实时机器学习：缺失的部分

摘要
机器学习应用程序越来越多地被部署，不仅用于使用静态模型进行预测，而且还作为涉及动态实时决策制定的反馈回路的紧密集成组件。这些应用程序提出了一组新的需求，但其中的任何一个都不难以单独实现，但其组合给现有的分布式执行框架带来了挑战：高吞吐量的毫秒级延时计算，任意任务图的自适应构建以及执行异构内核遍布不同的资源集合。我们断言这种 ML 应用需要一个新的分布式执行框架，并且提出了一种具有概念证明架构的候选方法，该方法比代表性应用的最先进的执行框架实现了 63 倍的性能提升。

1 介绍

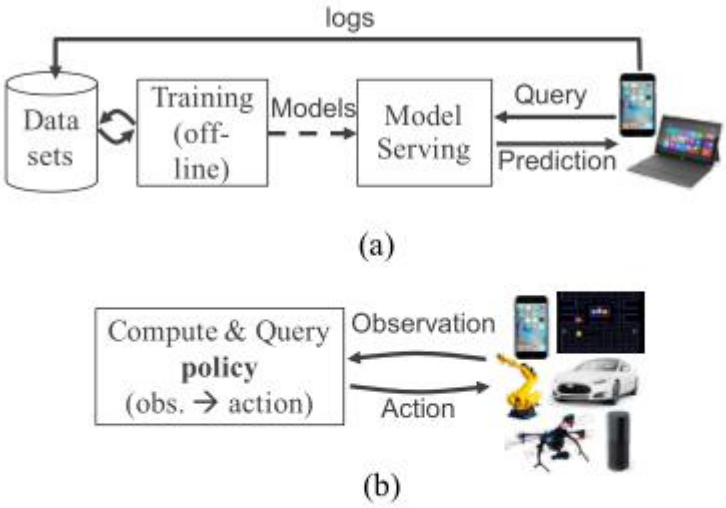

Figure 1: (a) Traditional ML pipeline (off-line training). (b) Example reinforcement learning pipeline: the system continously interacts with an environment to learn a policy, i.e., a mapping between observations and actions.

机器学习（ML）应用领域正在发生重大变化。 尽管 ML 主要侧重于基于静态模型的训练和服务预测（Figure 1a），但现在有一个强烈的转向，即 ML 模型在反馈环路中的紧密集成。事实上，ML 应用正在从监督式学习范式（在该范式中，静态模型由离线数据训练成）变成更广泛的范例，以强化学习（RL）为例，其应用可以在真实环境中运行，融合并对来自众多输入流的感测数据作出反应，执行连续的微观模拟，并通过采取影响感知环境的动作来结束循环（Figure 1b）。

由于通过与真实世界交互进行学习可能不安全，不切实际或带宽受限，因此许多强化学习系统严重依赖于物理或虚拟环境的模拟。可以在训练期间（例如，学习神经网络策略）和部署期间使用模拟。 在后一种情况下，我们可以在模拟环境与我们与真实世界交互时不断更新模拟环境，并执行许多模拟来确定下一步行动（例如，使用在线计划算法，如蒙特卡洛树搜索）。这需要能够比实时更快地执行模拟的能力。

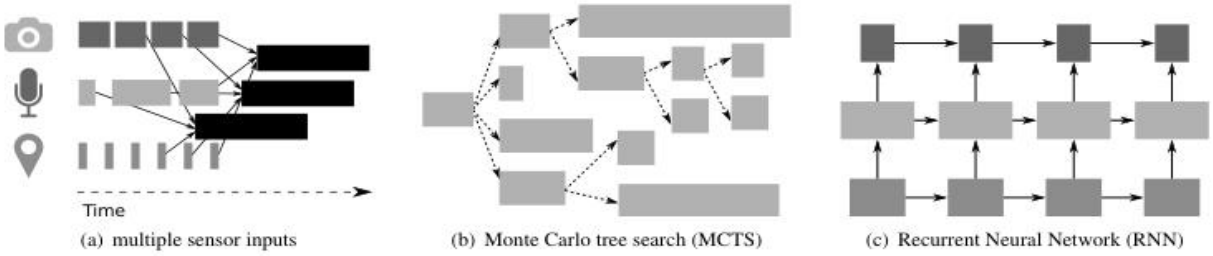

Figure 2: Example components of a real-time ML application: (a) online processing of streaming sensory data to model the environment, (b) dynamic graph construction for Monte Carlo tree search (here tasks are simulations exploring sequences of actions), and (c) heterogeneous tasks in recurrent neural networks. Different shades represent different types of tasks, and the task lengths represent their durations.

这些新兴应用程序需要新水平的编程灵活性和性能。 在不丧失现代分布式执行框架（例如，应用级容错）优点的情况下满足这些要求带来了重大挑战。我们在用 Spark，MPI 和 TensorFlow 实现 ML 和 RL 应用时强调了其中的一些挑战，并提出了三组支持这些应用的要求。这些需求对于 ML 和 RL 应用来说至关重要，而且我们相信它们非常有用。

性能要求。新兴的 ML 应用程序具有严格的延迟和吞吐量要求。

R1：低延迟。新兴 ML 应用程序的实时性，反应性和交互性需要毫秒级的端到端延迟来实现细粒度的任务执行。

R2：高吞吐量。无论是训练还是部署期间的推理，微模拟都需要支持每秒执行数百万任务的高吞吐量任务。

执行模型要求。尽管许多现有的并行执行系统在识别和优化通用计算模式方面已经取得了很大的成功，但新兴 ML 应用程序需要更大的灵活性。

R3：动态任务创建。 诸如蒙特卡罗树搜索之类的 RL 基元可能会在执行期间根据结果或其他任务的持续时间生成新任务。

R4：异构任务。 深度学习原语和 RL 模拟产生执行时间和资源需求差异很大的任务。 明确的系统（支持任务和资源的异构性）对于 RL 应用程序是必不可少的。

R5：任意数据流相关性。 类似地，深度学习基元和 RL 模拟产生任意且通常细粒度的任务依赖（不限于批量同步并行）。

实际要求。

R6：透明容错。容错仍然是许多部署场景的关键要求，支持高吞吐量和非确定性任务也是一项挑战。

R7：可调试性和性能分析。调试和性能分析是编写任何分布式应用程序最耗时的方面。ML 和 RL 应用程序尤其如此，这些应用程序通常是计算密集型和随机的。

现有框架未达到一个或多个要求（第 5 节）。我们提出了一个灵活的分布式编程模型（第 3.1 节）来实现 **R3-R5**。此外，我们提出了一个系统架构来支持这种编程模型，并且在不放弃关键实践要求（**R6-R7**）的情况下满足我们的性能要求（**R1-R2**）。所提出的系统架构（第 3.2 节）建立在两个主要组件上：一个逻辑集中控制平面和一个混合调度器。前者支持无状态分布式组件和沿袭重播。后者以自下而上的方式分配资源，将节点级和群集级调度程序之间的本地化工作分开。

其结果是微基准上的毫秒级性能以及批量同步并行（BSP）实现的代表性 RL 应用端对端速度的 63 倍。

2 激励示例

为了满足要求 **R1-R7**，考虑一个假设的应用程序，其中物理机器人试图在陌生的真实世界环境中实现目标。各种传感器可以融合视频和 LIDAR 输入，以建立机器人环境的多个候选模型（Fig. 2a）。 然后通过循环神经网络（RNN）策略（Fig. 2c）以及蒙特卡洛树搜索（MCTS）和其他在线规划算法（Fig. 2b）来实时控制机器人。使用物理模拟器以及最新的环境模型，MCTS 可以并行地尝试数百万个动作序列，从而自适应地探索最有前途的动作序列。

应用程序要求。启用这些类型的应用程序需要同时解决许多挑战。在这个例子中，等待时间要求（**R1**）是严格的，因为机器人必须实时控制。需要高任务吞吐量（**R2**）来支持 MCTS 的在线仿真以及流式传感输入。

任务异质性（**R4**）呈现出许多规模：一些任务运行物理仿真器，其他任务处理各种数据流，另一些任务使用基于 RNN 的策略进行一些计算操作。 即使类似的任务也可能表现出持续时间的显著变化。例如，RNN 由每个“层”的不同功能组成，每个“层”可能需要不同的计算量。或者，在模拟机器人动作的任务中，模拟长度可能取决于机器人是否达到其目标。

除了任务的异质性之外，任务之间的依赖关系可能很复杂（**R5**，Figs. 2a 和 2c），并且难以表现为批量 BSP 任务。

动态构建任务及其依赖关系（**R3**）至关重要。仿真将自适应地使用最新的环境模型，并且 MCTS 可能会选择启动更多探索特定子树的任务，具体取决于它们有多么有希望或计算速度有多快。因此，数据流图必须动态构建，以使算法适应实时约束和机会。

3 建议的解决方案

在本节中，我们概述了针对实时 ML 应用程序的分布式执行框架和满足 **R1-R7** 要求的编程模型的建议。

3.1 API 和执行模型

为了支持执行模型需求（**R3-R5**），我们概述了一个 API，该 API 允许将任意函数指定为可远程执行的任务，并具有数据流依赖关系。

1.任务创建是非阻塞的。创建任务时，将立即返回表示任务的最终返回值的 future，并且异步执行任务。

2.可以将任意函数调用指定为远程任务，从而可以支持任意执行内核（**R4**）。任务参数可以是常规值或 futures。当参数是 future 时，新创建的任务将依赖于产生 future 的任务，从而启用任意 DAG 依赖关系（**R5**）。

3.任何任务执行都可以创建新的任务而不会阻止其完成。任务吞吐量因此不受任何一个工作者（**R2**）的带宽限制，并且计算图是动态构建的（**R3**）。

4.通过在相应的 future 调用 get 方法可以获得任务的实际返回值。这会阻塞，直到任务完成执行。

5.等待方法有一系列 future，一个超时和一些数值。当超时或完成请求的数量时，它会返回已完成任务的 futures 的子集。

等待原语允许开发者当超时时指定时间延迟要求（**R1**），从而计算任意大小的任务（**R4**）。 这对 ML 应用程序很重要，因为其中一个后续的任务可能会产生微不足道的算法改进，虽然会阻止整个计算。这个原语增强了我们根据执行时间（**R3**）动态修改计算图的能力。

为了补充细粒度的编程模型，我们建议使用一个数据流执行模型，在该模型中，当且仅当它们的依赖完成执行时，才能执行任务。

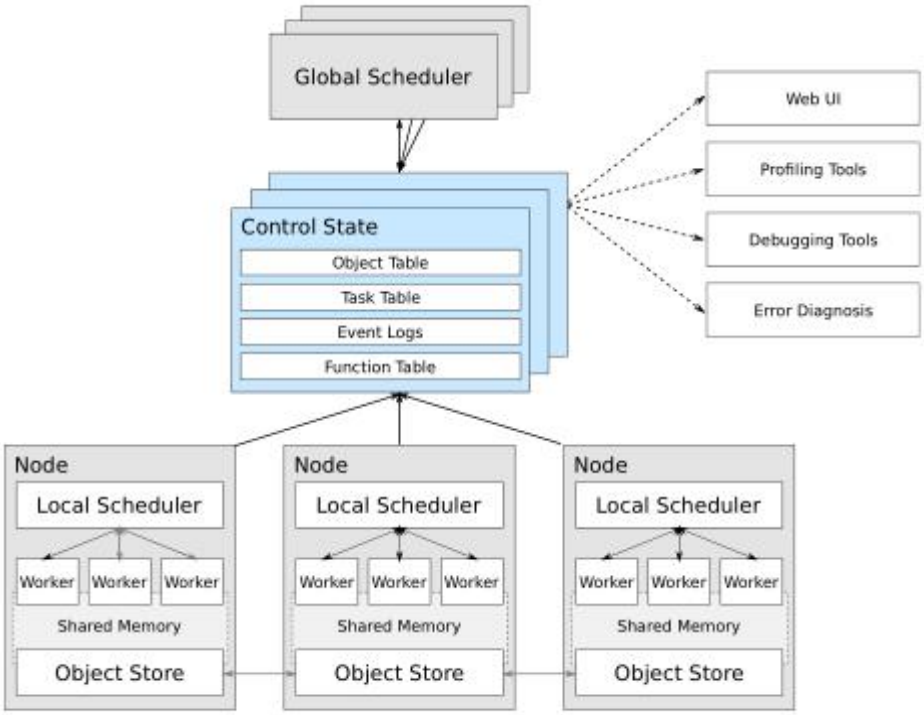

Figure 3: Proposed Architecture, with hybrid scheduling (Section 3.2.2) and a centralized control plane (Section 3.2.1).

3.2 建议的架构

我们提出的体系结构由在集群中的每个节点上运行多个工作进程，每个节点上有一个本地调度程序，整个集群中有一个或多个全局调度程序以及用于在工作节点之间共享数据的内存对象存储（见 Figure 3）组成。

启用 **R1-R7** 的两个主要架构特征是混合调度程序和集中式控制平面。

3.2.1 集中控制状态

如 Figure 3 所示，我们的架构依赖于逻辑集中的控制平面。为了实现这种架构，我们使用一个数据库，该数据库为系统的控制状态提供（1）存储，以及（2）发布-订阅功能，以使各种系统组件能够相互通信。

这种设计实际上使系统的任何组件（数据库除外）都成为无状态的。所以只要数据库具有容错性，我们可以通过简单地重新启动发生故障的组件来从组件故障中恢复。此外，数据库存储计算过程，这使我们能够通过重新计算来重建丢失的数据。因此，这种设计具有容错能力（**R6**）。数据库还可以轻松编写工具来分析和检查系统状态（**R7**）。

为了达到吞吐量要求（**R2**），我们分割数据库。由于我们只需要精确的匹配操作，并且由于密钥计算为散列值，因此分片很简单。我们早期的实验表明，这种设计可以实现亚毫秒的调度延迟（**R1**）。

3.2.2 混合调度

我们对延迟（**R1**），吞吐量（**R2**）和动态图形构建（**R3**）的要求自然会促使需要混合调度程序，其中本地调度程序将任务分配给工作节点或将责任委派给一个或多个全局调度程序。

工作节点将任务提交给他们的本地调度程序，这些调度程序决定将任务分配给同一物理节点上的其他工作节点，或将任务提交到全局调度程序。全局调度程序可以根据相关因素的全局信息将任务分配给本地调度程序，包括本地对象和可用的资源。

由于任务可能会创建其他任务，因此可调度工作可能来自集群中的任何工作节点。通过显著减少全局调度程序负载，避免通信开销和吞吐量（**R2**），启用任何本地调度程序来处理本地生成的任务而不涉及全局调度程序，可降低延迟（**R1**）。这种混合调度方案非常适合最近大型多核服务器的趋势。

4 可行性

为了证明这些 API 和架构建议原则上可以支持 **R1-R7** 要求，我们使用第 3 节中概述的初步系统设计提供了一些简单示例。

4.1 微基准延迟

使用我们的原型系统，创建一个任务，意味着该任务将在 35 秒内异步提交并返回 future。一旦任务完成执行，它的返回值可以在大约 110 秒内检索出。从提交空执行任务到检索其返回值的端到端时间，在本地调度任务时大约为 290 秒，而在远程节点上调度任务时为 1ms。

4.2 强化学习

我们实现了一个简单的工作负载，在该工作负载中，RL agent 被训练来玩 Atari 游戏。工作负载会在并行采取行动的阶段之间交替进行模拟，并且在 GPU 上并行计算行动。尽管该示例具有 BSP 特性，但由于系统开销，Spark 中的实现比单线程实现慢 9 倍。我们原型中的实现比单线程版本快 7 倍，比 Spark 实现快 63 倍。

这个例子展现了两个关键特征。首先，任务非常小（每个约 7ms），这使得任务开销很低。其次，这些任务的持续时间和资源需求是不同的（例如，CPU 和 GPU）。

此示例只是 RL 工作负载的一个组件，通常会用作更复杂（非 BSP）工作负载的子例程。例如，使用 wait 原语，我们可以调整示例以按照完成的顺序处理仿真任务，以便更好地通过 GPU 上的操作计算来管理仿真执行，或者运行嵌套在更大自适应范围内的整个工作负载超参数搜索。使用第 3.1 节中描述的 API，这些更改都很简单，只涉及一些额外的代码行。

5 相关工作

静态数据流系统在分析和 ML 中非常成熟，但它们需要预先指定数据流图，例如通过驱动程序。有些像 MapReduce 和 Spark 强调 BSP 执行，而另一些像 Dryad 和 Naiad 则支持复杂的依赖结构（**R5**）。其他如 TensorFlow 和 MXNet 则针对深度学习工作负载进行了优化。但是，这些系统都不支持动态扩展数据流图以响应输入数据和任务进度（**R3**）的能力。

像 CIEL 和 Dask 这样的动态数据流系统支持许多与静态数据流系统相同的功能，并为动态任务创建（**R3**）提供额外的支持。这些系统符合我们的执行模式要求（**R3-R5**）。然而，它们的架构限制（例如完全集中式调度）使得低延迟（**R1**）必须经常牺牲高吞吐量（**R2**）（例如，通过批处理）为代价，而我们的应用程序需要两者。

其他系统如 Open MPI]和 actor 模型变体 Orleans 和 Erlang 提供了低延迟（**R1**）和高吞吐量（**R2**）分布式计算。尽管这些系统原则上提供了支持我们的执行模型需求（**R3-R5**）的原语并且已经用于 ML，但系统级功能所需的大部分逻辑（例如容错（**R6**））和本地感知任务调度必须在应用程序级别实施。

6 结论

机器学习应用程序正在不断发展，需要具有毫秒级延迟和高吞吐量的动态数据流并行性，这对现有框架构成了严峻的挑战。我们概述了支持这种新兴的实时 ML 应用程序的要求，并且我们提出了一种编程模型和架构设计来解决关键需求（**R1-R5**），而不影响现有需求（**R6-R7**）。初步的概念验证结果证实了具有代表性的 RL 应用的毫秒级系统开销和有意义的加速。