基于MARS&RAY&TENSORFLOW的分布式计算框架对比

# 引言

由于物联网、云计算以及智能设备的普及，大量数据在以前所未有的速度从不同来源产生（如健康、政府、社交网络、营销、金融）[1]。然而，摩尔定律正在逐步失效，硬件性能翻倍所需时间从上世纪的1.5年提高至如今的20年。数据量的增长远超摩尔定律，因此无法用单机单核满足现有的计算需求。现有的数据科学加速方式主要有两种：Scale UP和Scale Out，前者是增加核的数量，或提高硬件质量（如利用GPU、TPU）；后者是采用更大的计算集群，即我们所熟知的分布式计算方式[2][3]。最新的分布式计算平台已经结合两者优势，对分布式集群中的主机进行硬件升级，增加单个节点的数据处理能力。由于分布式计算平台的不同架构，相同的大数据集在不同平台呈现出不同的性能表现，因此需要对不同平台的对比研究。本组实施了对Mars、Ray、Tensorflow三个平台的对比研究，我本人主要负责Mars平台的实现和测量，工作量占整体工作的22%，具体内容见下述介绍。

# Mars平台介绍

Mars的架构如图1所示，主要分为用户层和执行层。在用户层，Mars封装了numpy、pandas等常用库以及Pytorch、Tensorflow等机器学习框架；在执行层，Mars设计了一个轻量级actor框架，用于实现任务的分布式执行。

图形用户界面

描述已自动生成

图1 Mars架构

Mars库的设计是用户友好的，实现了常用库的接口，但内部实现与原库大不相同，主要用于代码的分布式执行。如图2所示，仅需要对库导入部分进行修改就能实现代码的单机执行到分布式执行的修改。

图形用户界面

描述已自动生成

图2 从单机到Mars分布式执行的代码改动

Mars的具体执行流程如图3所示。由于Mars的延迟执行特点，在代码运行到.execute()部分后，会根据已有逻辑生成计算图，并发送至Web服务器和Scheduler，Scheduler节点会根据实时Worker连接情况将计算图拆分，并将拆分的子图逻辑发送至每个Worker，以进行分布式的任务执行。

图形用户界面

描述已自动生成

图3 Mars的执行流程

# 实验过程

## 3.1 Mars平台配置

在本实验中，我们在主机上建立三抬虚拟机，每台VM配两个网卡，一个设置为NAT模型，一个设置为内部网络。对第一台虚拟机配置为Supervisor类型，代码如下。其中-H表示Supervisor节点的IP地址，-p表示使用的端口，-w表示Web服务器所使用的端口。

|  |
| --- |
| sudo ifconfig enp0s8 192.168.0.1  mars-supervisor -H 192.168.0.1 -p 4316 -w 8909 |

另外两台虚拟机配置为Worker节点，代码相同，这里以一台为例。-H表示Worker节点的IP地址，-p表示Worker使用的端口，-s表示连接的Supervisor节点的IP地址:端口号。

|  |
| --- |
| sudo ifconfig enp0s8 192.168.0.2  mars-worker -H 192.168.0.2 -p 7869 -s 192.168.0.1:4316 |

接下来测试连接效果，运行如下代码，并观察<http://0.0.0.0:8909>显示的集群情况。

|  |
| --- |
| import mars  import mars.tensor as mt  import mars.dataframe as md  # create a default session that connects to the cluster  mars.new\_session('http://<web\_ip>:<web\_port>') # 表现为该网站上多了个session  a = mt.random.rand(2000, 2000, chunk\_size=200)  b = mt.inner(a, a)  b.execute() # submit tensor to cluster  df = md.DataFrame(a).sum()  df.execute() # submit DataFrame to cluster |

在约33s后，任务执行完成，Supervisor节点返回结果。但我在测试中发现Mars的一个bug。官方文档中说mars-worker命令具有参数设置命令--n-cpu，以设置该worker可以使用主机上的CPU的数量。在具有4核的主机上运行如下命令，设置CPU使用限制为1，此时DashBoard的显示如图4所示，猜测该worker仍然使用了所用的CPU。为验证猜想，我们在仅该worker运行时使用top指令查看虚拟机查看，发现四个CPU核的利用率均达到90%，说明猜想正确。

|  |
| --- |
| mars-worker -H 192.169.0.2 -p 7870 -s 192.168.0.1:4399 --n-cpu 1 |

图片包含 图形用户界面

描述已自动生成

图4 bug测试

## 3.2 实验方案设置

考虑到宿主机的性能，我们采用两个任务测试分布式平台的性能，分别是10000次(1000, 1000)矩阵的乘法，以及MNIST手写数字数据集的识别。虚拟机的配置皆为CPU4核、内存4G、磁盘20G。

# 实验评估

## 4.1 矩阵乘法

我们首先测试了worker数量对任务执行时间的影响。如图5所示，随着worker数量的增加，时间呈现出震荡趋势，但绝对值差别并不大。

公司名称

描述已自动生成

图5 矩阵乘法任务执行时间

为了研究震荡的规律性，我们取最大为8个worker的情况进行测试。如图6所示，发现奇数个worker会比前后两个时间更长一些，回到图5发现大部分奇数点都具有这个特点。而偶数点之间时间差距相对较小。

图表

描述已自动生成

图6 worker数量为1-8的矩阵乘法任务执行时间

通过对程序输出的观察，我们猜测时间的差异可能与两个虚拟机上的worker分配情况有关。因此我们对虚拟机配置进行修改，并逐步增大任务数，观察Mars自动的分配方案。结果如图7所示，在worker数较多时，Mars会自动先按虚拟机核数分配。当剩余worker数小于虚拟机核数时，会考虑worker共享虚拟机所有CPU核的特点，进行任务执行时间最短的分配。

表格

描述已自动生成

图7 异构情况下的worker分配情况

图8为Mars分配方案的实例，此时worker数量设为16。在进行了三=两轮的4-2分配后，还剩余4个worker，此时会进行3-1分配，所需时间为Max(3/4, 1/2)，是此时的最优时间分配。黄色框出的最后一轮执行时间也能说明worker占用所有CPU，同一虚拟机上所有worker并发使用虚拟机的所有CPU。

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

图8 16个worker的分配方案

我们还测试了Mars的CPU利用率，如图9所示，Mars对4个CPU的利用率非常高，一直处于接近100%的情况。

图片包含 图表

描述已自动生成

图9 CPU利用率

## 4.2 MNIST识别

图10所示为MNIST任务的执行时间。由于Mars的CPU利用率极高，其任务执行时间是三个平台中最短的。

图表, 条形图

描述已自动生成

图10 MNIST任务的执行时间

图11所示为MNIST任务的精度。由于参数设置相同，三个平台最后都收敛到相似的精度，但收敛时间不同，可以看到Mars的收敛速度基本上是最快的。

折线图

中度可信度描述已自动生成

图11 MNIST任务的精度

图12所示为MNIST任务的网络流量情况。Ray平台的网络流量是Mars平台的近10倍，因为Ray的worker会在不同虚拟机间“移动”，而Mars的worker在初始化后便不再移动，因此网络流量较小。

图表, 条形图

描述已自动生成

图12 MNIST任务的网络流量

# 参考文献

1. Botta A, De Donato W, Persico V, et al. Integration of cloud computing and internet of things: a survey[J]. Future generation computer systems, 2016, 56: 684-700.
2. Messer A, Greenberg I, Bernadat P, et al. Towards a distributed platform for resource-constrained devices[C]//Proceedings 22nd International Conference on Distributed Computing Systems. IEEE, 2002: 43-51.
3. Verbraeken J, Wolting M, Katzy J, et al. A survey on distributed machine learning[J]. ACM Computing Surveys (CSUR), 2020, 53(2): 1-33.