科研信息化技术与应用
2019, 10(2): 20–30

·技术 / TECHNOLOGY·

基于 Mesos 的分布式参数优化调度策略及系统设计

李　铄[1,2]，陆忠华[1]，孙永泽[1,2]

1. 中国科学院计算机网络信息中心，北京 100190
2. 中国科学院大学，北京 101408

摘　要： 本文针对分布式参数优化系统的调度策略进行研究，并实现了一个基于 Mesos 的分布式参数优化系统。利用 Mesos 的资源接口，把多种常见的参数优化算法和任务调度封装为一个可以在 Mesos 上运行的框架软件。并针对 Mesos 的两级调度机制，提出了一种对混合部署集群上多作业竞争环境下的分布式参数优化系统的调度优化策略。本文设计了多组实验，对框架软件的资源调度策略与 FIFO 调度策略进行对比测试，可以满足多租户在混合部署场景下的使用。降低了在集群环境下进行深度学习等常见场景下的分布式参数优化的难度，在多任务竞争时提高资源使用效率。

关键词： Apache Mesos；资源调度；参数优化；分布式系统；分布式计算

doi: 10·11871/j·issn·1674-9480·2019·02·003

Distributed Parameter Optimization Scheduling Strategy and System Design Based on Mesos

Li Shuo[1,2], Lu Zhonghua[1], Sun Yongze[1,2]

1. Computer Network Information Center of Chinese Academy of Sciences, Beijing 100190, China
2. Chinese Academy of Sciences, Beijing 101408, China

Abstract: This paper implements a distributed parameter optimization system based on Mesos and studies the scheduling strategy of this system· Using the resource interface of Mesos, the system packages a variety of common parameter optimization algorithms and task scheduling strategy into a framework software that can run on Mesos· Aiming at the two-level scheduling mechanism of Mesos, a dynamic scheduling strategy for distributed parameter optimization system in multi-job environment on hybrid deployment cluster is

proposed· This paper designs several experiments, and compares the resource scheduling strategy of the architecture software with the FIFO scheduling strategy in the hybrid deployment scenario· This work reduces the difficulty of optimizing distributed parameters in common scenarios such as deep learning in a cluster environment, and improves resource utilization efficiency in multi-task environment·

Keywords: Apache mesos; scheduling; parameter optimization; distributed system; distributed computing

引言

随着计算机技术的不断进步和网络技术普及，人工智能技术近年来不断地提升，机器学习和深度学习已经广泛地用于计算机视觉，自然语言处理等各个领域。

一方面，机器学习和深度学习这一类需要大量资源和算力，TensorFlow、Caffe、Pytorch 等深度学习框架也都相继推出了分布式版本，可见，这类任务对大规模的计算、存储、传输等方面的要求也越来越高，而单台计算机硬件和软件的发展却远远落后于这类应用对这些方面的需求。另一方面，机器学习和深度学习的算法效果受到多方面的影响，如训练数据，模型选择，超参数优化等。在人工智能中普遍使用的学习式模型里通常情况下需要人为指定的一些参数，这些参数就是超参数[2]。针对这些超参数的优化，继而提出了超参数优化的概念，是指在确定模型和参数组合确定的情况下，设定每个参数的优化范围，对这些参数进行优化以达到一个满意的训练效果。但是，参数优化的过程中缺乏资源调度机制，目前的实现中不支持多租户，平台模式提交参数优化作业，这对于现在大量使用大规模集群资源的状况相违背，导致用户可能只能使用十分有限的资源来进行参数优化，无法很好利用集群资源。

目前很多机器学习算法库都提供了他们的参数调优方案，在这一领域称为 Auto-ML，AutoML 的最终目标是为具有有限数据科学或机器学习背景的领域专家提供易于访问的深度学习工具。Google 在其云产品上进行了自动参数优化的集成，也就是 Vizier[2] 的外部接口。Auto-sklearn [3] 是一个自动化机器学习的工具包，其基于 sklearn 编写，Auto-sklearn可以进行机器学习算法的自动选择与超参数的自动优化, 它使用的技术包括贝叶斯优等。Auto-Keras[4] 是一个用于自动机器学习的开源软件库。它由 Texas A & M 大学的 DATA 实验室和社区贡献者开发。Auto-Keras 提供自动搜索深度学习模型的架构和超参数的功能。

分布式计算领域有许多用于大数据和高性能计算领域著名实现。MapReduce[5] 是 Google 公司于 2004 年提出的能并发处理海量数据的并行编程模型，其特点是简单易学、适用广泛，能够降低并行编程难度，让程序员从繁杂的并行编程工作中解脱出来，轻松地编写简单、高效的并行程序。MPI [6] 能广泛应用于多类并行机群和网络环境，是建立在多种可靠的消息传递库的基础上的一种接口模式。Parameter Server[7] 基于参数服务器框架的分布式机器学习系统上设计并行优化算法，采用灵活的一致性模型，提高算法执行效率。参数优化的场景和这些常用分布式计算区别在于需要持续的任务提交，这一特性对集群资源的管理提出了更高的要求。

调度系统同样有很重要的地位[1]，Kubernetes[11] 是 Google 开源的容器集群管理系统（谷歌内部：Borg[12]）。此外还有其他各种调度系统例如 YARN[13]，Slurm[14]，LSF[15] 等。Mesos[9] 调度系统，Mesos 自身只是一个资源抽象的平台，要使用它往往需要结合运行其上的分布式应用 (在 Mesos 中被称作框架 Framework[16])，这些应用包括 Hadoop、Spark、Kafka、Elastic Search。还可配合框架 Marathon 来管理大规模的 Docker[8] 等容器化应用，另外这些框架还需要实现自己的二级调度，也就是在 Mesos 提供资源的同时对资源进行再分配，提供对自己任务本身的更为动态和细粒度的支持，给了分布式应用开发者更大

的空间[10]。

在通常的单机实现中，往往由用户使用的软件包串行执行训练程序，无法充分利用资源，在机器学习和深度学习的参数调优的过程中，这一缺点暴露得更加明显，因为这些场景下的训练任务往往要执行更多的时间，并且需要大量的资源。

参数优化和分布式存在着很大的契合点，尤其是在机器学习和深度学习这一类需要大量计算资源的领域，如何通过分布式的采样算法去并行执行参数优化任务，以及如何在提高并发度的同时更好地利用集群资源是将两者结合的关键问题。通过分布式的参数优化系统可以在一定程度上提升参数优化的效率，同时提高集群资源的利用率。

本文同时从分布式计算和基础设施两个层面来考虑这一问题，与分布式的参数优化算法相结合来形成分布式计算这一环节，改进 Mesos Framework 的资源调度策略来更好地利用集群资源。设计和实现一个基于 Mesos 的分布式参数优化系统，计算层由各类优化算法组成，进行分布式参数优化的采样和生成计算任务，调度层是一个 Mesos Framework 的具体实现，主要负责资源分配和执行计算任务。更进一步地，我们可以针对参数优化场景提出一种调度影响计算的策略来控制参数优化这一过程，以平衡多租户分布式参数优化场景下的资源竞争问题。

1　基于 Mesos 的参数优化系统设计

Mesos 是常见的集群资源管理系统之一。Mesos 相较于其他调度系统更加轻量，用于支持研究者进行自定义应用的开发的接口更加友好[9]，其特有的两级调度机制为研究者制造了更大的探索空间。Mesos 作为集群的基础设施，围绕这一基础设施，不同的应用使用不同的 Mesos Framework 来进行对其建设，为的是可以将应用衔接进入 Mesos 集群系统，获取集群的资源。为了使参数优化系统能够使用集群资源，需要设计和实现一个针对这一应用的 Mesos Framework 和计算框架，形成一个完整可用，可扩展性高的原型系统。

1.1　Mesos 的基本架构

Mesos 主要由 Mesos Master，Mesos Slave，Mesos Framework 组成，其中 Mesos Framework 又包括 Scheduler 和 Executor 两个重要模块。

上图显示了 Mesos 的主要组成部分。Mesos 由一个主守护进程来管理从守护进程在每个集群节点上的运行，Mesos Frameworks在这些 Slaves 上运行 Tasks。

Master 使用 Resource Offers 实现跨越不同应用的细粒度资源共享，如 CPU、内存、磁盘、网络等。Master 根据指定的策略来决定分配多少资源给 Framework，如公平共享策略，或优先级策略。为了

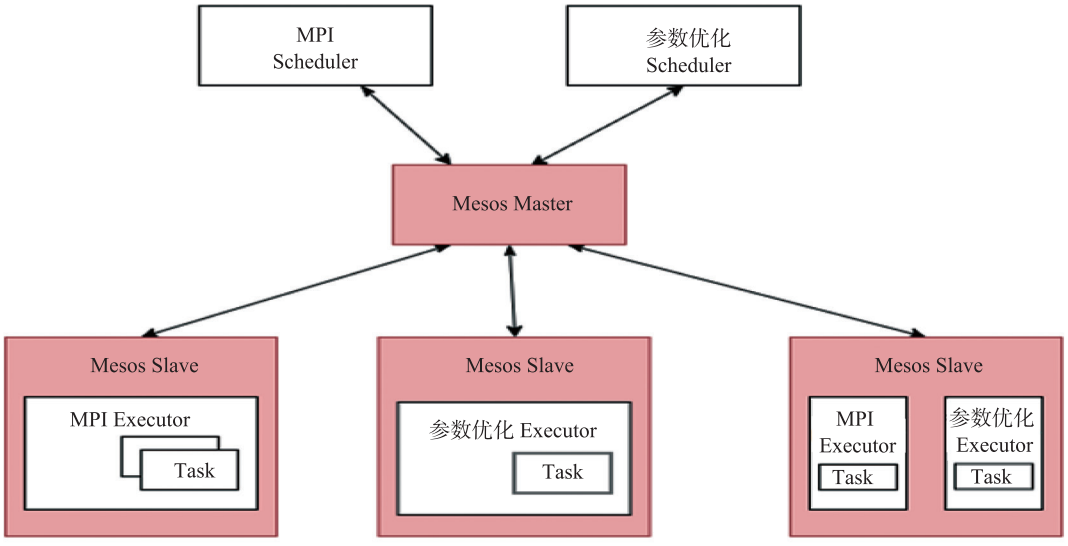

图 1　Mesos 架构图

Fig.1　Mesos architecture diagram

支持更多样性的策略，Master 采用模块化结构，这样就可以方便的通过插件形式来添加新的分配模块。

在 Mesos 上运行的 Framework 由两部分组成：一个是 Scheduler，通过注册到 Master 来获取集群资源。另一个是在 Slave 节点上运行的 Executor 进程，它可以执行 Framework 的 Task。Master 决定为每个 Framework 提供多少资源，Framework 的 Scheduler 来选择 Master 提供的资源。当 Framework 同意了提供的资源，通过 Master 将 Task 发送到提供资源的 Slaves 上运行。

1.2　分布式参数优化系统框架架构

根据不同的应用场景来实现不同 Mesos Framework，是 Mesos 系统和其他调度系统区别所在，同时这一架构体现了 Mesos 系统所特有的两级调度方式，文章会在 2·1 节对 Mesos 的资源调度策略进行更进一步的介绍。常见的 Mesos Framework 实现是针对不同的业务模型和作业方式对 Mesos Master 给予的资源进行更加优化的管理调度，例如在线长作业，离线批处理作业等模式。

本文所提出的分布式参数优化系统，其核心调度功能也是通过实现一个 Mesos Framework 来满足多用户使用 Mesos 集群资源，提高资源的整体利用率，减少用户在参数优化过程中需要做的资源调度工作。此外，在参数优化系统中，如何进行参数的计算和迭代也是至关重要的一环。采用了常见的 Master-Slave 架构，App Core 作为 Master，App Runner 作为 Slave，计算结果汇报给 App Core。每个单独的计算的任务可以作为一个 Mesos Task 托管在 Mesos 中运行，通过 Docker 进行资源隔离。

本系统解决了多租户在集群系统上进行参数优化的问题，提供分布式参数优化的支持，通过 Mesos Framework 很好地集成了负责参数优化的应用程序，对于 Mesos 集群系统侵入性很小，如图 2 所示。

整个系统包括四种组件：

Client：客户端，负责和用户进行交互，用户根据其需求提供参数并启动框架

Mesos Framework：负责启动 App Core，App Runner，进行资源的二级的调度，与 Mesos Master 通信，负责监控作业运行状态等工作。

App Core：是一个交由 Mesos Task 托管的核心应用程序，主要负责运行参数优化算法，保存参数优

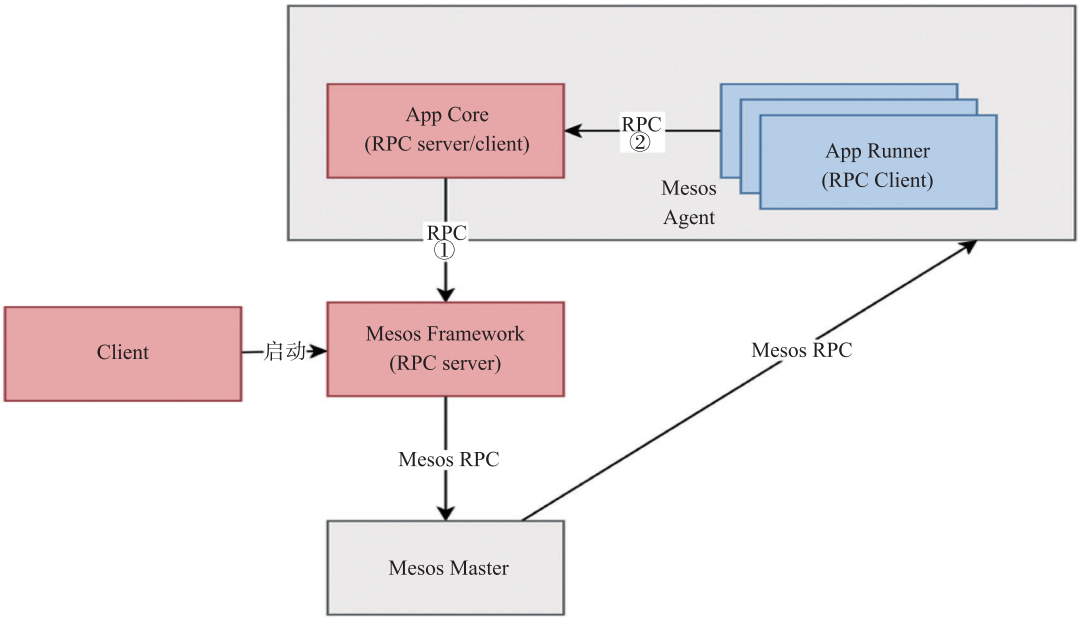

图 2　分布式参数优化系统架构

Fig.2　Distributed parameter optimization system architecture

化结果等。

App Runner：交由 Mesos Task 托管的用户提供应用程序，按照用户提交的代码进行结果运算，并将运算结果反馈至 App Core。当用户提高并行参数时，App Core 会向 Mesos Framework 根据并行度提交运行 App Runner 的参数。

1.3 系统实现关键技术

1.3.1 Client设计

客户端供用户提交自己的 App Runner 程序，其关键部分主要有以下两点：

⑴ 任务参数输入

用户通过提交脚本的方式进行作业提交，并进行对应的参数输入，这类参数主要数据输入路径，App Runner 程序路径和启动命令，App Core 和 App Runner 各自的运行 CPU 核数，内存大小等，这一类参数与其他调度系统有很大的相似度。

如下所示：

本文所描述的系统增加了优化参数的特殊配置，以及优化算法的选择等与参数优化相关的配置，如下是个简单优化参数配置文件：

以机器学习的参数优化为例，通常情况下，待优化的参数选取如 learning rate, momentum, batch size 等，其值 value 可以是上下界区间，数值选择。

demo-parameters·json:

```
{
 "parameters": [
  {
   "name": : " momentum",
    "type": : "bound"
   "value": [0·1, 0·3]
  },
  {
   "name": : "learningRate",
   "type": : "number"
   "value": [0·1, 0·2, 0·3]
  }
 ],
}
```

```
#!/bin/sh
$DISAUTO_HOME/bin/da-submit \
    --app-name "testAutoML" \
    --input /tmp/data/testAutoML#data \
    --output /tmp/testAutoML_model#model \
    --files demo.py \
    --app-runner-launch-cmd "python demo.py" \
    --app-runner-memory 5G \
    --app-runner-cpus 3 \
    --app-core-memory 1G \
    --app-core-cpus 2 \
    --trail-steps 100 \
    --tuning-type "random-search" \
    --tuning-parameters demo-parameters.json \
```

图 3 用户提交命令与参数

Fig.3 User submits commands and parameters

⑵ 任务状态监控

任务状态获取，用户可以通过命令行或者 Mesos界面查看整个作业的运行状态，即作业中每个任务的运行状态。主要是通过 org·apache·mesos·statusUpdate 接口函数来实现的。对于每个 Task，主要包括PENDING，RUNNING，FINISHED，FAILED 等常见任务状态。

1.3.2 Mesos Framework 设计

Mesos Framework 是基于 Mesos 调度系统的分布式应用，相对于 Mesos Master 和 Mesos Slave 是独立的，Mesos Framework 和 Mesos Master 服务进行交互，并不直接和 Mesos Slave进行交互。本文设计的 Framework 实际上是通过 Client 直接在本地启动，并根据 Client 参数将用户需求抽象成 Job，Job 同时负责填充 Mesos 的 Task 参数。

Mesos Framework 和 Mesos Master 的交互，过 RPC[21] 调用的方式进行通信，包括 Framework 的注册，接收资源，拒绝资源，状态更新等。接收资源，即 resourceOffers，是一关键调用，涉及到资源的二次调度，填充 Task 参数，拒绝资源等。

Mesos Framework 和 App Core 的交互也是通过 RPC 进行的，包含一个 RPC Server。

1.3.3 App Core 设计

对于参数的优化，可以将这种优化看作是反映泛

化性能的未知黑盒函数的优化，并调用针对这些问题开发的算法进行参数选择。App Core 的任务就是集成了这些算法，在用户提交了任务之后由 App Core 根据用户选择的优化算法和参数进行计算。一个作业中有且只有一个 App Core，其中包含一个 RPC Server 和 Client。在选择了参数之后 App Core 的 RPC Client 向 Mesos Framework 通过 RPC Client 进行 RPC 通信过程①，提交下一轮实验的参数，内容是由 App Core 计算出来的参数，提供给调度的优先级系数等。App Core 其本身在系统中也是一个作业。如果参数优化算法串行的提交作业，在一定时间内能完成的作业如图 4 所示。

如果指定并行度为 n=3，那么如图 5 所示在相同的时间内可以完成更多的训练任务。

App Core 中支持的参数优化算法如表格 1 列出的，值得一提的是图 6 所示的是一种异步的并行，目的是为了提高并行效率，不必去等较慢的进程 [20]，但是在贝叶斯优化 [18–19] 中，需要反馈训练结果，以进行下一步的采样，所以其同步并行会产生一些效率问题。

1.3.4　App Runner 设计

这里的 App Runner 是一个统称，表示用户通过 Client 上传的训练程序，根据 App Core 选取的参数，App Runner 在 Mesos Framework 指定的资源内形成 Mesos Task 运行，训练结果会通过 RPC Client 向 App Core 以 RPC 通信过程②的方式进行汇报，其内容是训练结果和训练时间等。用户只需要根据系统提供的 Python 工具包，简单修改自己的程序，就能作为一个 App Runner 运行在本系统中，达到和 App Core 交互的目的。

2　参数优化系统的调度策略

基于本文第一章的论述，研究并设计实现了一个基于 Mesos 的分布式参数优化系统，因此，在本章中我们将对这一系统的调度策略进行改进。主要是指在原先 Mesos 的两级调度基础上，针对文中 Mesos

表 1　App Core 中支持的参数优化算法

Table 1　Parameter optimization algorithms supported in App Core

参数优化算法	并行方式	训练结果是否需要反馈
网格搜索	异步并行	否
随机搜索	异步并行	否
贝叶斯优化	同步/异步并行	是

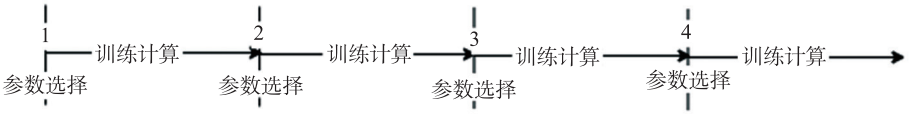

图 4　串行参数优化

Fig.4　Serial parameter optimization

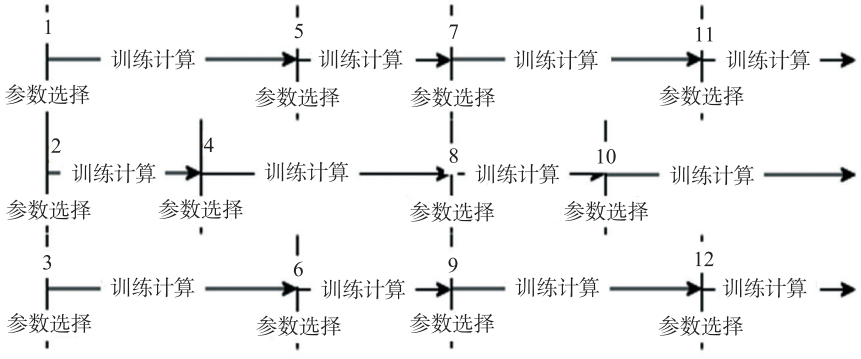

图 5　并行参数优化

Fig.5　Parallel parameter optimization

Framework 所承载的参数优化应用的运行模式进行优化，以达到在减少多租户分布式参数优化场景下的资源竞争的目的。

2.1 Mesos 资源调度策略

基于图 6 的事件流程:

a· Slave1 向 Master 报告，有 4 个 CPU 和 4 GB 内存可用

b· Master 发送一个 Resource Offer 给 Framework1 来描述 Slave1 有多少可用资源

c· FrameWork1 中的 FW Scheduler 会答复 Master，我有两个 Task 需要运行在 Slave1，一个 Task 需要<2 个 cpu，1 gb内存>，另外一个 Task 需要 <1 个 cpu，2 gb 内存>

d· 最后，Master 发送这些 Tasks 给 Slave1。然后，Slave1 还有 1 个 CPU 和 1 GB 内存没有使用，所以分配模块可以把这些资源提供给 Framework2

Mesos Master 即为第一级资源调度，在大部分情况下采用 DRF 调度算法 [17]，主资源公平调度 (Dominant Resource Fairness) 算法，非常适合应用于多维资源管理和调度的复杂环境。其中，主资源指的是各所需资源在相应总资源中所占比例最大的资源，在 Mesos 中，体现在一轮分配中对于每个 Framework 的分配的资源不同。

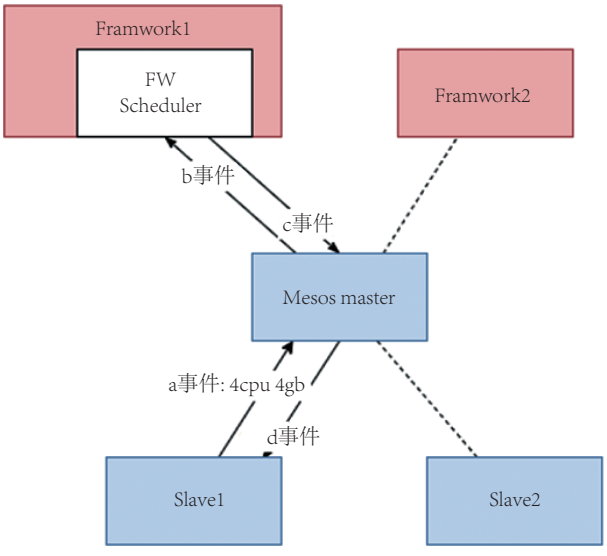

图 6 Mesos资源调度策略

Fig.6 Mesos resource scheduling strategy

Mesos Framework 为第二级资源调度，在大部分实现中采用的 FIFO 算法实现，即先来先服务算法。因为经过了一级资源调度，Mesos Master 通过 Resource Offer 接口给予 Framework 的资源已经是比较细粒度的资源，所以对于一些普通的在线作业或者批处理作业仅需要进行一些资源排查。在本文所指出的分布式参数优化的场景下，这一部分调度可以做到动态调度以在集群负担较重时节约集群资源。

2.2 动态资源调度策略

本文提出一种针对分布式参数优化的原型系统中 Mesos Framework 里的动态资源调度策略，利用 Mesos 系统的两级调度特色去探索一种新的调度策略，以优化资源调度效率。在多租户多任务的分布式参数优化场景下，我们目前的优化目标是：系统中存在多个任务 J_i 时，在尽量短的时间 T 内，使多个任务n达到接近其本身应该达到的优化效果 F_i。算法1描述如下：

算法 1

if (n > k * m) then // n为当前运行的任务J的数目, k为倍数系数，m为取样个数

F = <f1 , f2 … fn> //每次计算返回的结果的序列，fi按照效果由好到坏排列

g = min(|f1 - f2|, …|fm-1 -fm|) //计算前m对的效果差值

if g < G || g ≈ g’ then //G为当前阈值，g’为前多次计算的gap均值

pi = P + 1 //降低下一个J的优先级，p表示对应Job的优先级

else

pi = P

if pi > P then

P = pi //更新全局优先级

j = j + 1

G = Gj //Gj表示下一个阈值

Mesos 特有的两级调度系统，Mesos Framework 决定 Mesos 提供的资源是否合适，从而接受或者拒绝这个资源，当 Framework 拒绝资源，Mesos 将跳过该 Framework，将资源提供给其他 Framework。因此，根据 App Core 返回的优先级，会根据配置跳过 d 个 Mesos 分配的资源，这些资源会分配给其他框架执行高优先级任务。如果 Framework 发现集群资源空闲即多次拒绝后分配的资源没有减少，就会调高作业的优先级，从利用优化结果和集群资源的变化达到了一种动态调度。

3　实验结果与分析

3.1　系统部署环境

3.1.1　硬件部署环境

实验环境由两台曙光高性能测试服务器组成，其中 Master 服务器配置为 Intel Xeon CPU E5-2620 v3 的 24 处理器，251G 内存，另一台Cluster服务器配置为 Intel Xeon CPU E5-2640 v4 的 40 核处理器，251G 内存。网络连接为万兆以太网。

3.1.2　软件部署环境

两台高性能测试服务器的软件部署环境相同：

Ubuntu 16·04·1 LTS, OpenJDK 1·8·0_181，Python 2·7，protobuf 3·6·1，Mesos 1·6·1

实验在 Cluster 服务器，启动 3 个 Mesos Agent 通过其资源隔离来模拟 3 台资源皆为 12 核 CPU，20G 内存的集群系统；Master 服务器部署 Mesos Master，及本系统的 Client 启动，和为 Mesos Framework 提供运行环境。

1.1　实验程序

在本文提到的系统中，从用户的角度来讲，需要准备提供给 App Runner 运行的程序，本文将平时使用的单机机器学习参数优化的程序作为 App Runner 来进行实验，来探讨其在本系统中不同场景下的运行效率问题。

首先对实验中的一些项目进行定义，定义用户通过 Client 提交的参数优化任务为一个作业，在一个作业中拥有自己的 Mesos Framework，App Core 和 App Runner，即本文所描述的参数优化系统，整个 Mesos集群会有多个作业同时运行。定义在一个作业通过 App Core 运行一个 App Runner 为一个任务，并行度即指作业中可以同时运行的任务个数。定义通过 Client 端输入的为作业参数。定义通过 App Core 产生的为训练参数，即参数优化的目标。

⑴ 测试数据：采用 MNIST 数据集，MNIST 数据集是采集阿拉伯数字 0–9 的手写数字数据，每幅图片均为 0 到 9 中 10 数字的任意一个，黑白像素。MNIST 数据集可在 http: //yann·lecun·com/exdb/mnist/ 获取。本文中使用 Python 的 MNIST 包接口对数据直接进行处理。

⑵ 测试程序：单机基于 CPU 的 TensorFlow 程序，作为 App Runner 提交到系统。其算法用于手写数字识别，网络架构为卷积神经网络。对于其中 5 个训练参数进行优化，包括 learning rate, momentum, batch size 等参数。测试程序在训练中达到正确率 95% 以上时对于用户来说达到了要求的正确率，此时如果继续优化也可以获得更好的效果，但是因为期望大部分用户可以在资源竞争的情况下更高效地获得更好的效果，因此实验中以这一正确率判断作业结束的标志。即我们在算法 1 中提到的 F_i。

⑶ 参数选择：随机搜索时一种常见的基础参数优化算法 [22]，因此我们设定参数优化算法为随机搜索，最大实验次数为 200，表示对于一个作业来说，App Core 会做出最多 100 次的训练参数选择。设定App Core 需要 1 核 CPU，1G 内存，App Runner 即 ⑵ 中的测试程序需要 4 核 CPU，5G 内存。

3.2　实验结果

实验主要分为串行场景和并行场景，主要对比 Mesos Framework 中使用 FIFO 算法和本文的提出的动态调度算法在相同作业数 m 和并行数 n 的情况下的表现，即在作业的正确率达到 95% 时作业所消耗的时间，其中作业数表示模拟 m 个用户提交任务。每种情况设置了 10 组实验，最后结果去除 10 组实验中的最高和最低值，取剩下 8 中的平均数作为实验结果。

3.2.1 串行实验

串行场景是指每个作业内参数优化过程为串行执行，主要是为了保证不出现资源竞争得状况，即在固定资源内，增加作业数，观察在没有资源竞争发生的情况下的作业执行效率。实验结果如表 2 所示。

由图 7 可以看出，动态资源调度方法和 FIFO 调度方法在无资源竞争的情况下表现差别不大，动态资源调度会稍微耗时一些，原因是 Mesos 的两级调度中，第一级资源调度的资源分配存在一定间隔周期，在动态资源调度中会存在多个这样的周期，FIFO 则不存在。

3.2.2 并行实验

并行场景增加了并行度，首先观察在只有一个作业情况下提高并行度产生的加速比。结果如表 3 所示。

从图 8 可以看出，增加并行数可以提高作业运行效率，参数优化作业只有在一次训练之后才会进行通信，不会产生频繁的通信开销。对比动态调度算法和 FIFO 在此场景下差别不大。

最后，将作业并行度设定为 4，也就是一个作业会消耗 17 个核 CPU，34G 内存，增加作业数，观察在有资源竞争情况下作业运行时间，记录如下表 4。

随着作业数的增加，三个节点一共 36 核 CPU，60G 内存的总体资源不再充裕，产生了资源竞争，如图 9 所示，在作业数为 3 和 4 时资源竞争已经非常激烈，增加了作业的运行时间，此时动态调度算法的效率略高于 FIFO，整体的作业效率得到了提升。

表 2 无资源竞争作业运行时间 单位：秒

Table 2 No resource competition job running time(second)

作业数	1	2	3	4
FIFO	213·776	215·258	216·328	217·251
动态资源调度	220·523	223·26	224·14	225·516

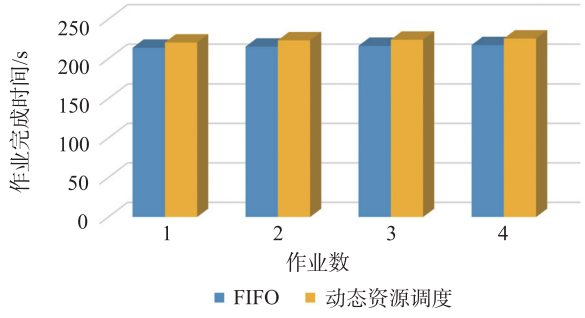

图 7 无资源竞争作业运行时间 单位：秒

Fig.7 No resource competition job running time(second)

表 3 单个作业在不同并行度下的运行时间 单位：秒

Table 3 xxRun time of a single job at different parallelism (second)

并行度	1	2	4	8
FIFO	213·776	112·445	75·42	42·457
动态资源调度	220·523	120·189	80·786	50·98

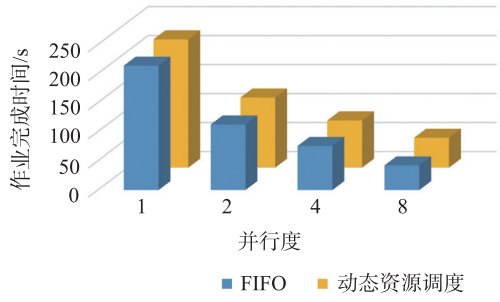

图 8 单个作业在不同并行度下的运行时间 单位：秒

Fig.8 Run time of a single job at different parallelism(second)

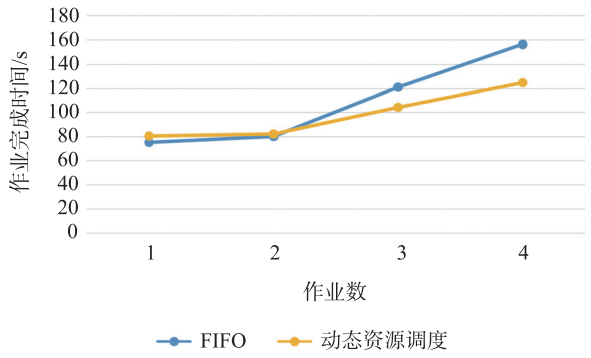

图 9 资源竞争环境下并行度为4的作业运行时间 单位：秒

Fig.9 xxJob running time with parallelism of 4 in a resource competitive environment(second)

表 4 资源竞争环境下并行度为4的作业运行时间 单位：秒

Table 4 Job running time with parallelism of 4 in a resource competitive environment(second)

作业数	1	2	3	4
FIFO	75·42	80·15	121·239	156·67
动态资源调度	80·786	82·414	104·45	124·725

4　总结与下一步工作

本文介绍了以 Mesos 为基础系统，在其上开发的一套分布式参数优化系统，主要指一个 Mesos 的 Framework，这个 Framework 可以运行在任何以 Mesos 为基础的分布式系统上。用户仅需要提供需要进行参数优化的训练程序，将其和 Framework 提交给 Mesos，由 Framework 来控制整个系统的运行流程，通过 Mesos Master 提供的资源 Offer 来进行调度，为后续的工作提供系统基础。在整个系统中，参数优化算法框架也是非常重要的一环，目前已经实现了常见的例如随机搜索，网格搜索等基础算法，根据用户的需要运行不同的优化算法。为了在资源竞争环境下提高系统的整体效率，本文从系统优化的角度来进一步研究适用于分布式参数优化的调度策略。本文充分利用了 Mesos 的两级调度机制来进行探索，取得了初步的效果。值得一提的是，因为本文所提出的系统可以满足多租户在混部场景下的使用，所以该策略的适用范围更加广泛，对已经存在的 Mesos 集群系统侵入性很低。

目前，许多需要参数优化的程序更多的采用了 GPU 等硬件来加速计算，未来针对本文的原型系统还可以加入 GPU 的调度，在调度策略上可以采用强化学习的方法来进行改进。

参考文献

[1] 兰舟· 分布式系统中的调度算法研究[D]· 电子科技大学, 2009·

[2] Golovin D, Solnik B, Moitra S, et al· Google Vizier: A Service for Black-Box Optimization[C]// ACM SIGKDD International Conference on Knowledge Discovery and Data Mining· ACM, 2017: 1487–1495·

[3] auto–sklearn[Online]Available: https: //automl·github·io/autosklearn/stable/#

[4] Auto–Keras [Online]Available: https: //autokeras·com/

[5] Dean J, Ghemawat S· MapReduce: simplified data processing on large clusters[M]· ACM, 2008·

[6] Lusk E, Gropp W· The MPI Message–Passing Interface Standard: Overview and Status[J]· Advances in Parallel Computing, 1995, 10(06): 265–269·

[7] Li M, Andersen D G, Park J W, et al· Scaling distributed machine learning with the parameter server[C]// Usenix Conference on Operating Systems Design and Implementation· USENIX Association, 2014: 583–598·

[8] Merkel D· Docker: lightweight Linux containers for consistent development and deployment[J]· 2014, 2014(239)·

[9] Hindman B, Konwinski A, Zaharia M, et al· Mesos: A Platform for Fine–Grained Resource Sharing in the Data Center[C]// Proceedings of the 8th USENIX conference on Networked systems design and implementation· USENIX Association, 2010: 429–483·

[10] Brewer E A· Kubernetes and the path to cloud native[C]// ACM Symposium on Cloud Computing· ACM, 2015: 167–167·

[11] Vavilapalli V K, Murthy A C, Douglas C, et al· Apache Hadoop YARN: yet another resource negotiator[C]// Symposium on Cloud Computing· ACM, 2013: 1–16·

[12] Verma A, Pedrosa L, Korupolu M, et al· Large–scale cluster management at Google with Borg[C]// Tenth European Conference on Computer Systems· ACM, 2015: 1–17·

[13] Burns B, Grant B, Oppenheimer D, et al· Borg, Omega, and Kubernetes[J]· Communications of the Acm, 2016, 59(5): 50–57·

[14] Jette M A, Yoo A B, Grondona M· SLURM: Simple linux utility for resource management[C]// International Workshop on Job Scheduling Strategies for Parallel· 2003·

[15] Zhou S· LSF : Load sharing in large–scale heterogeneous distributed systems[J]· 1992·

[16] Vellaipandiyan S· Big Data Framework – Cluster Resource Management with Apache Mesos[J]· 2014, 3(4): 228–294·

[17] 柯尊旺, 于炯, 廖彬· 适应异构集群的Mesos多资源调度

DRF增强算法[J]· 计算机应用, 2016, 36(5): 1216–1221·

[18] Pelikan M· Bayesian Optimization Algorithm[J]· Technical Report Illigal, 1999: 525–532·

[19] Snoek J, Larochelle H, Adams R P· Practical Bayesian optimization of machine learning algorithms[C]// International Conference on Neural Information Processing Systems· Curran Associates Inc· 2012: 2951–2959·

[20] Kandasamy K, Krishnamurthy A, Schneider J, et al· Asynchronous Parallel Bayesian Optimisation via Thompson Sampling[J]· 2017·

[21] Srinivasan R· RPC: Remote Procedure Call Protocol specification: Version 2[J]· Rfc, 1988, 11(3): 5531·

[22] Bergstra J , Bengio Y · Random Search for Hyper–Parameter Optimization[J]· Journal of Machine Learning Research, 2012, 13(1): 281–305·

收稿日期: 2019 年 1 月 30 日

李　铄：中国科学院计算机网络信息中心，中国科学院大学，硕士研究生，主要研究方向为高性能计算应用。

E–mail: *yinhao@cnic.cn*

陆忠华：中国科学院计算机网络信息中心，研究员，博士，主要研究方向为高性能计算应用技术。

E–mail: *zhlu@cnic.cn*