TensorFlow 内核剖析

TensorFlow Internals

刘光聪

This page is intentionally left blank.

前言

本书定位

这是一本剖析 TensorFlow 内核工作原理的书籍，并非讲述如何使用 TensorFlow 构建机器学习模型，也不会讲述应用 TensorFlow 的最佳实践。本书将通过剖析 TensorFlow 源代码的方式，揭示 TensorFlow 的系统架构、领域模型、工作原理、及其实现模式等相关内容，以便揭示内在的知识。

面向的读者

本书假设读者已经了解机器学习相关基本概念与理论，了解机器学习相关的基本方法论；同时，假设读者熟悉 Python, C++ 等程序设计语言。

本书适合于渴望深入了解 TensorFlow 内核设计，期望改善 TensorFlow 系统设计和性能优化，及其探究 TensorFlow 关键技术的设计和实现的系统架构师、AI 算法工程师、和 AI 软件工程师。

阅读方式

初次阅读本书，推荐循序渐进的阅读方式；对于高级用户，可以选择感兴趣的章节阅读。首次使用 TensorFlow 时，推荐从源代码完整地构建一次 TensorFlow，以便了解系统的构建方式，及其理顺所依赖的基本组件库。

另外，推荐使用 TensorFlow 亲自实践一些具体应用，以便加深对 TensorFlow 系统行为的认识和理解，熟悉常见 API 的使用方法和工作原理。强烈推荐阅读本书的同时，阅读 TensorFlow 关键代码；关于阅读代码的最佳实践，请查阅本书附录 A 的内容。

版本说明

本书写作时，TensorFlow 稳定发布版本为 1.2。不排除本书讲解的部分 API 将来被废弃，也不保证某些系统实现在未来版本发生变化，甚至被删除。

同时，为了更直接的阐述问题的本质，书中部分代码做了局部的重构；删除了部分异常处理分支，或日志打印，甚至是某些可选参数列表。但是，这样的局部重构，不会影响读者理解系统的主要行为特征，更有利于读者掌握系统的工作原理。

同时，为了简化计算图的表达，本书中的计算图并非来自 TensorBoard，而是采用简化了的，等价的图结构。同样地，简化了的图结构，也不会降低读者对真实图结构的认识和理解。

英语术语

因为我所撰写的文章，是供相关专业人士阅读，而非科普读物，因此在书中保留专业领域中朗朗上口的英语术语，故意不做翻译。例如，书中直接使用 OP 的术语，而不是将其翻译为「操作」。

但是，这会造成大面积的中英混杂的表达方式。幸运的是，绝对部分所使用的英语术语都是名词，极少出现动词或者形容词。但是，无论如何都不会丢失原本的主体语义和逻辑。

万事都有例外，对于无歧义的，表达简短，且语义明确的术语，会使用中文术语表示。特殊地，对于中文术语表达存在歧义时，会同时标注中文术语和英语术语。例如，检查点 (Checkpoint)，协调器 (Coordinator)。

一般地，无歧义的中文术语表定义在表1（第iv页）。

英文	中文
Variable	变量，参数
Session	会话
Device	设备

表 1 规范约定

在线帮助

为了更好地与读者交流，在我的 Github 上建立了勘误表，及其相关补充说明。由于个人经验与能力有限，在有限的时间内难免犯错。如果读者在阅读过程中，如果发现相关错误，请帮忙提交 Pull Request，避免他人掉入相同的陷阱之中，让知识分享变得更加通畅，更加轻松，我将不甚感激。

同时，欢迎关注我的简书。我将持续更新相关的文章，与更多的朋友一起学习和进步。

1. Github: https://github.com/horance-liu/tensorflow-internals-errors
2. 简书：http://www.jianshu.com/u/49d1f3b7049e

致谢

感谢我的太太刘梅红，在工作之余完成对本书的审校，并提出了诸多修改的一件。

目录

This page is intentionally left blank.

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

- Martin Flower

1 介绍

TensorFlow 是一个支持大规模和异构环境的机器学习系统。它使用数据流图 (Dataflow Graph) 表示计算过程和共享状态，使用节点表示抽象计算，使用边表示数据流。

数据流图的节点被映射在集群中的多个机器，在一个机器内被映射在多个计算设备 (Device) 上，包括 CPU, GPU, TPU。TensorFlow 灵活的架构支持多种计算平台，包括台式机，服务器，移动终端等。

TensorFlow 最初由 Google Brain 的研究员和工程师们开发出来，用于开展机器学习和深度神经网络方面的研究，但 TensorFlow 优异的通用性使其也可广泛用于其他领域的数值计算。

1.1 前世今生

TensorFlow 是 DistBelief 的后继者，它站在巨人的肩膀上，革命性地重新设计架构设计，使得 TensorFlow 在机器学习领域一鸣惊人，在社区中产生了重大的影响。

为了更好地理解 TensorFlow 系统架构的优越性，得先从 DistBelief 谈起。

1.1.1 前世：DistBelief

DistBelief 是一个用于训练大规模神经网络的的分布式系统，是 Google 第一代分布式机器学习框架。自 2011 年以来，在 Google 内部大量使用 DistBelief 训练大规模的神经网络。

编程模型

DistBelief 的编程模型是基于层 (Layer) 的 DAG(Directed Acyclic Graph) 图。层可以看做是一种组合多个运算操作符的复合运算符，它完成特定的计算任务。

例如，全连接层完成 $f(W^T x + b)$ 的复合计算，包括输入与权重的矩阵乘法，随后再与偏置相加，最后在线性加权值的基础上实施非线性变换。

架构

DistBelief 使用参数服务器 (Parameter Server) 的系统架构，训练作业包括两个分离的

进程：无状态的 Worker 进程，用于模型训练的计算；有状态的 PS(Parameter Server) 进程，用于维护模型参数。

如图1-1（第2页）所示，在分布式训练过程中，各个模型备份 (Model Relica) 异步地从 PS 上拉取 (Fetch) 训练参数 w，当完成一步迭代运算后，推送 (Push) 参数的梯度 ∇w 到 PS 上去，并完成参数更新。

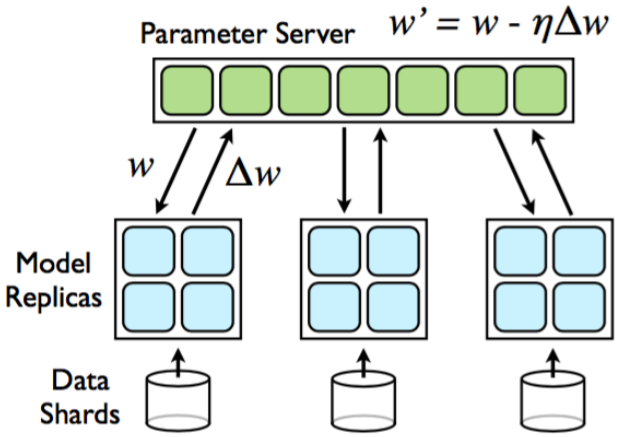

图 1-1 DistBelief: Parameter Server 架构

缺陷与不足

但是，对于高级用户，DistBelief 的编程模型，及其 Parameter Server 的系统架构，缺乏如下几个方面的扩展性。

1. 优化算法：添加新的优化算法，必须修改 Parameter Server 的实现；`get()`，`put()` 的抽象方法，对某些优化算法并不高效；
2. 训练算法：支持非前馈的神经网络具有很大的挑战性，例如包含循环的 RNN，交替训练的对抗网络，及其损失函数由分离的代理完成增强学习模型；
3. 加速设备：DistBelief 设计之初仅支持多核 CPU，并不支持 GPU；遗留的系统架构对支持新的计算设备缺乏弹性空间。

1.1.2 今生：TensorFlow

正因为 DistBelief 遗留的架构和设计，不再满足潜在的深度学习与日俱增的需求，Google 毅然决定在 DistBelief 基础上做全新的架构设计，从而诞生了 TensorFlow。

编程模型

TensorFlow 使用数据流图 (Dataflow Graph) 表示计算过程和共享状态，使用节点表示抽象计算，使用边表示数据流。如图1-2（第3页）所示，展示了 mnist 手写识别应用的数据流图。

在该模型中，前向子图使用了 2 层全连接网络，分别为 ReLU 层和 Softmax 层；随后，由 Gradients 构建了与前向子图对应的反向子图，用于训练参数的梯度计算；最后，使用‘SGD‘的优化算法，构造参数更新子图，完成参数的更新。

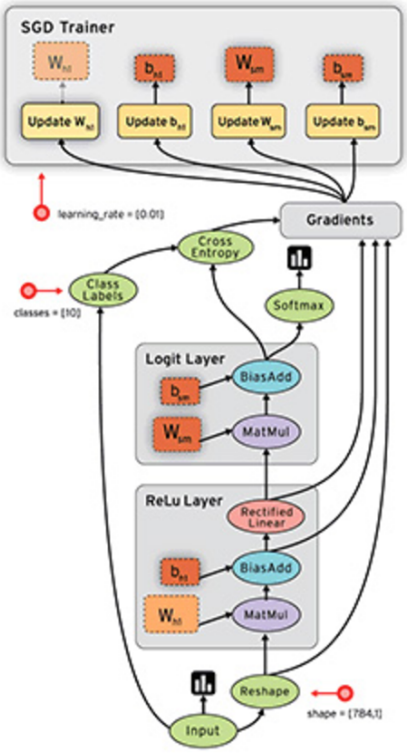

图 1-2 TensorFlow 数据流图

设计原则

1. 延迟计算：图的构造与执行分离，并推迟计算图的执行过程；
2. 原子 OP：OP 是最小的抽象计算单元，支持构造复杂的网络模型；
3. 抽象设备：支持 CPU, GPU, TPU 多种异构计算设备类型；
4. 抽象任务：基于 Task 的 PS 任务，对新的优化算法和网络模型具有良好的可扩展性。

优势

相对于其他机器学习框架，TensorFlow 具有如下方面的优势。

1. 跨平台：支持多 CPU/GPU/TPU 运算；支持台式机/服务器/移动设备；支持 Windows，Linux，MacOS；
2. 分布式：支持本地和分布式的模型训练和推理；
3. 多语言：支持 Python, C++, Java, Go 等多种程序设计语言的 API；
4. 通用性：支持各种复杂的网络模型的设计和实现；
5. 可扩展：支持 OP 扩展，Kernel 扩展，Device 扩展；

6. 可视化：使用 TensorBoard 可视化整个训练过程，包括计算图。

1.2 社区发展

TensorFlow 是目前最为流行的机器学习框架。自开源以来，TensorFlow 社区相当活跃。来自众多的非 Google 员工拥有数万次代码提交，并且每周拥有近百个 Issue 被提交；在 Stack Overflow 上也拥有上万个关于 TensorFlow 的问题被回答；在各类技术大会上，TensorFlow 也是一颗闪亮的明星，得到众多开发者的青睐。

1.2.1 开源

2015.11，Google Research 发布文章：TensorFlow: Google's latest machine learning system, open sourced for everyone，正式宣布新一代机器学习系统 TensorFlow 开源。

随后，TensorFlow 在 Github 上代码仓库短时间内获得了大量的 Star 和 Fork。如图1-3（第4页）所示，TensorFlow 的社区活跃度已远远超过其他竞争对手，逐渐成为目前最为炙手可热的机器学习和深度学习框架，已然成为事实上的工业标准。

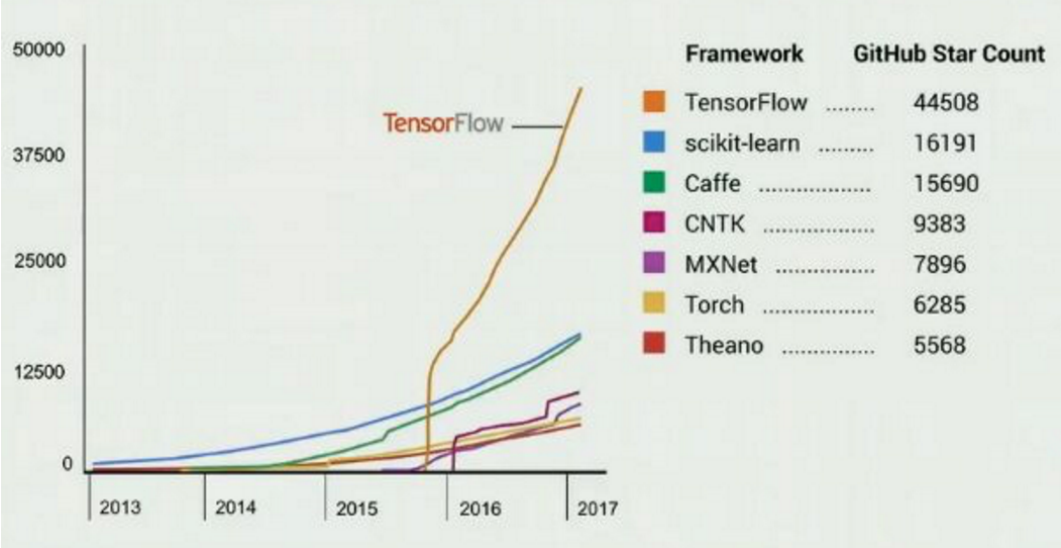

图 1-3 TensorFlow 社区活跃度

毫无疑问，TensorFlow 的开源对学术界和工业界产生了巨大的影响，其极大地降低了深度学习在各个行业中应用的难度。众多的学者，工程师，企业，组织纷纷地投入到了 TensorFlow 社区，并一起完善和改进 TensorFlow，推动其不断地向前演进和发展。

1.2.2 里程碑

TensorFlow 自 2015.11 开源依赖，平均一个多月发布一个版本。如图1-4（第5页），展示了 TensorFlow 几个重要特性的发布时间。

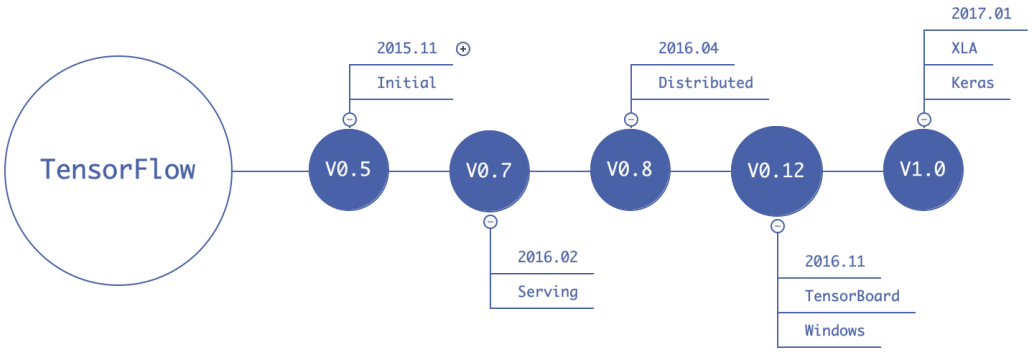

图 1-4 TensorFlow 重要里程碑

1.2.3 工业应用

TensorFlow 自开源发展一年多以来，在生产环境中被大量应用使用。在医疗方面，使用 TensorFlow 构建机器学习模型，帮助医生预测皮肤癌；在音乐、绘画领域，使用 TensorFlow 构建深度学习模型，帮助人类更好地理解艺术；在移动端，多款移动设备搭载 TensorFlow 训练的机器学习模型，用于翻译等工作。

如图1-5（第5页）所示，TensorFlow 在 Google 内部项目应用的增长也十分迅速，多个产品都有相关应用，包括：Search, Gmail, Translate, Maps 等等。

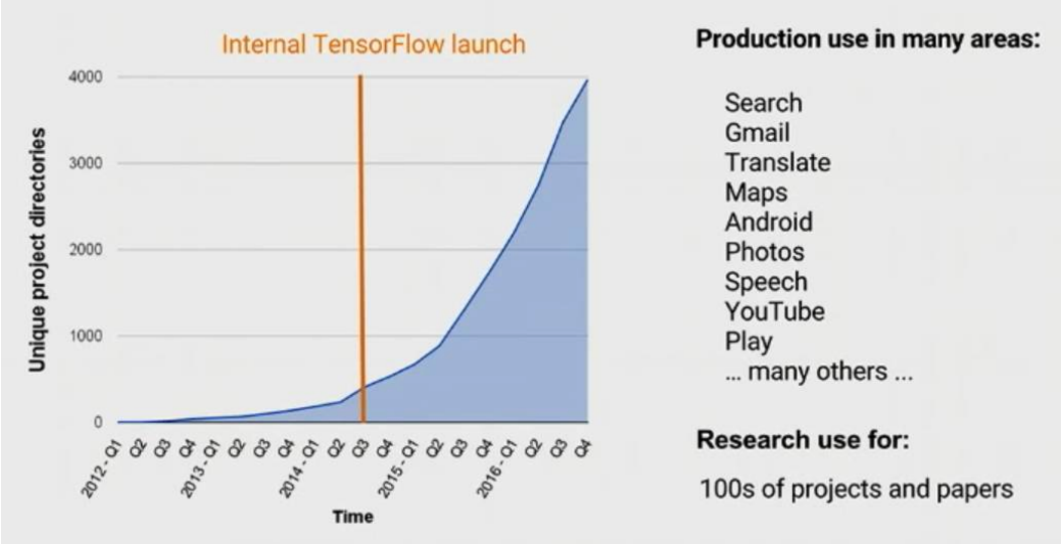

图 1-5 TensorFlow 在 Google 内部使用情况

This page is intentionally left blank.

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

- Martin Flower

2 系统架构

本章将阐述 TensorFlow 的系统架构，并一个简单的例子，讲述图结构的变换过程，加深理解 TensorFlow 运行时的工作机理。

2.1 系统架构

TensorFlow 的系统结构以 C API 为界，将整个系统分为「前端」和「后端」两个子系统：

1. 前端系统：提供编程模型，负责构造计算图；
2. 后端系统：提供运行时环境，负责执行计算图。

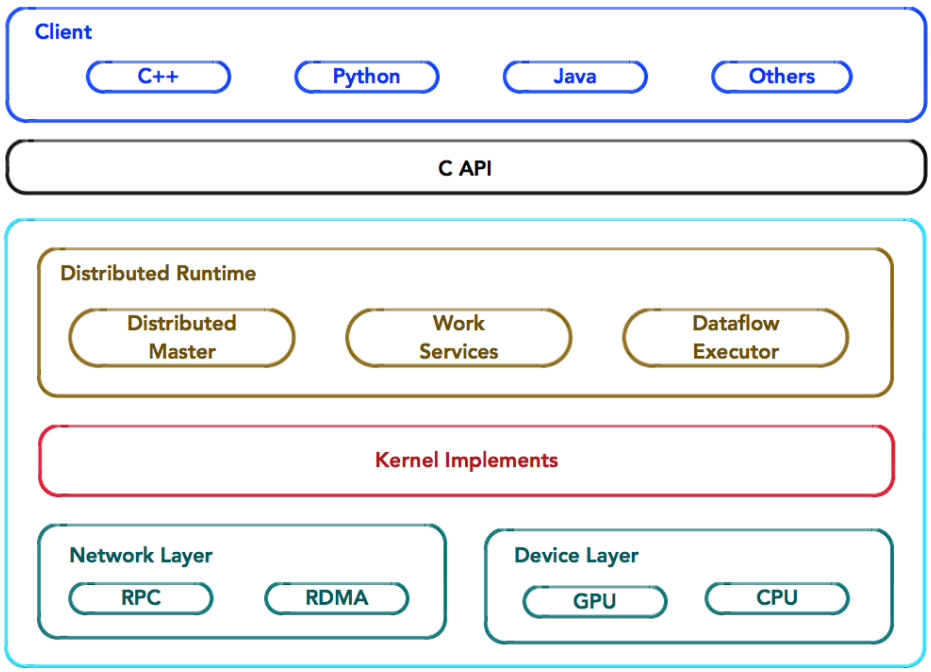

图 2-1 TensorFlow 系统架构

如图2-1（第7页）所示，重点关注系统中如下 4 个基本组件，它们是系统分布式运行时的核心。

2.1.1 Client

Client 是前端系统的主要组成部分，它是一个支持多语言的编程环境。Client 基于 TensorFlow 的编程接口，构造计算图。目前，TensorFlow 支持 Python 和 C++ 的编程接口较为完善，尤其对 Python 支持最佳；并且，对其他编程语言的编程接口的支持日益完善。

此时，TensorFlow 并未执行任何的图计算，直至与后台计算引擎建立 Session，并以 Session 为桥梁，建立 Client 与 Master 之间的通道，将 Protobuf 格式的 GraphDef 发送至 Master，启动计算图的执行过程。

2.1.2 Master

在分布式的运行时环境中，Client 根据 `Session.run` 传递整个计算图给后端的 Master；此时，计算图是完整的，常称为 Full Graph。

随后，Master 根据 Client 通过 `Session.run` 传递 `fetches` 参数列表，反向遍历 Full Graph，并按照依赖关系，对其实施剪枝，最终计算得到所依赖的「最小子图」，常称为 Client Graph。

随后，Master 负责将 Client Graph 按照任务的名称分裂 (split-by-task) 为多个「子图片段」，常称为 (Graph Partition)；其中，每个 Worker 对应一个 Graph Partition。

随后，Master 将 Graph Partition 分别注册到相应的 Worker 上，以便在不同的 Worker 上并发执行这些「子图片段」。

最后，Master 将通知所有 Work 启动相应的 Graph Partition 的执行；其中，Work 之间可能存在数据交互，Master 不参与两者之间的数据交换，它们自行通信，独立交换数据即可，直至计算完成。

2.1.3 Worker

对于每以个任务，TensorFlow 都将启动一个 Worker 实例。Worker 主要负责如下 3 个方面的职责：

1. 处理来自 Master 的请求；
2. 调度 OP 的 Kernel 实现，执行本地子图；
3. 协同任务之间的数据通信。

首先，Worker 收到 Master 发送过来的图执行命令，此时的计算图相对于 Worker 是完整的，也称为 Full Graph，它对应于 Master 的一个 Graph Partition。随后，Worker 也会执行图剪枝，得到最小依赖的 Client Graph。

随后，Worker 根据当前可用的硬件环境，包括 (GPU/CPU) 资源，按照 OP 设备的约束规范，再将 Cliet Graph 分裂 (split-by-device) 为多个 Graph Partition；其中，每个计算

设备对应一个 Graph Partition；随后，Worker 启动所有当前设备的 Graph Partition 的执行。

最后，对于每一个计算设备，Worker 将按照计算图中节点之间的依赖关系执行拓扑排序算法，并依次调用 OP 的 Kernel 实现，完成 OP 的运算 (一种典型的多态实现技术)。其中，Worker 还要负责将 OP 运算的结果发送到其他的 Work；或者接受来自其他 Worker 发送给它运算的结果，以便实现 Worker 之间的数据交互。

2.1.4 Kernel

Kernel 是 OP 在某种硬件设备的特定实现，它负责执行 OP 的具体运算。目前，TensorFlow 系统中包含 200 多个标准的 OP，包括数值计算，多维数组操作，控制流，状态管理等。

每一个 OP 根据设备类型都会存在一个优化了的 Kernel 实现。在运行时，运行时根据 OP 的设备约束贵方，及其本地设备的类型，为 OP 选择特定的 Kernel 实现，完成该 OP 的计算。

其中，大多数 Kernel 基于 Eigen::Tensor 实现。其中，Eigen::Tensor 是一个使用 C++ 模板技术，为多核 CPU/GPU 生成高效的并发代码。但是，TensorFlow 也可以灵活地直接使用 cuDNN 实现更高效的 Kernel。

此外，TensorFlow 实现了矢量化技术，在高吞吐量、以数据为中心的应用需求中，及其移动设备中，实现更高效的推理。如果对于复合 OP 的子计算过程很难表示，或执行效率低下，TensorFlow 甚至支持更高效的 Kernel 注册，其扩展性表现相当优越。

2.2 图控制

随后，通过一个最简单的例子，进一步抽丝剥茧，逐渐挖掘出 TensorFlow 计算图的控制与运行机制。

2.2.1 组建集群

如图2-2（第10页）所示。假如存在一个简单的分布式环境：1 PS + 1 Worker，并将其划分为两个任务：

1. ps0：使用 `/job:ps/task:0` 标记，负责模型参数的存储和更新；
2. worker0：`/job:worker/task:0` 标记，负责模型的训练。

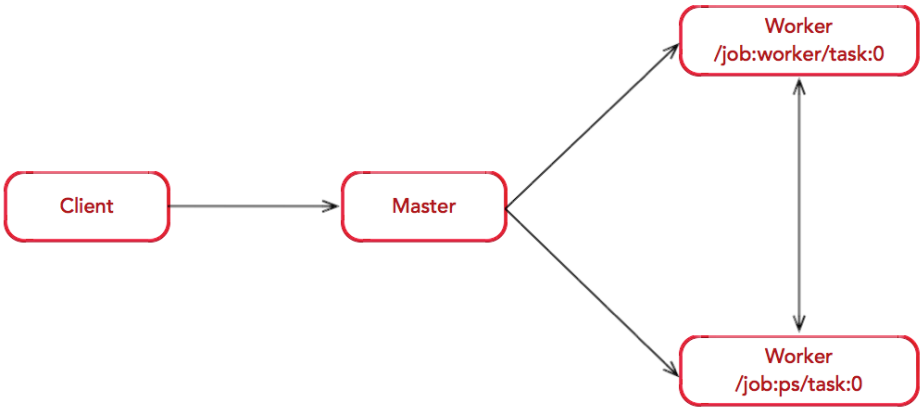

图 2-2 TensorFlow 集群：1 PS + 1 Worker

2.2.2 图构造

如图2.2.2（第10页）所示。Client 构建了一个简单计算图；首先，将 w 与 x 进行矩阵相乘，再与截距 b 按位相加，最后更新至 s。

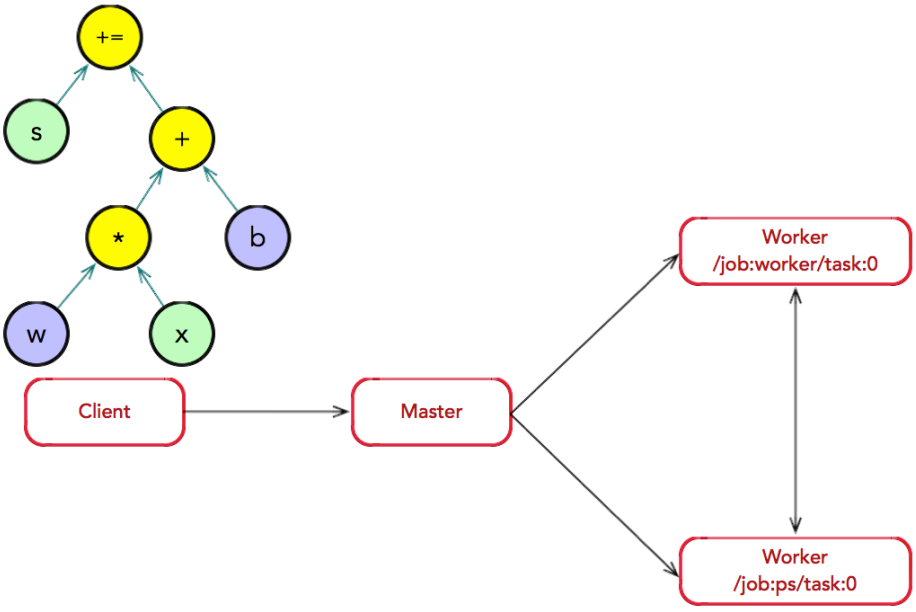

图 2-3 图构造

2.2.3 图执行

如图2.2.3（第11页）所示。首先，Client 创建一个 `Session` 实例，建立与 Master 之间的通道；接着，Client 通过调用 `Sess.run` 将计算图传递给 Master。

随后，Master 便开始启动一次 Step 的图计算过程。在执行之前，Master 会实施一系列优化技术，例如「公共表达式消除」，「常量折叠」等。最后，Master 负责任务之间的协

同，执行优化后的计算子图。

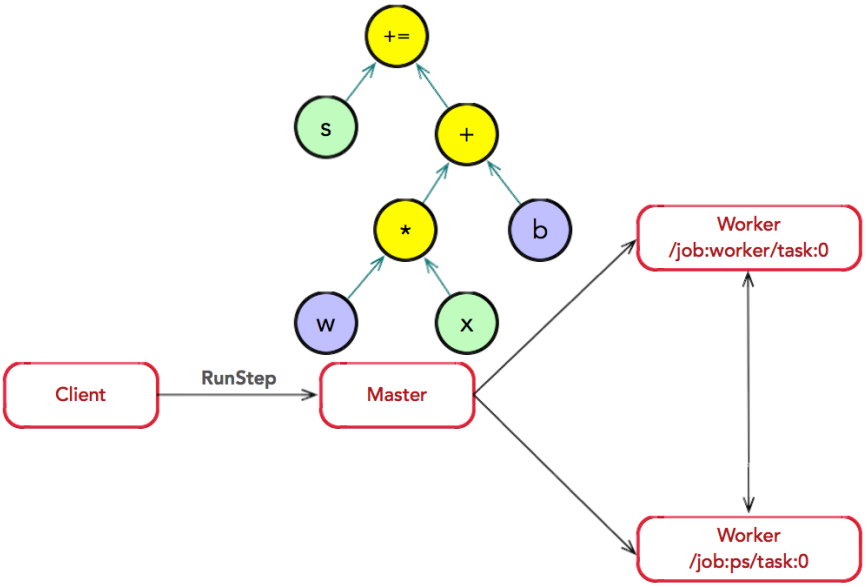

图 2-4 图执行

图分裂

如图2.2.3（第11页）所示。存在一种合理的图划分算法。Master 将模型参数相关的 OP 划分为一组，并放置在 ps0 任务上；其他 OP 划分为另外一组，放置在 worker0 任务上执行。

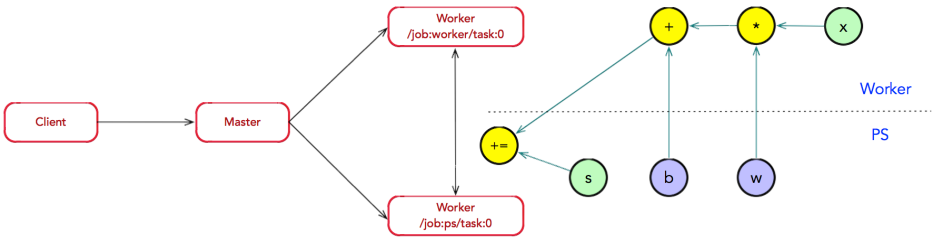

图 2-5 图分裂：按任务划分

子图注册

如图2.2.3（第12页）所示。在图分离过程中，如果计算图的边跨越任务节点，Master 将该边进行分裂，在两个任务之间插入 Send 和 Recv 节点，实现数据的传递。

其中，Send 和 Recv 节点也是 OP，这是两个特殊的 OP，由内部运行时管理和控制，对用户不可见；并且，它们仅用于数据的通信，并没有任何数据计算的逻辑。

最后，Master 通过调用 `RegisterGraph` 接口，将 Graph Partition 注册给相应的任务中，并由相应的 Worker 负责执行。

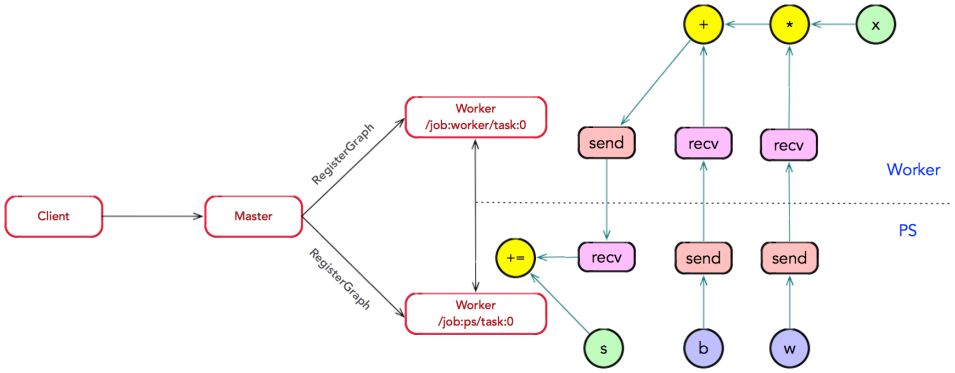

图 2-6 子图注册：插入 Send 和 Recv 节点

子图运算

如图2.2.3（第12页）所示。Master 通过调用 `RunGraph` 接口，通知所有 Worker 执行子图运算。其中，Worker 之间通过调用 `RecvTensor` 接口，完成数据的交互。

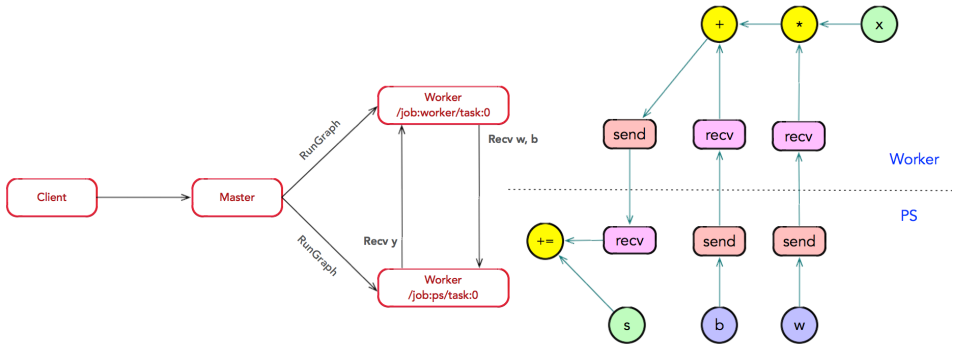

图 2-7 子图执行