存在很多不相关的乘法运算,利用这个性质,我们就可以将不相关的运算分散到运算节点上去。 假设计算节点有CNT个,我们可以原本 $O(n^3)$ 的复杂度降低为 $O(blk^3 \times blc \cdot (\lceil \frac{blc}{CNT} \rceil + \lceil \frac{blc \cdot (blc - 1)}{CNT} \rceil))$

Docker常用指令

1. 从Docker容器创建镜像

```
docker commit <conatiner ID> name:tag
```

2. 查看Docker之间的网络组织情况

```
docker network inspect bridge
```

3. 统计Docker容器使用情况

```
docker stats # CPU使用情况
docker ps # 运行中容器列表
docker ps -a # 所有容器列表
```

4. 关闭所有Docker执行进程

```
docker stop $(docker ps -a | awk '{ print $1}' | tail -n +2)
```

5. 关闭所有Docker容器

```
docker rm $(docker ps -a | awk '{ print $1}' | tail -n +2)
```