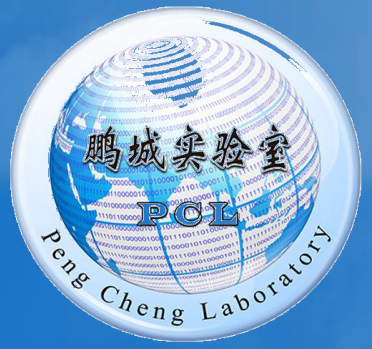


自动并行技术简介

鹏城实验室网络智能部开源所 王进

2021年08月

一、分布式深度学习综述

- Meng Wang et al., 2020, A Survey on Large-scale Machine Learning
- JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learning
- Shuo Ouyang et al., 2020, Communication optimization strategies for distributed deep neural network training: A survey
- RUBEN MAYER et al., 2019, Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools

二、自动并行技术及已有框架

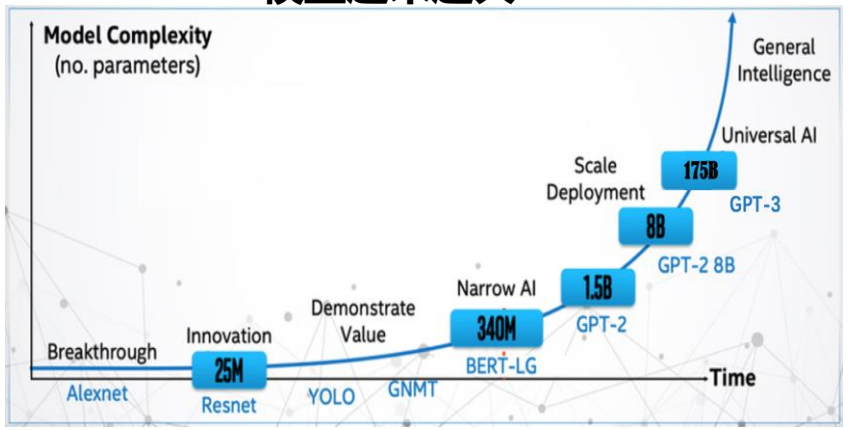
- 自动并行问题定义
- MindSpore自动并行
- 自动并行框架FlexFlow
- 分布式训练框架Whale

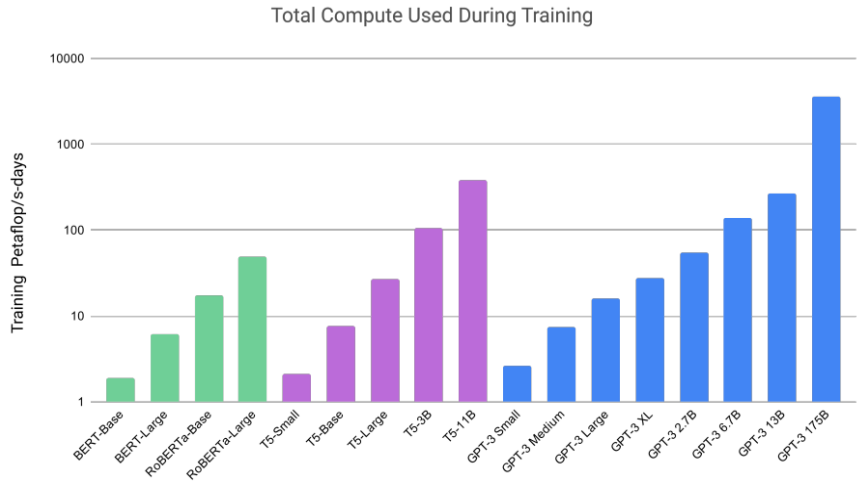
三、优化思考

- 并行搜索空间建模
- 边训练边搜索模式
- 自感知自优化配置

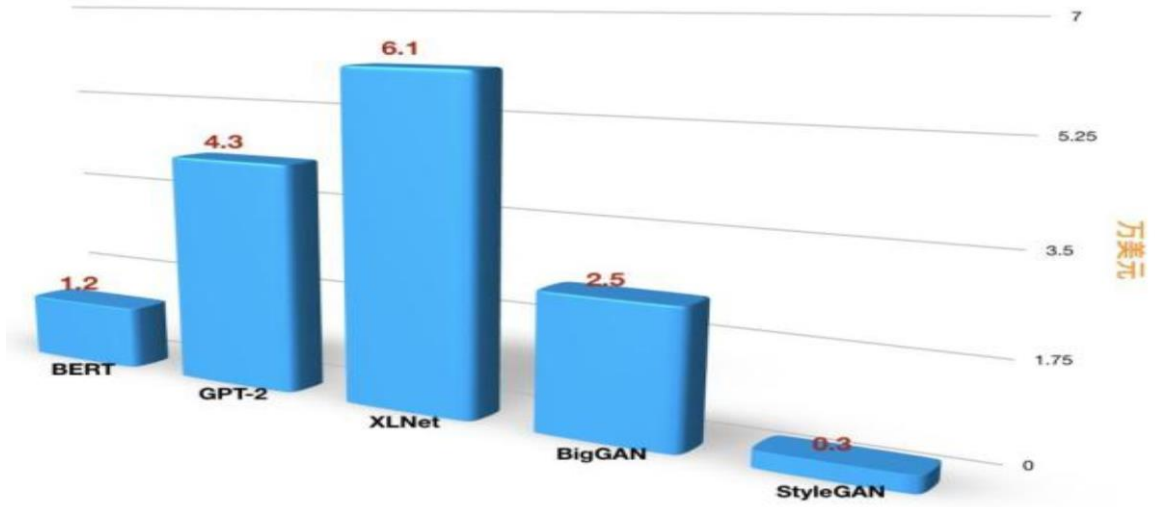
一、分布式深度学习综述

模型越来越大





算力需求越来越大



模型训练成本高：

- ✓ **0.25 万美元-5 万美元（1.1 亿参数的模型）**
- ✓ **1 万美元-20 万美元（2.4 亿参数的模型）**
- ✓ **8 万美元-160 万美元（15 亿参数的模型）**

大机器+大数据 = 大模型?

- **大模型是一种统筹了数据预处理、模型架构、算法训练与优化的一套完整体系，即便有足够的算力、原始数据、原始模型，也并不意味能够做出真正跑得通的大模型，这其中非常考验技术研发和协同能力**

算法与编程难题

- 简洁，模块化，可复现，接口灵活
- 神经网络训练速度提升

并行策略难题

- 模型并行
- 数据并行

集群调优难题

- 硬件规划，网络拓扑
- 软件规划，操作系统

图算融合难题

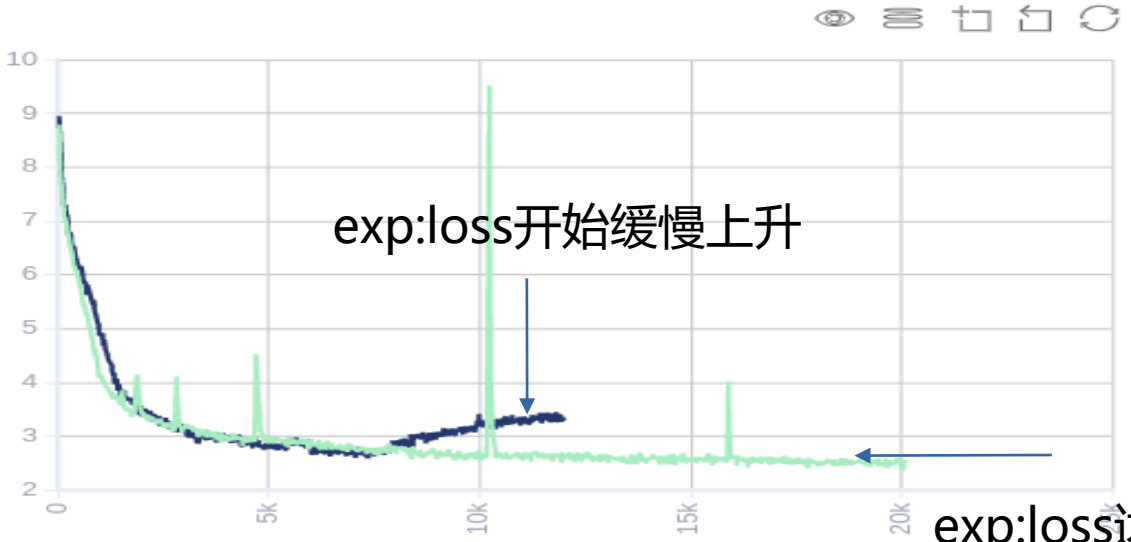
- 图层的算法描述转换
- 算子的图层优化
- 跨算子的全局性能优化

线性扩展加速难题

- 模型性能损耗控制

大规模训练挑战：算法与编程难题

- **海量数据的快速加载、调参、训练断点恢复、溢出预警抑制、scale动态缩放等是大模型目前训练的难题**



name	lr	Warm up	Beta2	Eps
exp(蓝色)	1e-4	2000	0.95	1e-8
exp(绿色)	1e-4	2000	0.999	1e-8

exp:loss达到2.5，出现溢出

以一个大模型训练过程为例

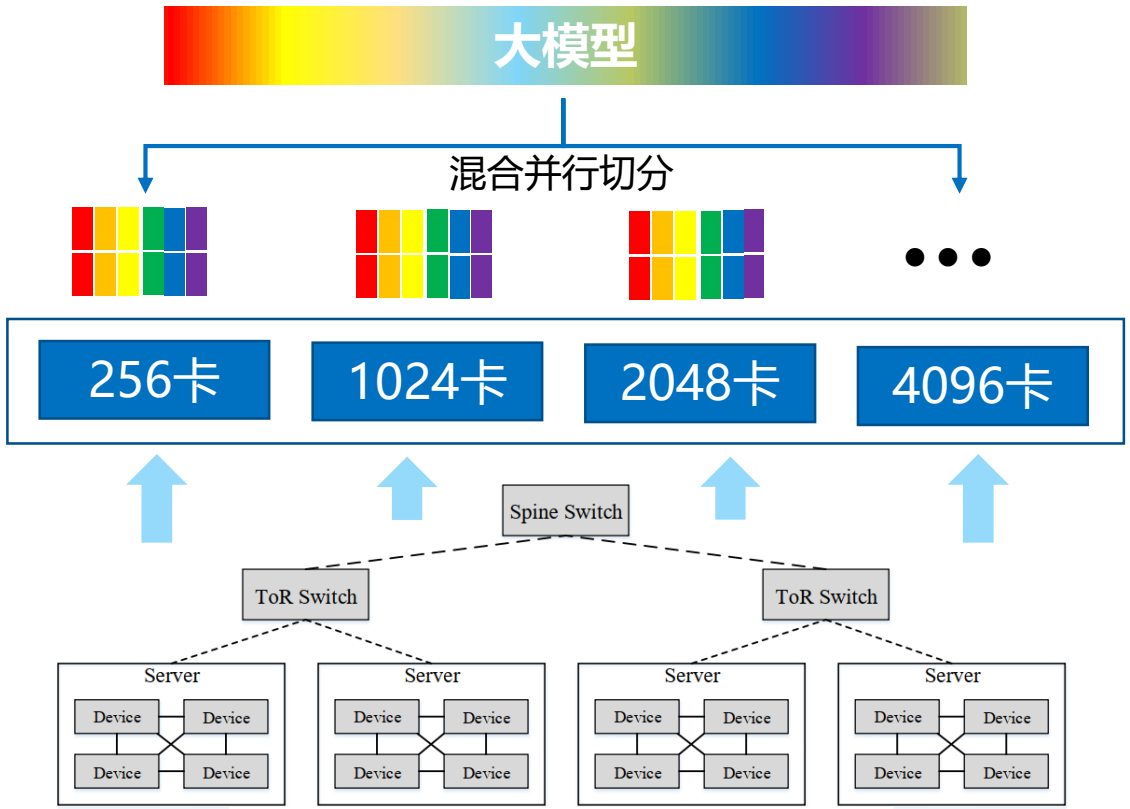
✓ bata2超参:优化器的beta2影响loss下降的快慢，但是太大的beta2会造成loss的毛刺，在beta2为一定值时，增大lr或许可以使loss下降更快

✓ Eps: 学习率衰减最小值,如果调大eps(例1e-8 →1e-4)，可以避免溢出，但是loss曲线会较高

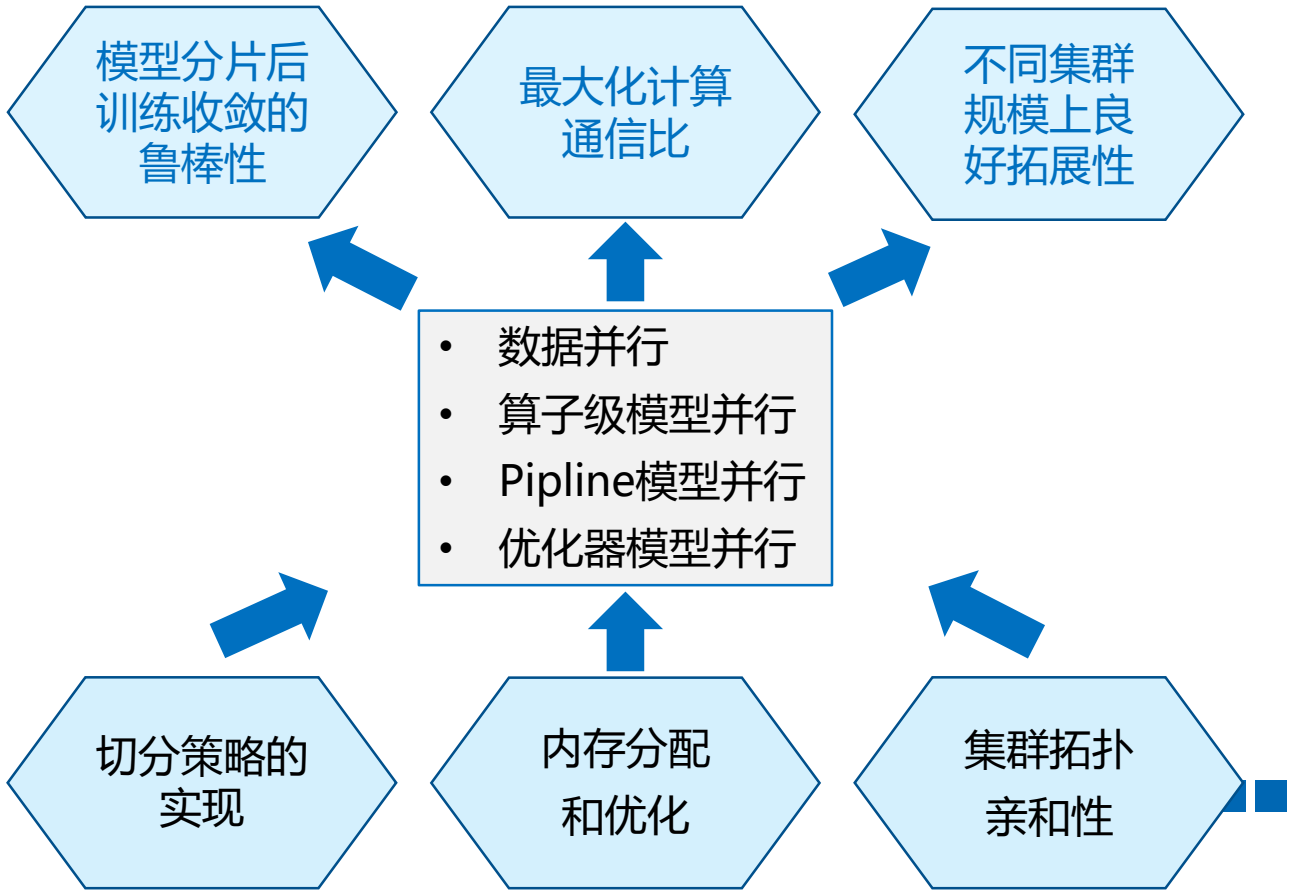
总结：大模型训练时，参数的微小变化、模型的结构、数据集的组成与分布都可能影响loss的收敛情况，需要大量反复的实验，最终考验的是算法、数据、框架、资源调度等全栈和全流程的综合能力。

大规模训练挑战：并行策略难题

- 200B参数量大模型，参数占用745GB内存，训练过程需要3500GB+内存（权重+激活+优化器状态），一个模型需要100多张卡才能放下，**怎样对模型进行切分，4D混合并行如何最大化计算通信比**，对每一个算子切分、分配、组合进行精细化的操控，都具有非常大的考验。



集群拓扑结构

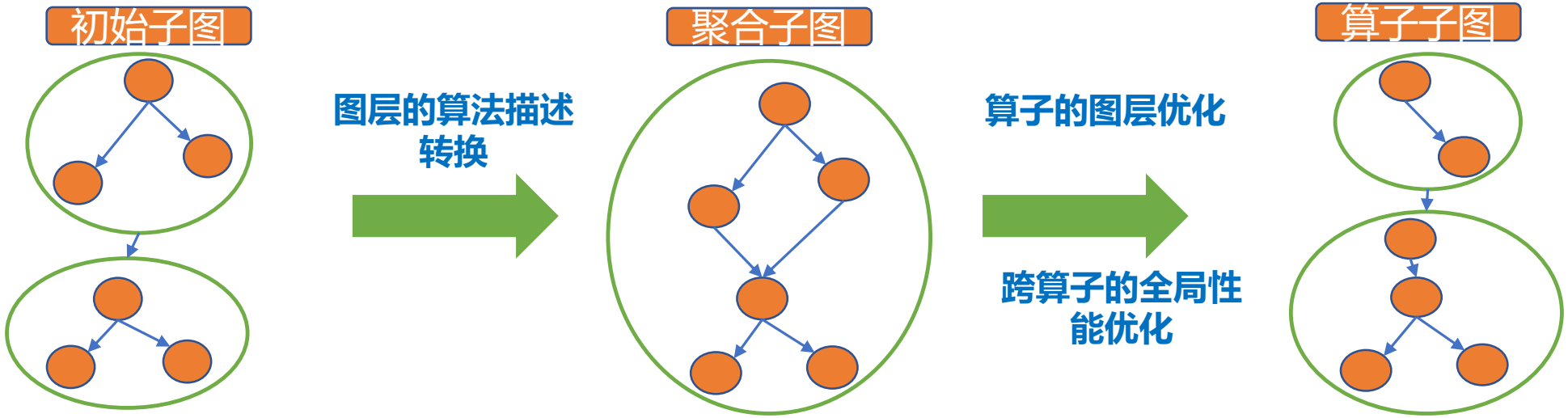


大规模训练挑战：图算融合难题

- **图算融合方案：**对“用户使用角度的易用性算子”进行重组融合，然后转换为“硬件执行角度的高性能算子”，充分发挥硬件性能。

- **难题：**
  1. **图层的算法描述转换：** 如何对整张图进行抽象化，方便子图的聚合、重构操作。
  2. **算子的图层优化：**
  3. **跨算子的全局性能优化：**

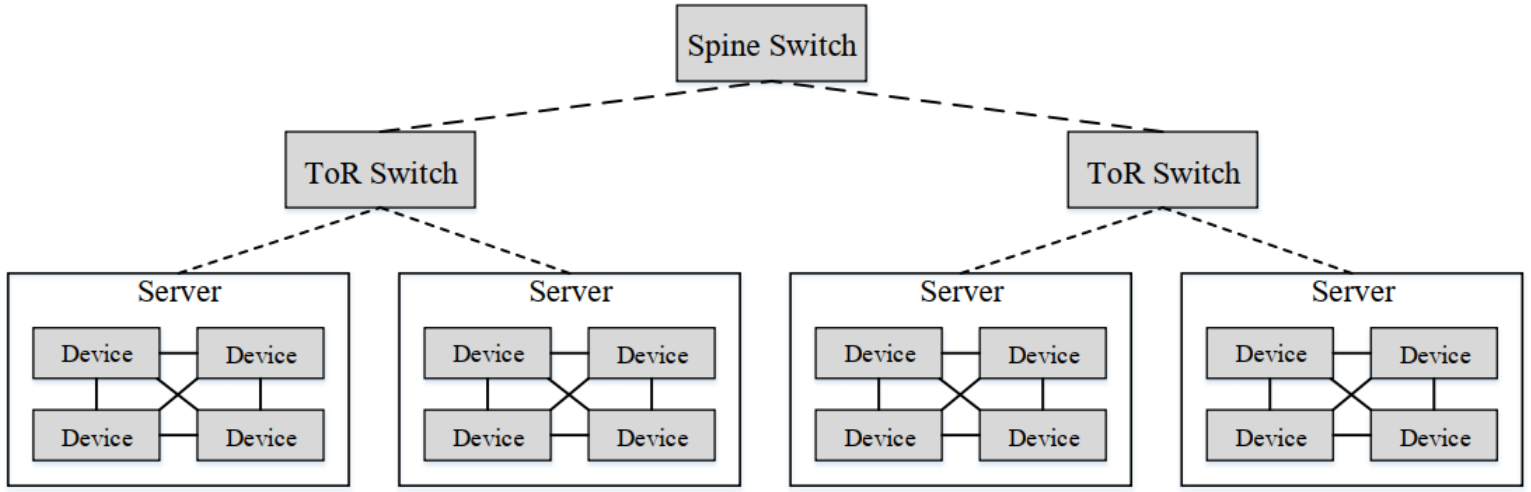
  如何对图层进行聚合和重新拆分，大模型的算子搜索空间巨大，对优化方案要求很高



大规模训练挑战：集群调优难题

- **难题：不同的网络拓扑结构会对性能有很大影响**
  - 大模型一张卡放不下，需要把模型切分到多张卡上
  - 模型切分到多张卡会引入通信，如何实现通信最小化或者计算通信比最大化
  - 如何管理集群，提供集群逻辑拓扑
  - 模型切片如何调度部署到各个节点，使得通信模式匹配集群拓扑，发挥带宽

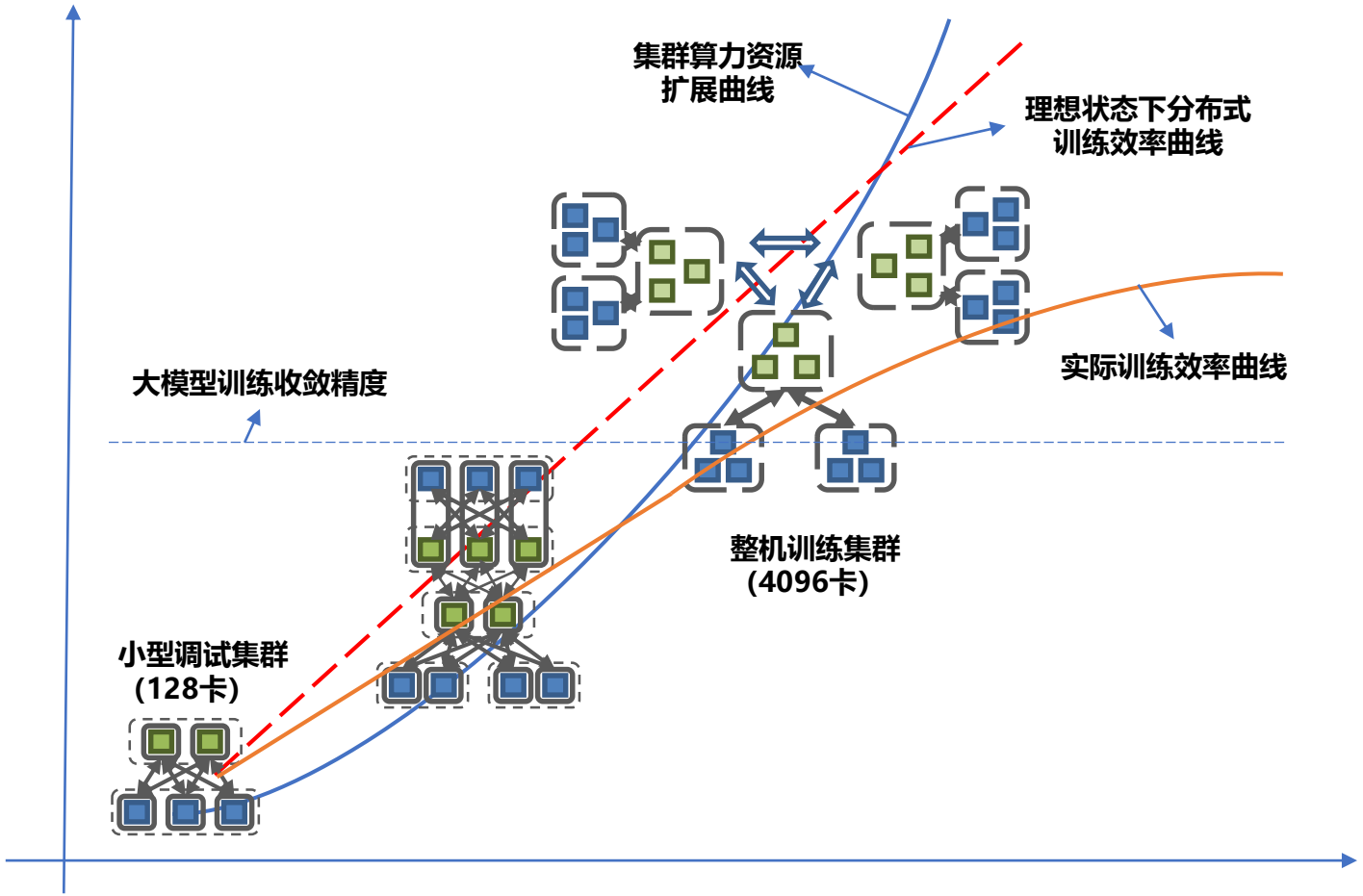
如何将模型切片依据集群拓扑进行调度非常有挑战性



- **方案：根据组网特性，服务器内和服务器间采用不同的并行模式，充分发挥带宽**

大规模训练挑战：线性扩展加速难题

- **在大模型训练收敛精度稳定的条件下，如何保障大模型不同规模集群上的快速扩展，缩小集群算力资源扩展曲线与训练效率曲线差异，提高算力资源线性扩展带来的训练效率增益，是大模型训练重要难题。**



- ✓ 不同集群规模线性扩展带来集群拓扑差异，影响分布式并行策略选择和切换
- ✓ 集群线性扩展需要针对性的进行网络配置及调优，增加训练规模切换的复杂度
- ✓ 小规模调试集群与整机训练集群对于大模型训练收敛精度影响的不确定性和差异性，为整机训练带来挑战

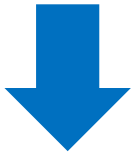
一、分布式深度学习综述

分布式深度学习出发点：

（1）单机算力不够、模型太大导致单卡放不下

（2）数据量大、模型大、训练迭代多，单机效率太慢

（3）采用更多机器进行并行计算



✓ 如何更便捷

✓ 如何更高效

解决方式：

- 数据并行？模型并行？
- 梯度压缩？
- 通信协议优化？

✓ 上述过程是否可以由算法开发者感知不到的软件模块自动化完成？

自动并行技术：深度学习模型的自动化分布式并行训练（兼顾效率）

一、分布式深度学习综述——三种视角

提供三种视角去概括分布式深度学习整体研究内容框架：

（1）如何量化分布式深度学习效率？

（2）分布式深度学习研究哪些要素？

（3）分布式深度学习系统的通信如何优化？

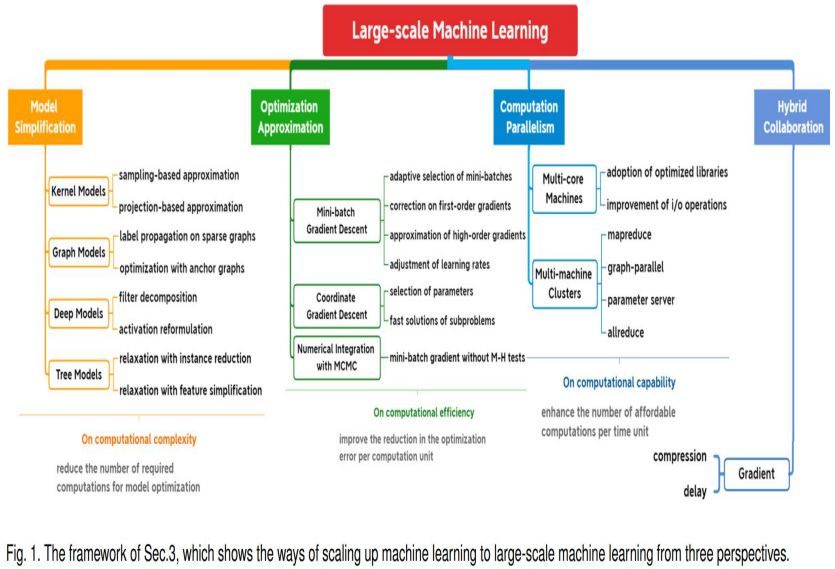
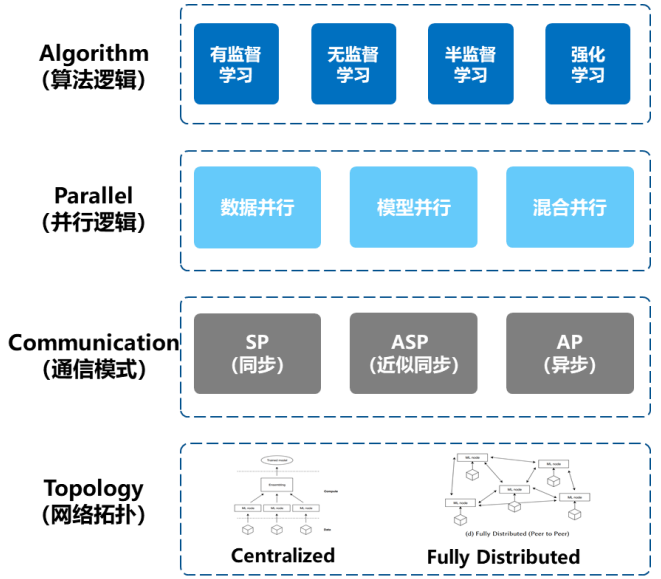
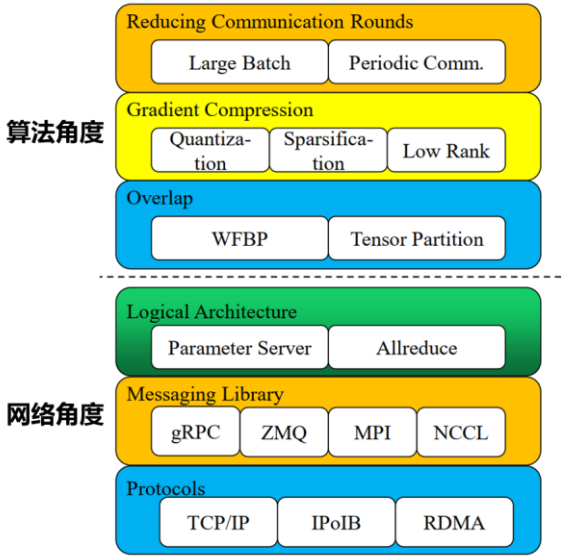


Fig. 1. The framework of Sec.3, which shows the ways of scaling up machine learning to large-scale machine learning from three perspectives.





视角1：量化分布式深度学习复杂度边界　**视角2：分布式深度学习关键要素**　**视角3：分布式深度学习通信优化角度**

一、分布式深度学习综述[Meng Wang et al., 2020, A Survey on Large-scale Machine Learning]

From effectiveness to efficiency：从如何量化定义通用机器学习问题求解的效率出发

- ✓ present three perspectives to **quantify the efficiency** based on an acknowledged error decomposition on effectiveness.
- ✓ excess error ε obtained within an allotted time cost T_{max} can be decomposed into three terms: $\boldsymbol{\varepsilon = \varepsilon_{app} + \varepsilon_{est} + \varepsilon_{opt}}$
- ✓ 大规模机器学习三个主要研究方向：（1）如何优化$\boldsymbol{\varepsilon_{app}}$：模型计算复杂度简化；（2）如何优化$\boldsymbol{\varepsilon_{est}}$：模型计算效率更高的近似优化；（3）如何优化$\boldsymbol{\varepsilon_{opt}}$：计算并行

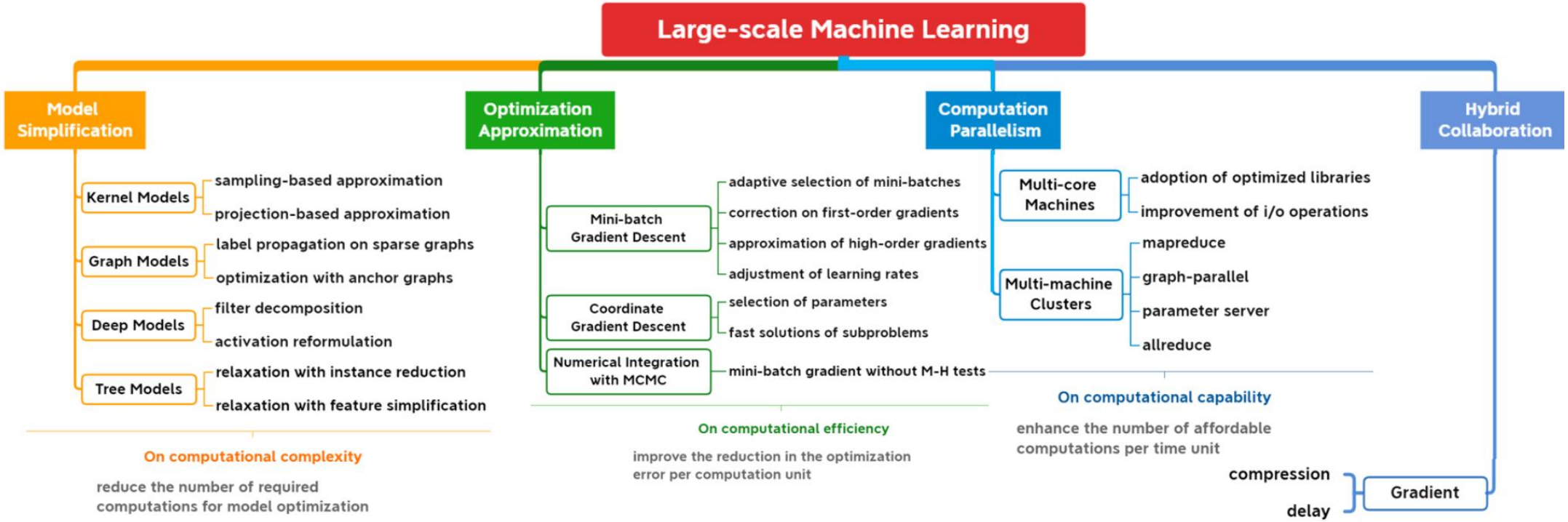
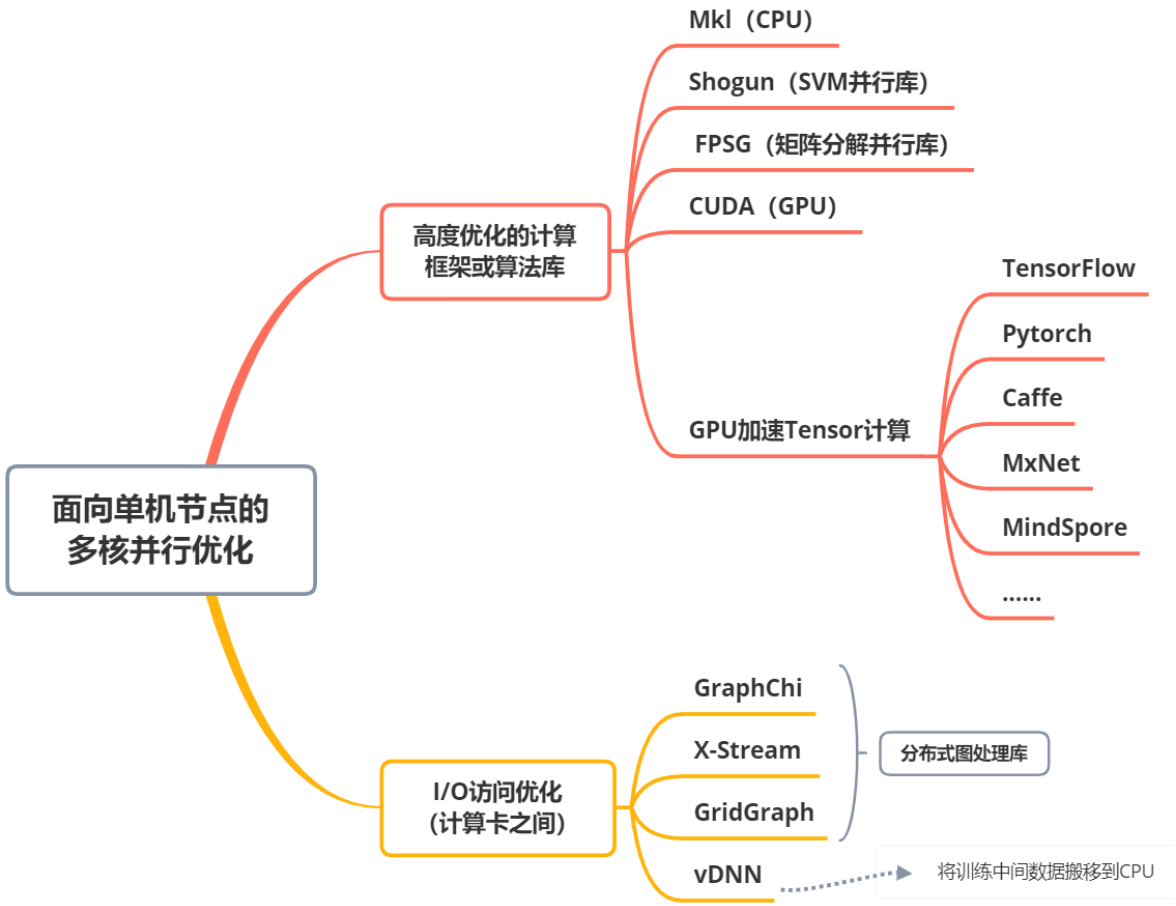


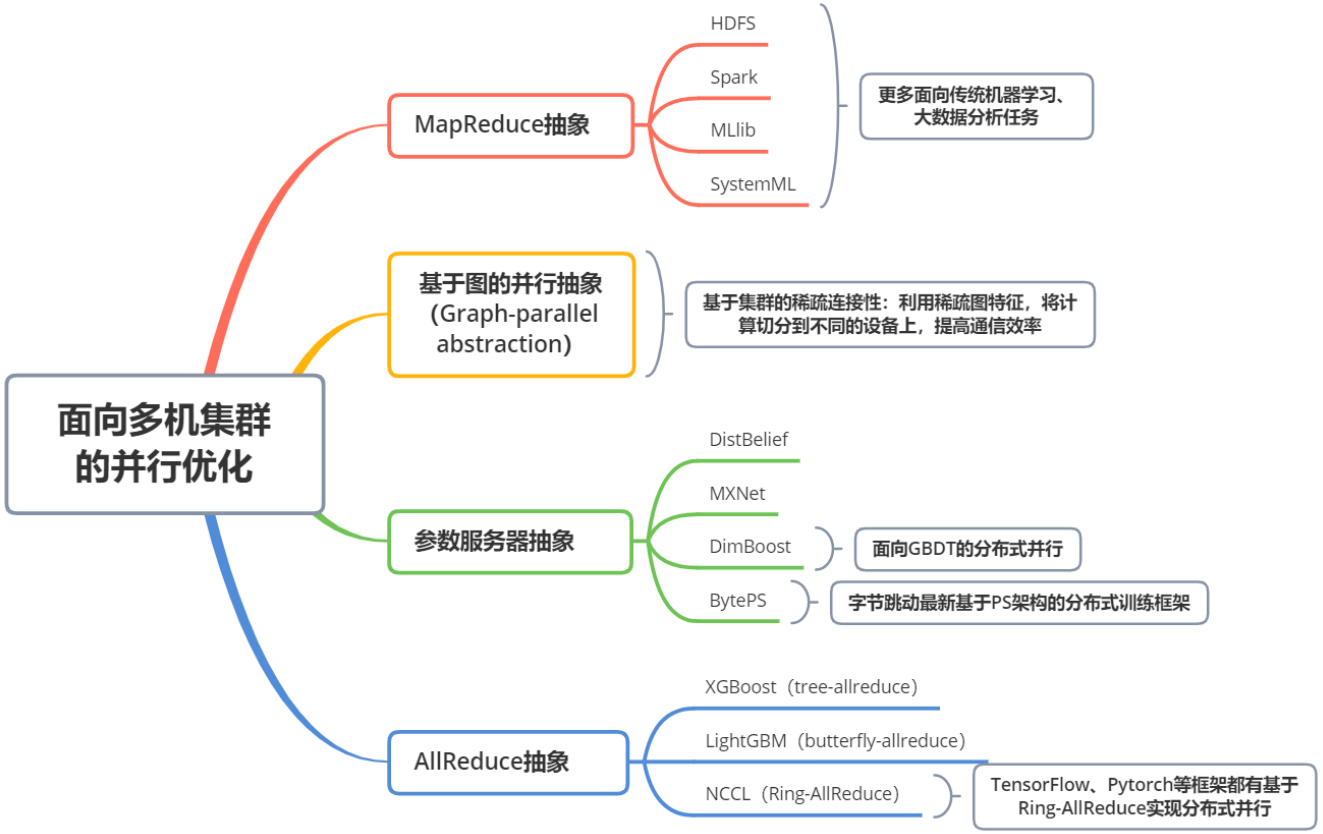
Fig. 1. The framework of Sec.3, which shows the ways of scaling up machine learning to large-scale machine learning from three perspectives.

一、分布式深度学习综述[Meng Wang et al., 2020, A Survey on Large-scale Machine Learning]

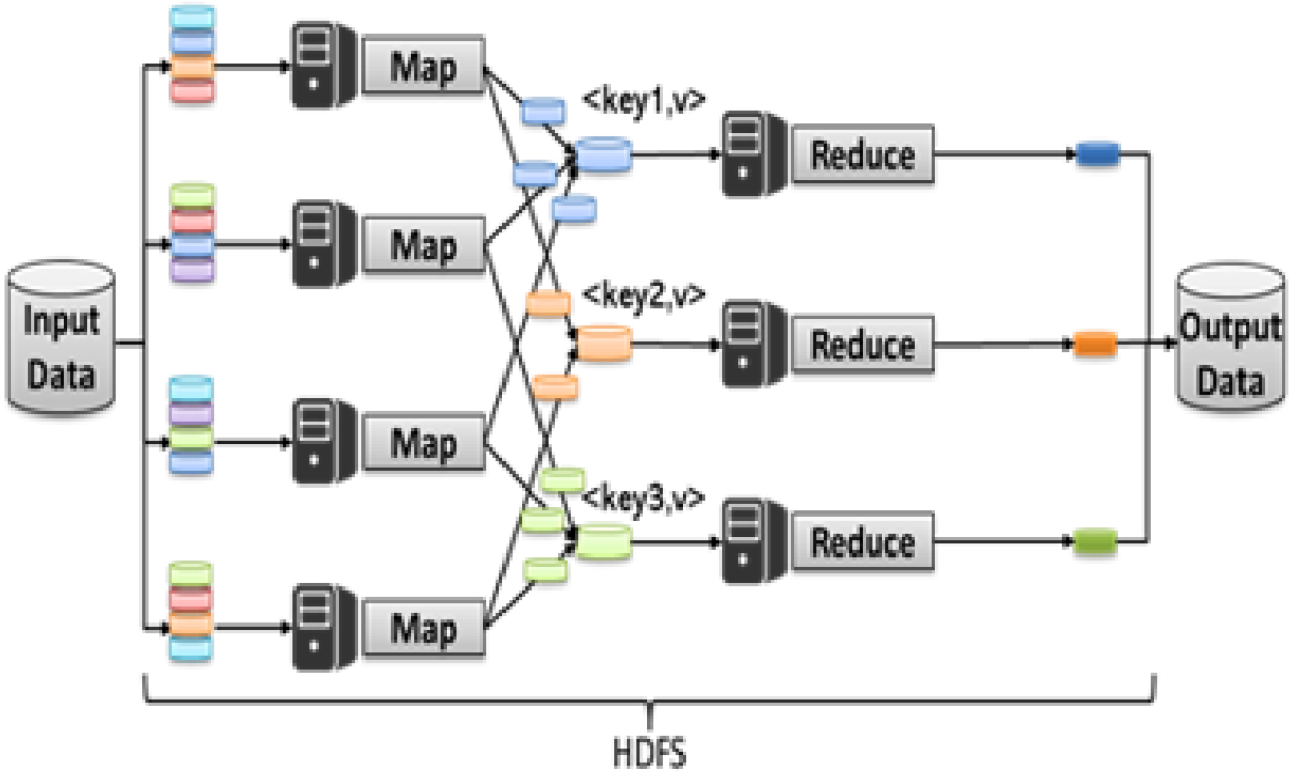
Computation Parallelism Motivation:

- ✓ 如何更高效率将算法的子任务同步运行在多个计算设备上（mutually-independent subtasks can be processed simultaneously over multiple computing devices）
- ✓ 由此产生两个层级的并行优化：**面向单机节点的多核并行优化、面向多机集群的并行优化**

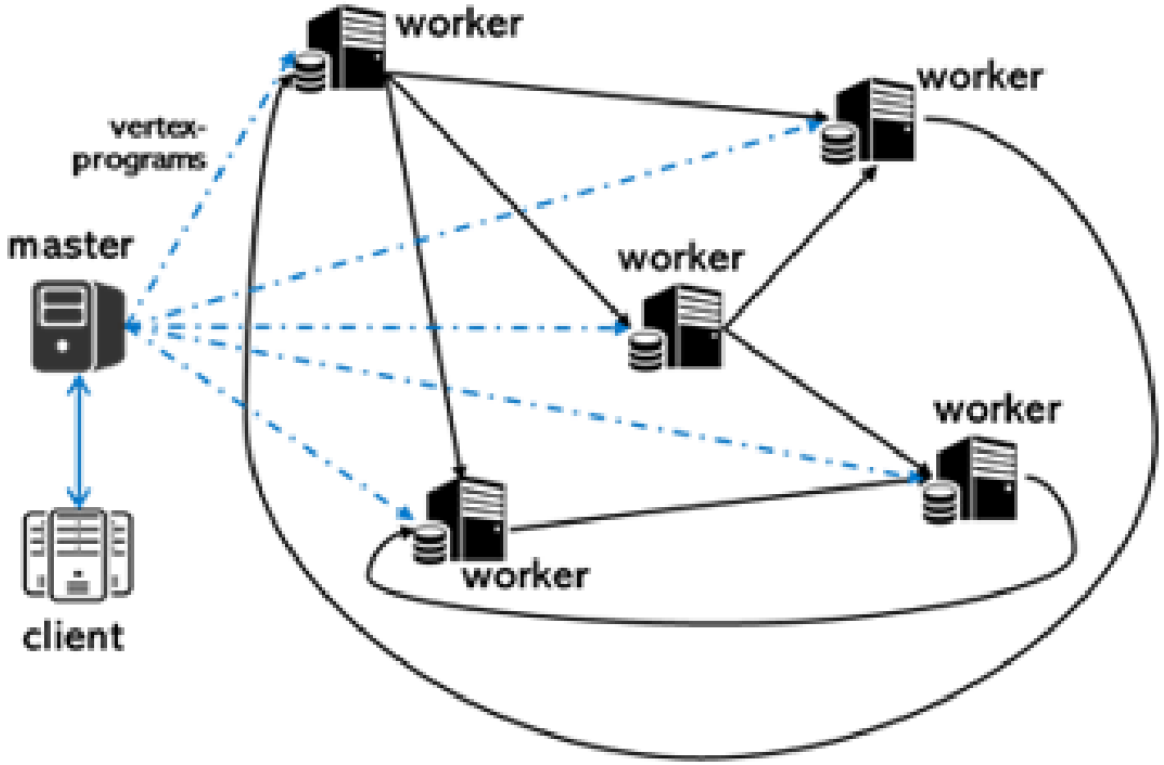




一、分布式深度学习综述[Meng Wang et al., 2020, A Survey on Large-scale Machine Learning]

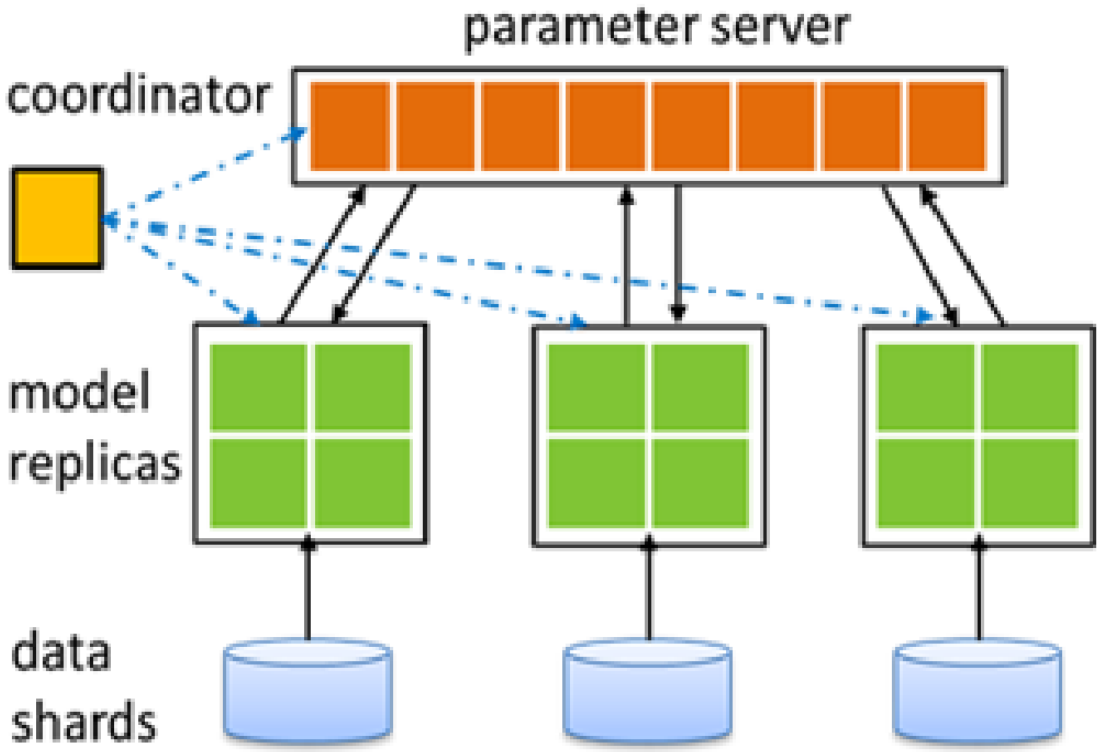


(a). MapReduce

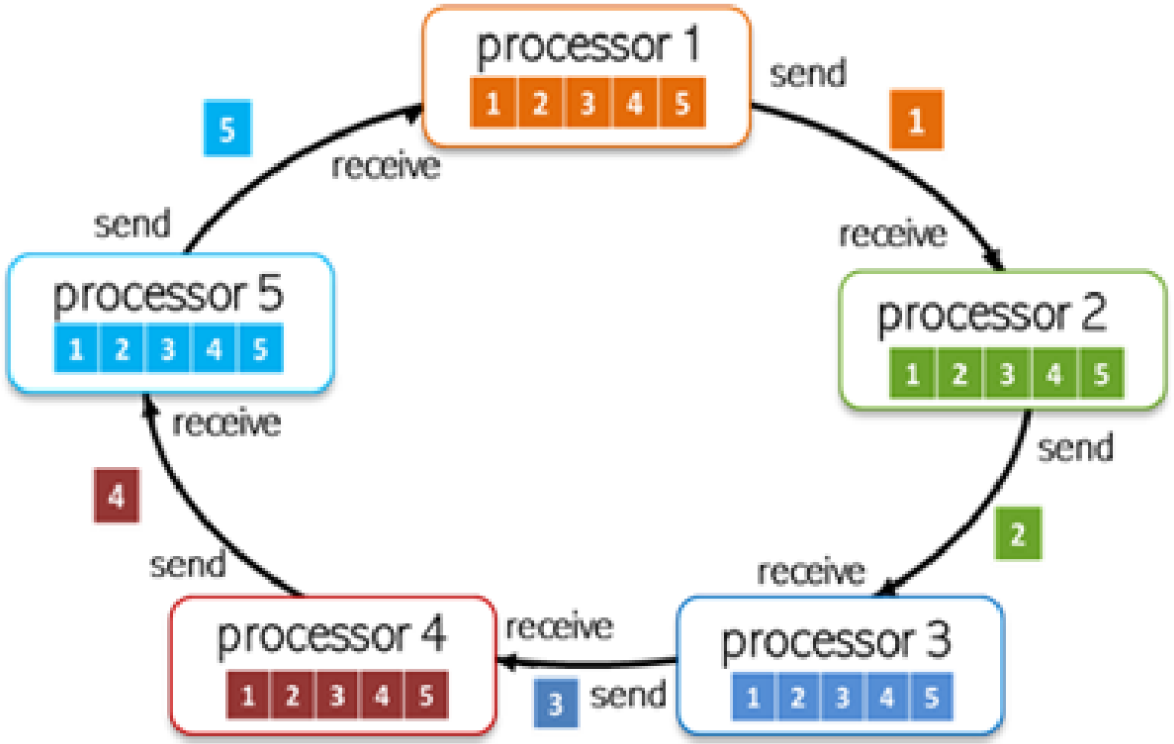


(b). Graph-Parallel

一、分布式深度学习综述[Meng Wang et al., 2020, A Survey on Large-scale Machine Learning]



(c). Parameter Server



(d). AllReduce

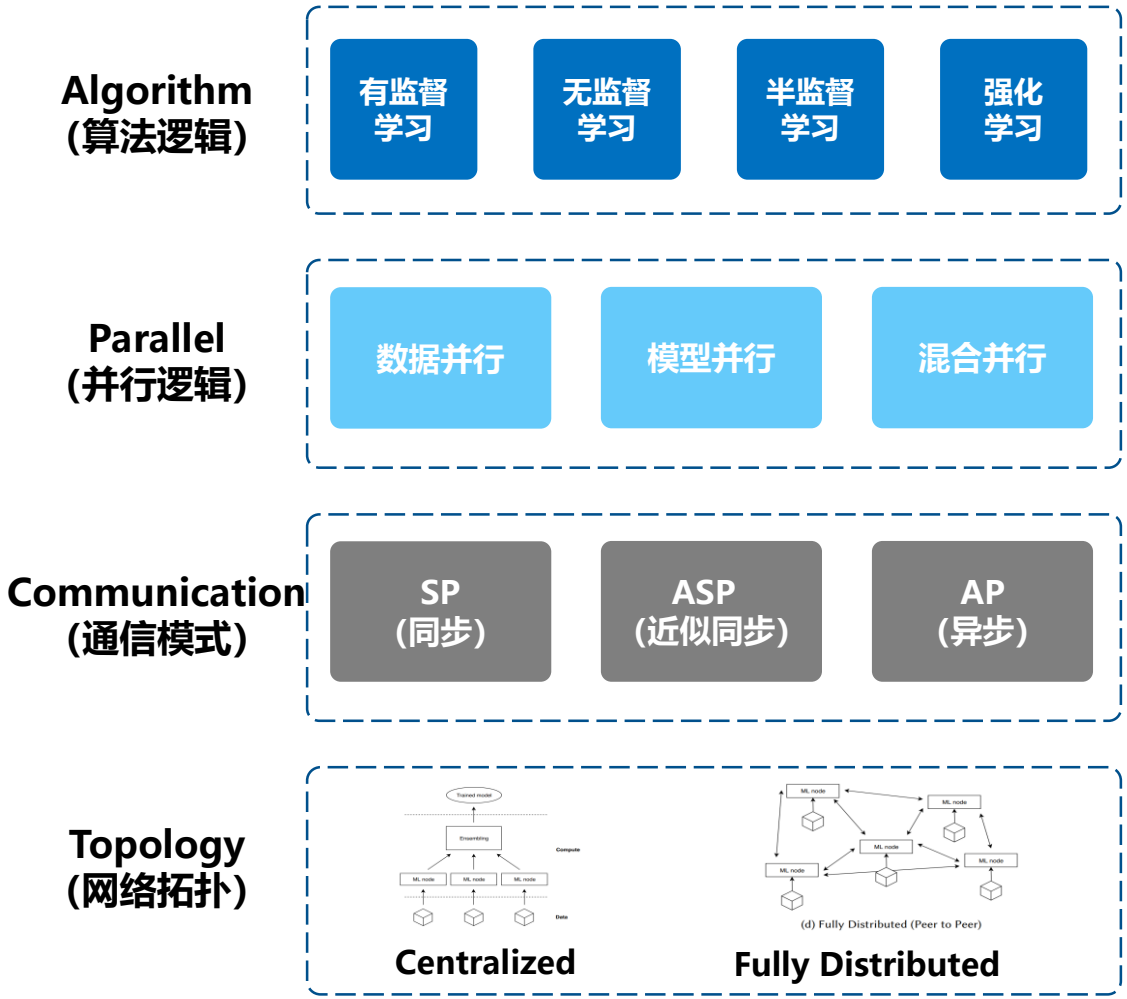
Computation Parallelism扩展研究方向：

- ✓ **灵活的硬件抽象(Flexible hardware abstractions)：**AllReduce能够针对很多算法实现通信开销最小化，但是它假设集群中每个节点能力是等价的，实际场景不同节点往往是有差异的，整体的利用率取决于能力最小的节点，同时在真实场景下甚至有可能是异构的硬件，因此研究基于更加灵活的硬件资源抽象以适配Ring-AllReduce架构非常重要，如通过任务执行的可预测性来平衡不同计算资源
- ✓ **模块化开源软件：**模块化开源软件有利于降低多机分布式场景下的系统复杂度，同时吸引更多开发者参与开源软件开发

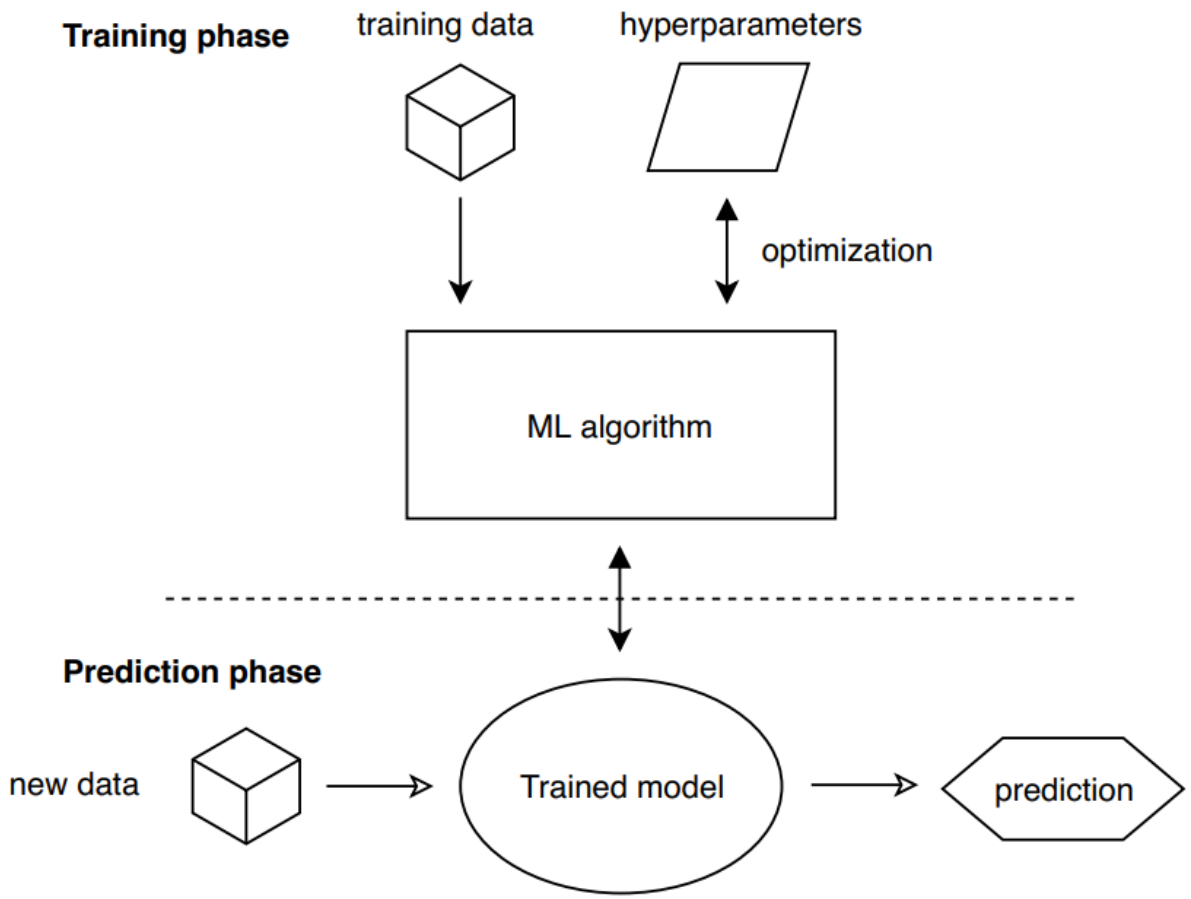
开放性问题：

- ✓ **更严格的复杂度边界分析：**计算复杂度边界决定了计算设备和内存空间，而现有方法更多是基于训练数据而不是真实世界数据的分析
- ✓ **更有价值的数据：**高效的数据标注工具、数据增强技术
- ✓ 更专业化的硬件、量子计算机、隐私保护

一、分布式深度学习综述[JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learning]

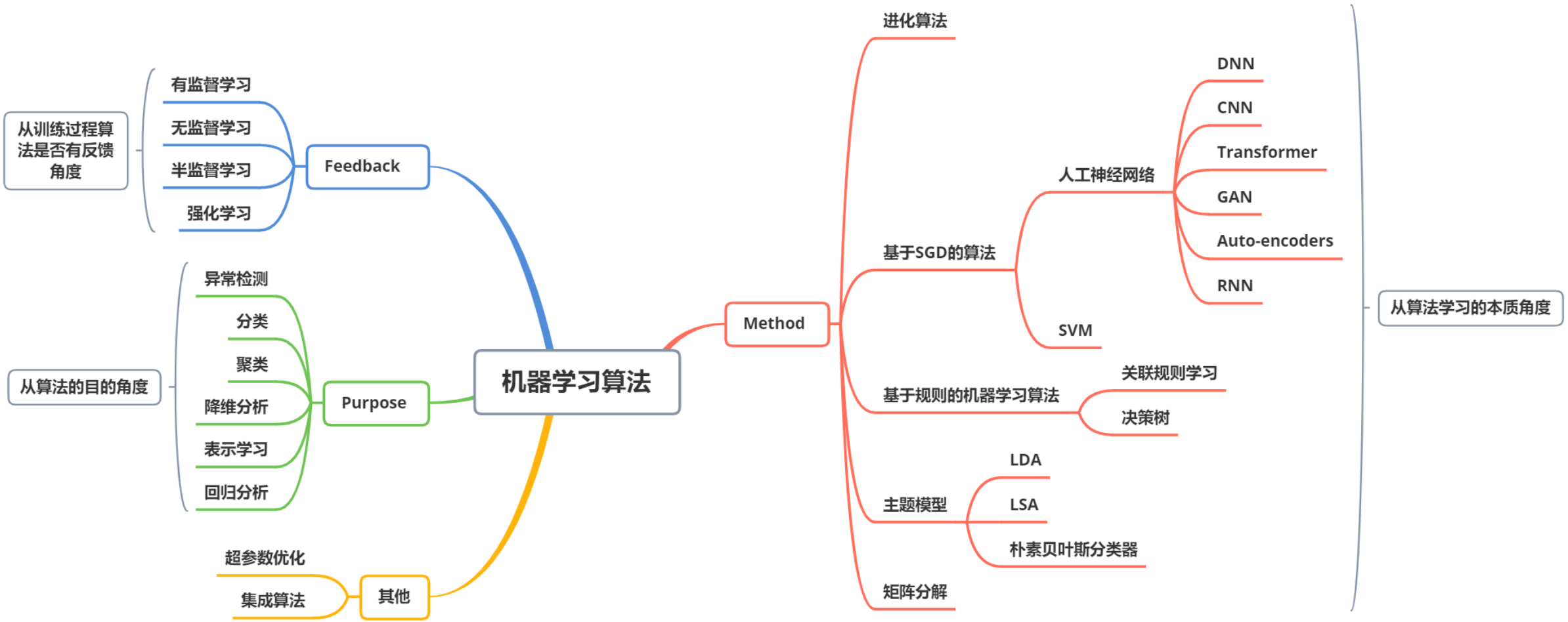


分布式机器学习四个维度整体架构



通用机器学习流程

一、分布式深度学习综述[JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learn



一、分布式深度学习综述[JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learning]

Communication（通信模式）：

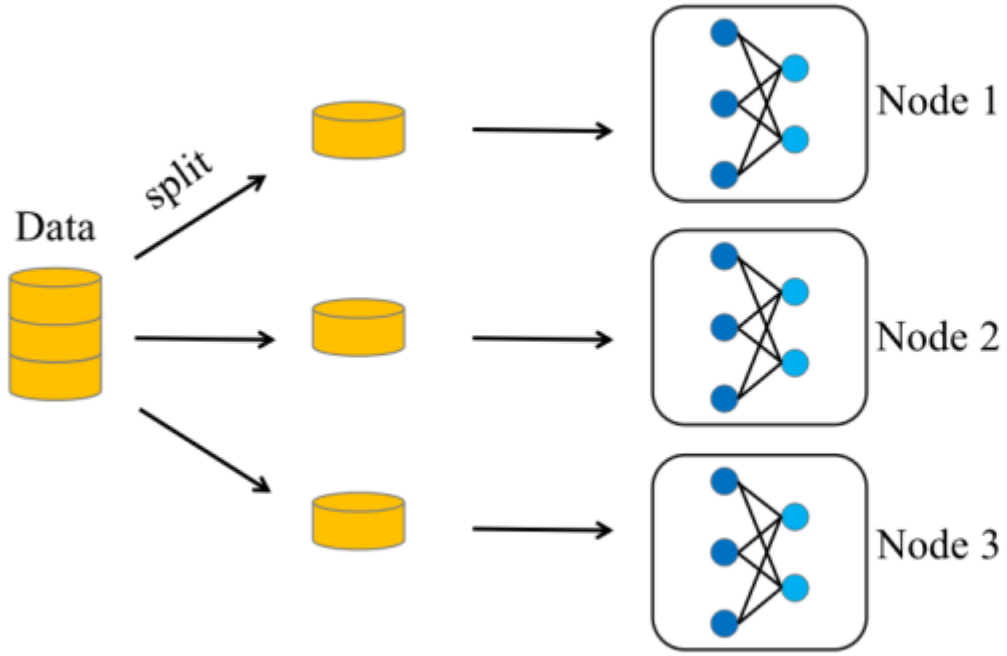
- **Bulk Synchronous Parallel (BSP)**
  - ✓ 严格一致的同步通信模式
  - ✓ 优点：严格保证分布式输出的正确性
  - ✓ 缺点：所有节点需要等待最慢的节点执行完毕才能开始下一个step

- **Stale Synchronous Parallel (SSP)**
  - ✓ 允许最快的worker与最慢的worker之间相差确定的迭代步数，超出阈值则暂停训练
  - ✓ 优点：效率提升同时仍然可以比较严格的保证模型收敛性
  - ✓ 缺点：一些节点总是比较慢是就会影响模型收敛

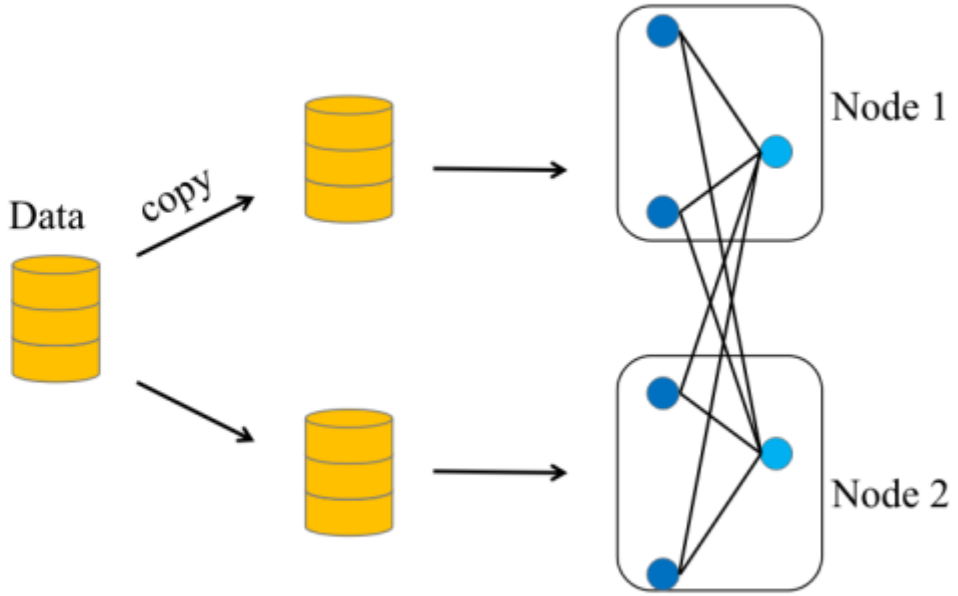
- **Approximate Synchronous Parallel (ASP)**
  - ✓ 限制参数聚合更新的不正确性，设置参数的更新落后时间阈值
  - ✓ 优点：保证了每次迭代更新的有效性
  - ✓ 缺点：如何选择正确有效的梯度是很难的

- **Barrierless Asynchronous Parallel/ Total Asynchronous Parallel(BAP/TAP)**
  - ✓ Worker之间不在相互等待
  - ✓ 优点：能够实现最大程度的训练加速
  - ✓ 缺点：模型可能收敛会更慢甚至由于一些错误梯度的影响而不收敛

一、分布式深度学习综述[JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learning]



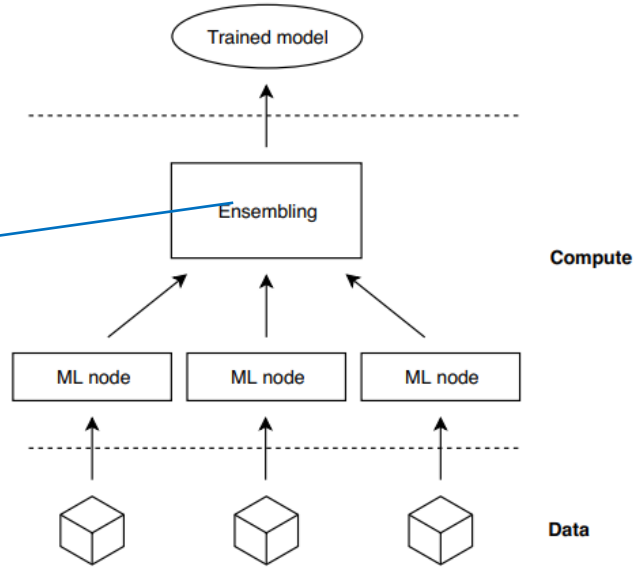
(a) Data Parallelism



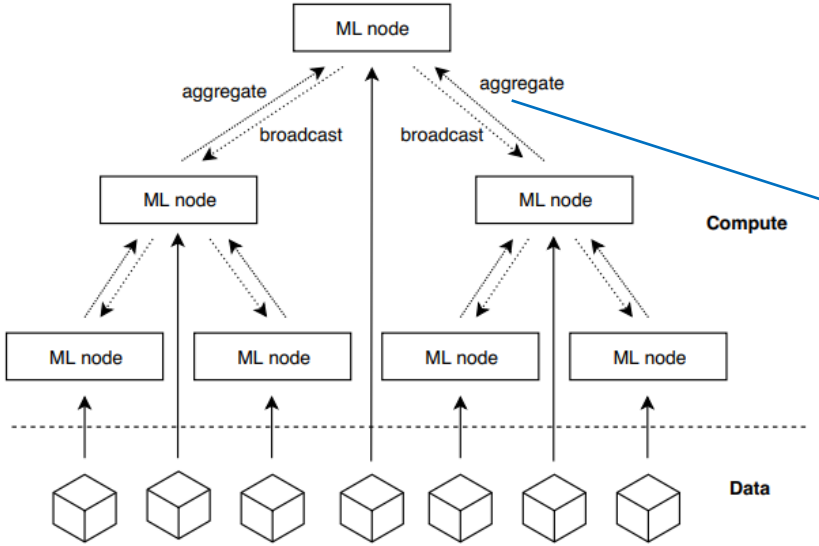
(b) Model Parallelism

一、分布式深度学习综述[JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learning]

在确定的中心节点聚合



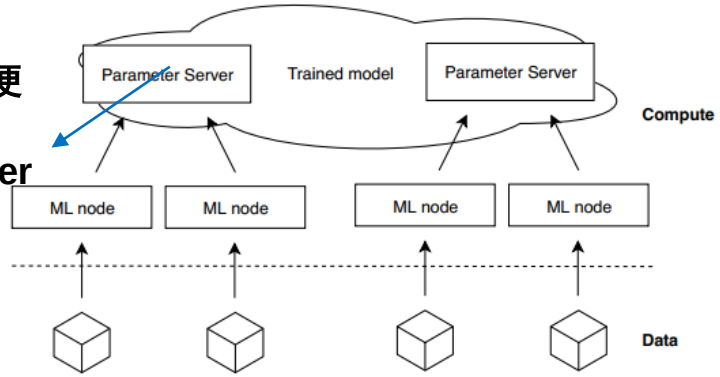
(a) Centralized (Ensembling)



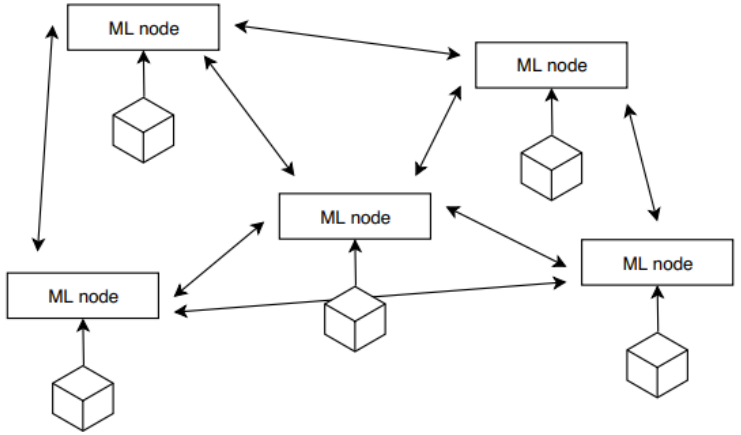
(b) Decentralized (Tree)

- ✓ 多层级树形拓扑，最后再根节点聚合
- ✓ 易于扩展，每个节点只需要与父节点和子节点建立联系

- ✓ 模型参数在PS中聚合，便于模型全局check
- ✓ PS需要处理对所有worker的通信，会成为瓶颈



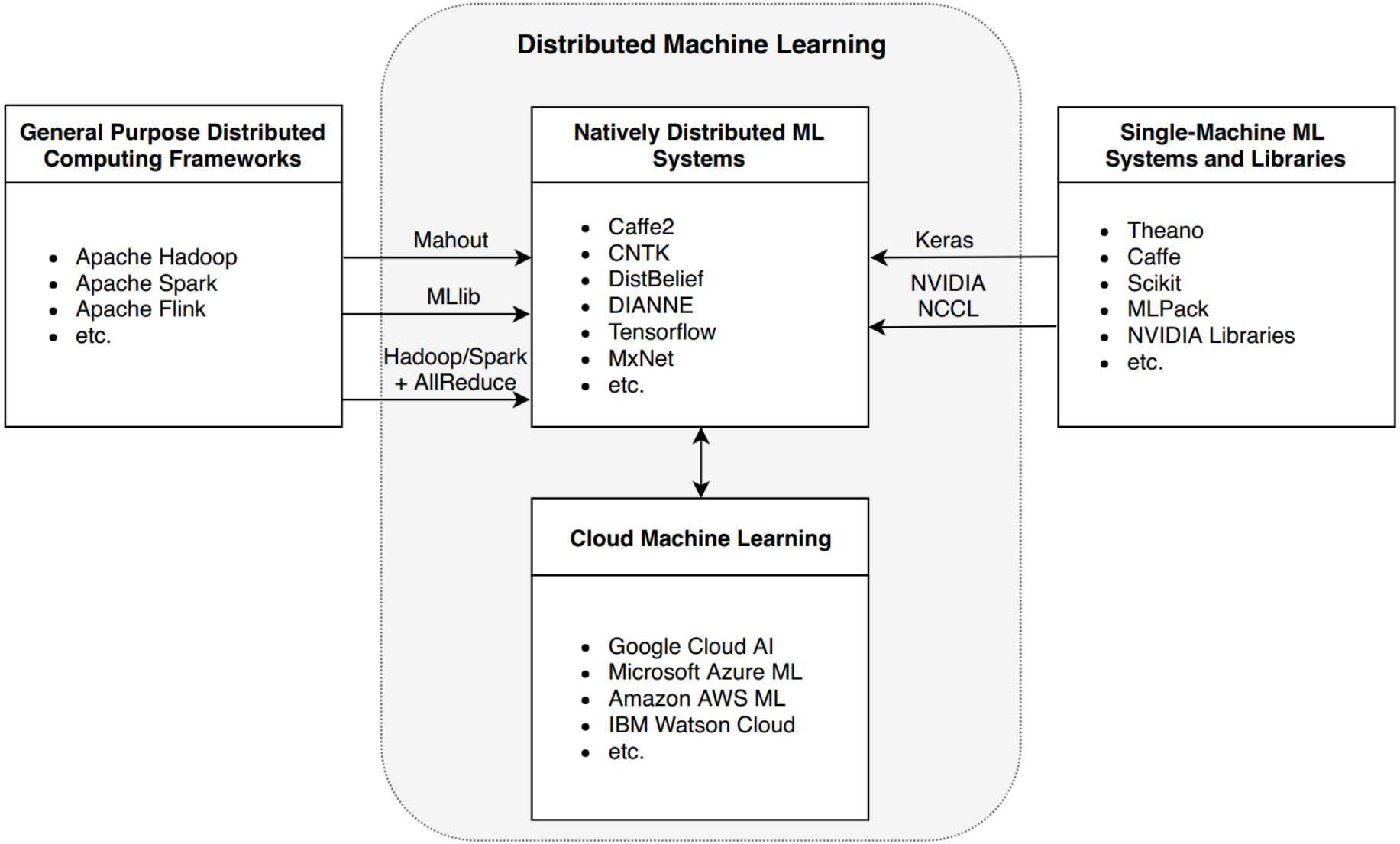
(c) Decentralized (Parameter Server)



(d) Fully Distributed (Peer to Peer)

- ✓ 各节点独立的完全分布式结构
- ✓ 具有更高的可伸缩性和单点故障恢复能力

一、分布式深度学习综述[JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learn

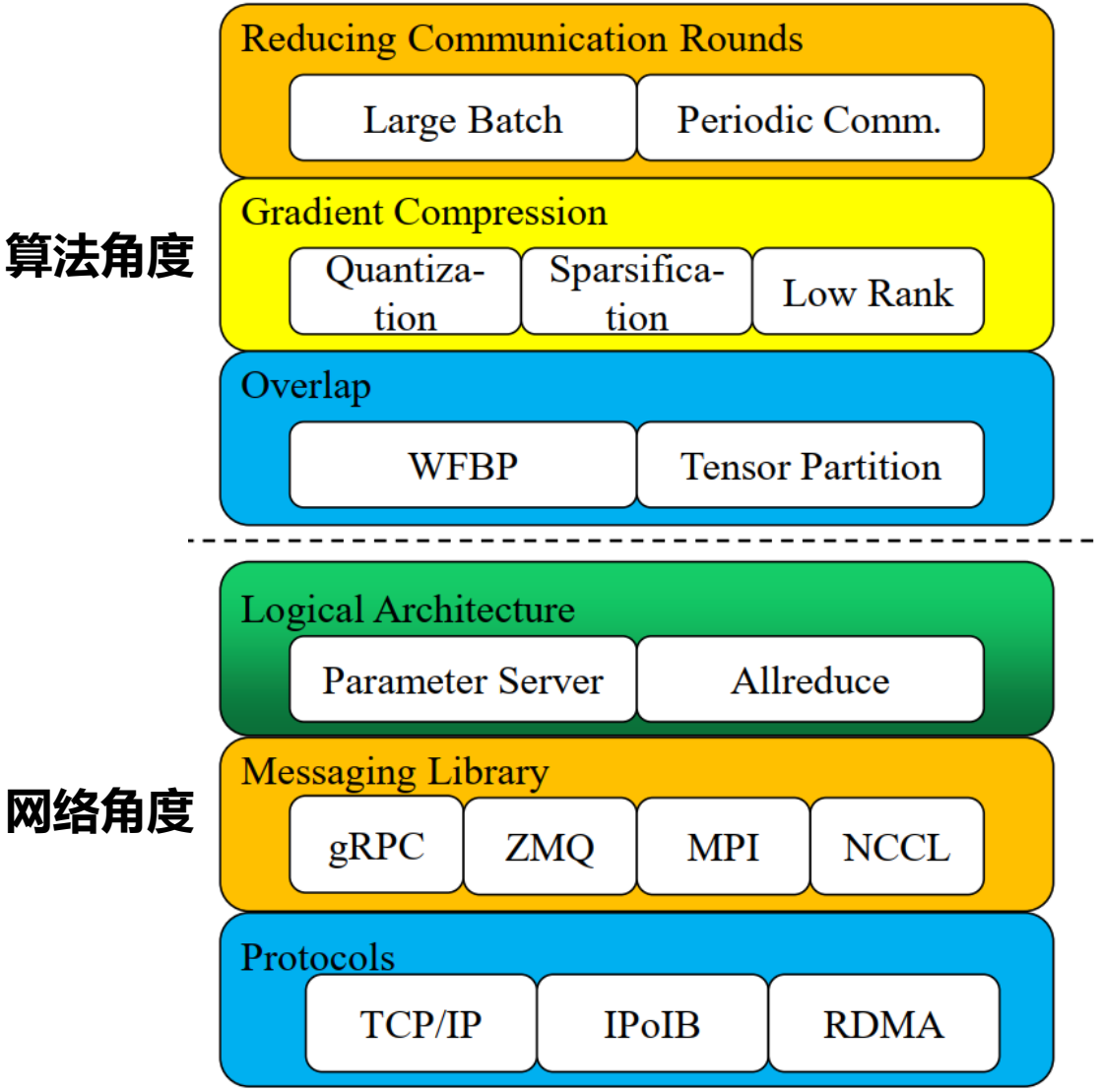


分布式机器学习生态

一、分布式深度学习综述[JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learning]

DDLS Name (a-z)	*Parallelism* (Model / Data)	*Optimization*	*Scheduling*	*Parameter Exchange*	*Topology*	*Remarks*
BigDL[27]	DP only	central	sync.	scatter-red.	distributed PS (always $k = n$)	Each worker acts as a parameter server for $\frac{1}{n}$ of the model (cf. Section 3.4.1). Distributed parameter exchanges are realized via the Spark block manager.
CaffeOnSpark[24]	DP only	central	sync.	scatter-red.	distributed PS (always $k = n$)	Parameter exchange realized via RDMA using repeated invocations of MPI functions. Equivalent implementations are available for **Caffe2** and **Chainer**.
COTS HPC[9]	MP only	central	sync.	–	distrib. array abstraction	Model layers partitioned along tensor dimensions and distributed across cluster. Fine-grained access is managed via a low-level array abstraction.
D-PSGD[26]	DP only	decentral	sync.	2:1 reduce	closed ring	Each node exchanges parameters with only its neighbors on the ring (cf. Section 3.4.2).
DistBelief[21]	MP + DP	central	async.	ad hoc	distrib. PS	Model partitions spread across dedicated parameter server nodes (cf. Section 3.4.1).
EASGD[30]	DP only	decentral	async.	ad hoc	single master	Decentralized asynchronous system as discussed in Section 3.3.5. Reactive adjustment of hyper-parameters can speedup training [42].
FireCaffe[15]	DP only	central	sync.	binom. tree	single PS	Simplistic centralized synchronous system as discussed in Section 3.3.1.
GoSGD[45]	DP only	decentral	soft-bounded async.	ad hoc	p2p mesh	No dedicated master node. Parameter exchanges between any two workers realized via sum-weighted randomized gossip protocol as discussed in Section 3.4.2.
MPCA-SGD[11]	DP only	decentral	soft-bounded async.	binom. tree	dedicated master node	Model updating and sharing updates are decoupled. Penalization occurs as a part of the model's cost function. Staleness effects are dampened using an extrapolation mechanism.
MXNet[19]	MP + DP	central	bounded async.	scatter-reduce *async.*: ad hoc	distributed PS (default $k = n$)	Supports various advanced parameter server configurations, including but not limited to hierarchical multi-stage proxy servers (cf. Section 3.4.1).
Parameter Server[14]	MP + DP	central	bounded async.	reduce *async.*: ad hoc	distrib. PS	Model partitions spread redundantly across parameter server group. Workers organized in model parallelism enabled groups. One worker per group can act as a proxy server.
Petuum[16]	MP + DP	central	bounded async.	ad hoc with eager scatter	distrib. PS	Pioneered the use of delay bounds to control staleness (cf. Section 3.3.4). Average model staleness is further reduced through the eager distribution of model parameters.
Project Adam[23]	MP + DP	central	async.	ad hoc	distrib. PS	Dedicated parameter server group that is managed as a Paxos cluster. Hybrid parallelism realized through transferring gradient computation for fully connected layers into PS.
PyTorch[47]	MP + DP	central	sync.	all-reduce	single PS or replicated PS	Model parallelism capabilities were added recently with version 1.4.0. Can only use either synchronous data-parallelism or model parallelism.
SparkNet[10]	DP only	decentral	sync.	reduce	dedicated master node	Decentralized synchronous implementation as discussed in Section 3.3.2. Realized using Spark map-reduce. Production-grade re-implementation present in **deeplearning4j**.
TensorFlow[17]	MP + DP	central	bounded async.	scatter/all-red. *async.*: ad hoc	distributed PS (default $k = n$)	Supports single and multi parameter server setups, as well as all-reduce-based approaches. By default, each worker acts as a parameter server for a portion of the model.
TreeEASGD[25]	DP only	decentral	bounded async.	ad hoc	tree	All nodes are workers and form a tree. Each worker only exchanges parameters with its immediate up- and downstream neighbors (cf. Section 3.4.2).

一、分布式深度学习综述[Shuo Ouyang et al., 2020, Communication optimization strategy...A survey]



分布式深度学习中的通信优化策略概述

现有研究工作：

- ✓ 算法角度：讨论如何降低通信轮数、梯度压缩、通信计算重叠
- ✓ 网络角度：讨论逻辑通信架构、集合通信库、通信协议

可能研究方向：

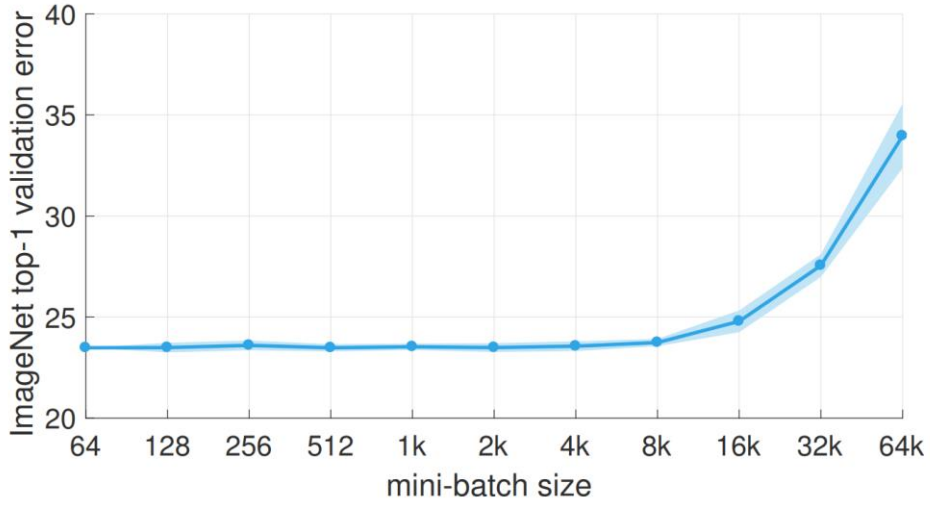
- ◘ 由于不同类型算法差异性，需要关注更多类型的深度学习分布式训练集优化
- ◘ 非凸优化问题的局部SGD训练方法
- ◘ 模型精度与梯度压缩比之间的平衡如何选择，目前研究更多倾向减少错误梯度反馈对模型收敛的影响
- ◘ 更高的通信计算重叠比例，现有的计算通信调度问题更多采用启发式算法，并不一定得到最优解
- ◘ 不同数据中心拓扑之上的大规模DNN模型训练
- ◘ 通信开销与分布式训练性能评估工具缺乏，需要更好的分析性能评估模型和工具用于分布式训练的瓶颈和问题分析

一、分布式深度学习综述[Shuo Ouyang et al., 2020, Communication optimization strategy...A survey]

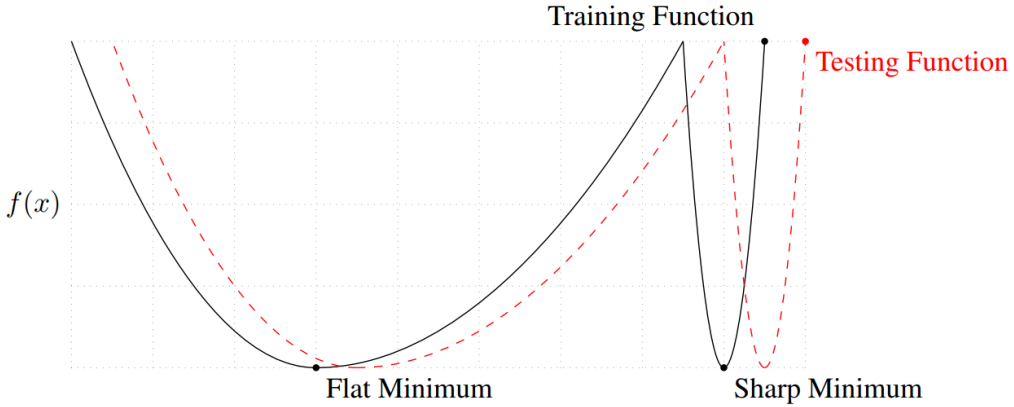
1.减少通信轮数——增大Batch Size、采用Model Averaging

Table 1: Compare ResNet-50 Training Detail with Different Wroks

Works	Batch Size	Hardware	Top-1 Accuracy	Training Time
He et al.[9]	256	Tesla P100 × 8	75.3% (baseline)	29h
Goyal et al.[30]	8k	Tesla P100 × 256	76.3%	1h
Cho et al.[32]	8k	Tesla P100 × 256	75.0%	50min
Smith et al.[33]	8k→16k	TPU (256 tensorcores)	76.1%	45min
Codreamu et al.[34]	32k	KNL × 1024	75.3%	42min
You et al.[35]	32k	KNL × 2048	75.4%	20min
Akiba et al.[36]	32k	Tesla P100 × 1024	74.9%	15min
Jia et al.[37]	64k	Tesla P40 × 1024	76.2%	8.7min
Ying et al.[38]	32k	TPU × 1024	76.3%	2.2min
Mikami et al.[39]	54k	Tesla V100 × 3456	75.29%	2.0mins
Yamazaki et al.[40]	80k	Tesla V100 × 2048	75.08%	1.2mins



- ✓ 每次迭代需要传输的梯度数据通信量与batch size无关，增加batch size减少了迭代次数，因此减少了通信轮数
- ✓ 大的batch size可能导致精度下降，因为模型倾向收敛到一些尖锐的极小值区域，导致其在测试集上的泛化性能降低
- ✓ 采用warm-up学习率调整方案、layerwise adaptive rate scaling等可以比较好的缓解这个问题

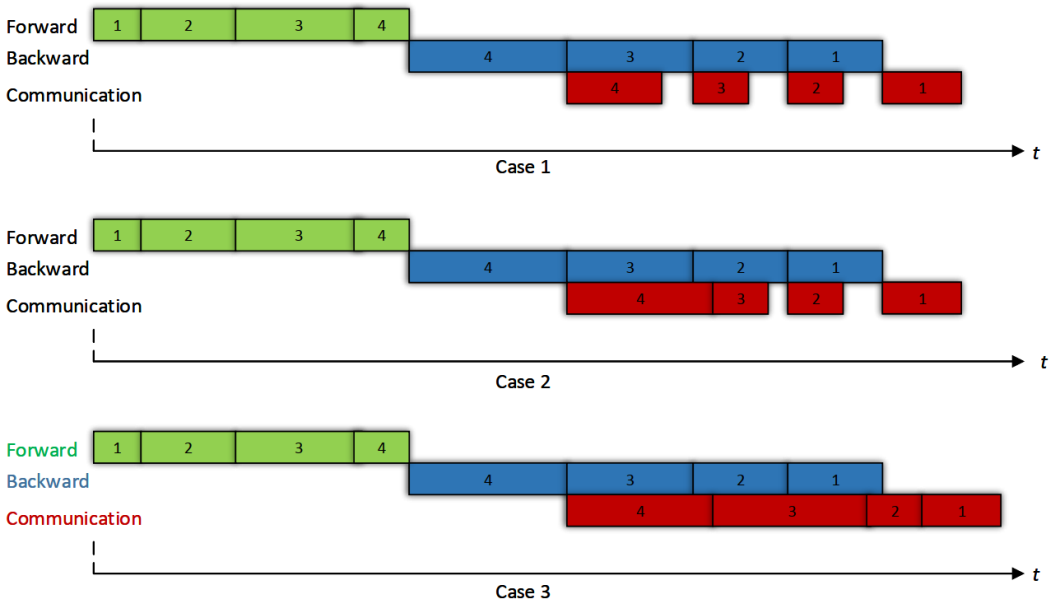


2.梯度压缩（Gradients Compression）

- ✓ 通常情况下，分布式训练时需要传输的梯度和模型参数采用32bit变量，以BERT-340M模型为例，每次迭代有1.2G数据传输需求
- ✓ **量化（Quantization）**：采用更低精度去表示梯度，如one-bit SGD、sign SGD、QSGD
- ✓ **稀疏化（Sparsification）**：传输更加重要的梯度变量，避免没有意义的不必要开销，Deep gradient compression (DGC)方法能够将ResNet-50的梯度在没有精度损失的情况下从97 MB降低到0.35 MB，大大提高通信效率
- ✓ **矩阵分解（Matrix Decomposition）**：将一些很大的梯度矩阵分解成多个小矩阵，通过传输更小的矩阵降低通信开销，如ATOMO、PowerSGD。

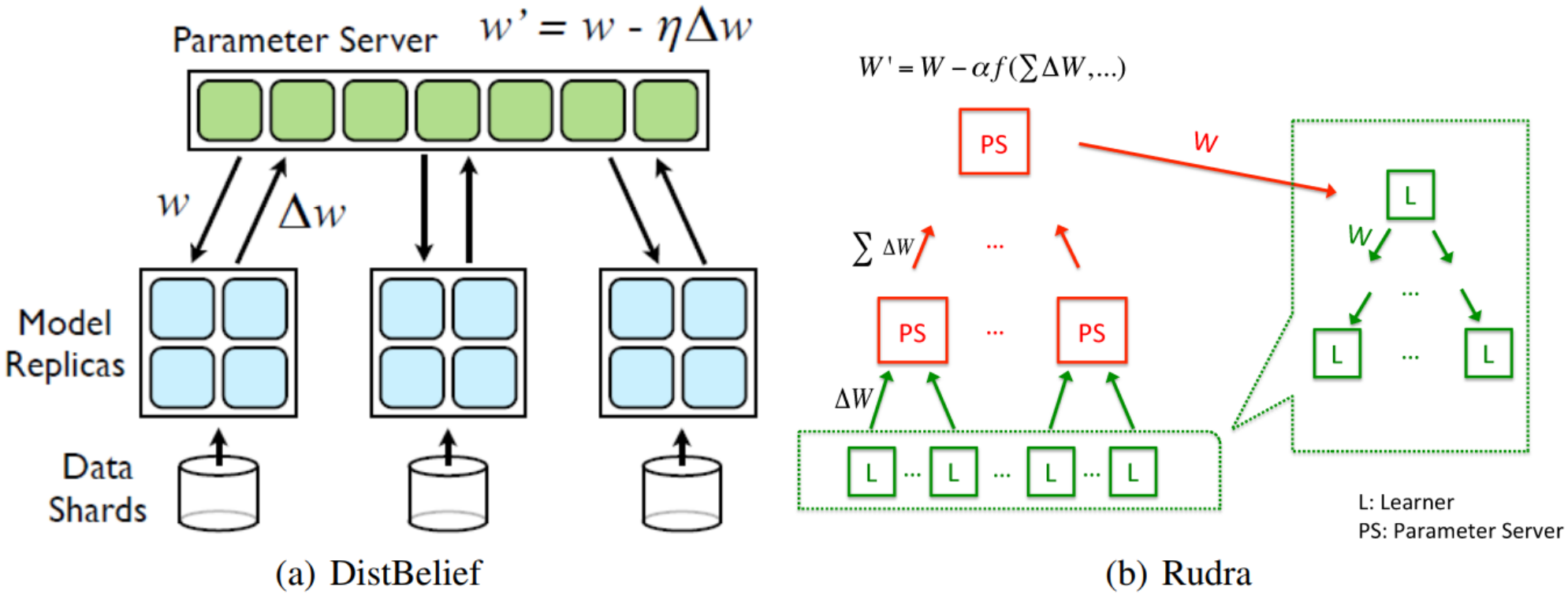
3.计算通信重叠（Computation-Communication Overlap）

- ✓ 模型第L层的梯度传输可以与L-1层的梯度计算在时间轴上重叠
- ✓ 看似降低了通信开销，但实际上并能绝对的解决通信瓶颈，还需要同其他优化方法一起使用
- ✓ wait-free backward propagation (WFBP)：每一层计算结束后开始通信传输
- ✓ merged-gradients WFBP：合并一些小的通信信息



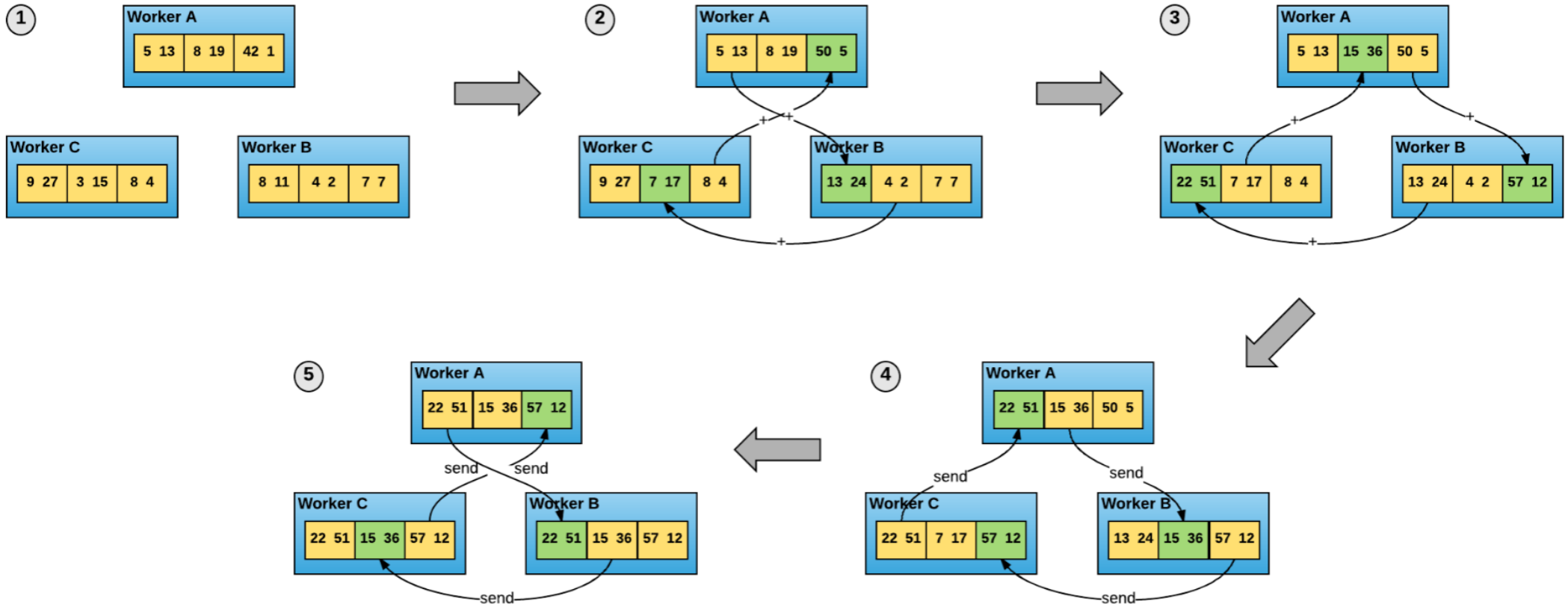
以多机数据并行为例的计算通信重叠示例

4. 逻辑架构（Logical Architectures）



Parameter Server架构

4. 逻辑架构（Logical Architectures）



AllReduce-sum 示例

5. 消息通信库（Messaging Libraries）

□ 支持Parameter Server架构

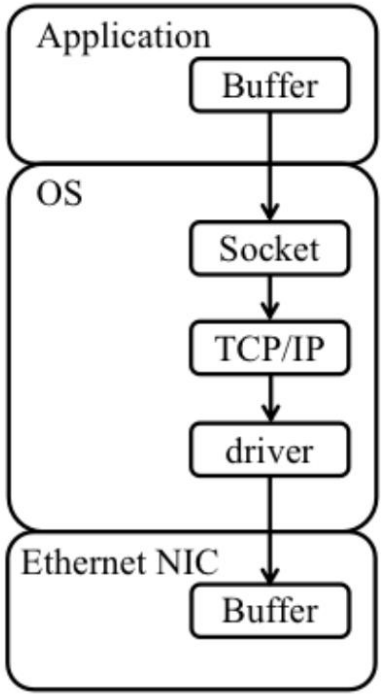
- ✓ **ZeroMQ**：一种介于介于应用层和传输层之间的基于消息队列的多线程通信库
- ✓ **gRPC**：一种使得应用程序之间可以进行通信的调用机制，远程过程调用

□ 支持AllReduce算法

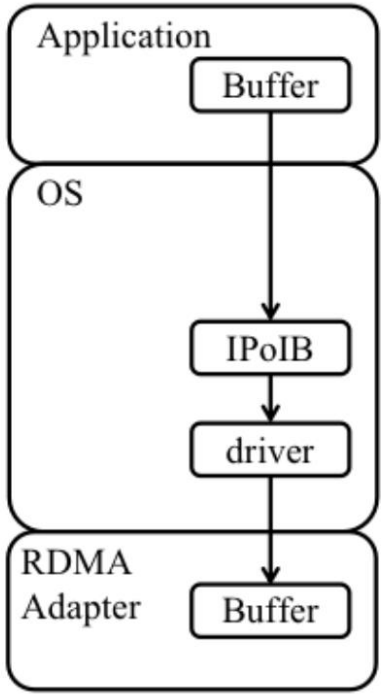
- ✓ **MPI**：一种消息传递编程模型，并成为这种编程模型的代表和事实上的标准。非常多基于MPI的优化版（Horovod、MXNet-MPI、TensorFlow-MPI）
- ✓ **Gloo**：facebook开源的一套集体通信库，提供了对机器学习中有用的一些集合通信算法如：barrier, broadcast, allreduce。
- ✓ **NCCL**：英伟达基于NCIDIA-GPU的一套开源的集体通信库，基于自身硬件定制的，能做到更有针对性且更方便优化
- ✓ **Baidu AllReduce**：百度首先提出的一种Ring AllReduce的实现
- ✓ **Aluminum**：一个开源通信库，它以MPI和NCCL等为后端，提供了泛化的通信API。相比于MPI和NCCL，Aluminum更像是一个接口层，只需要简单的改动，它就能跑在不同的硬件上。
- ✓ **BlueConnect**：通过将一个AllReduce操作分解成一系列Reduce-Scatter和Allgather操作来优化降低通信开销

6. 网络协议（Network Protocols）

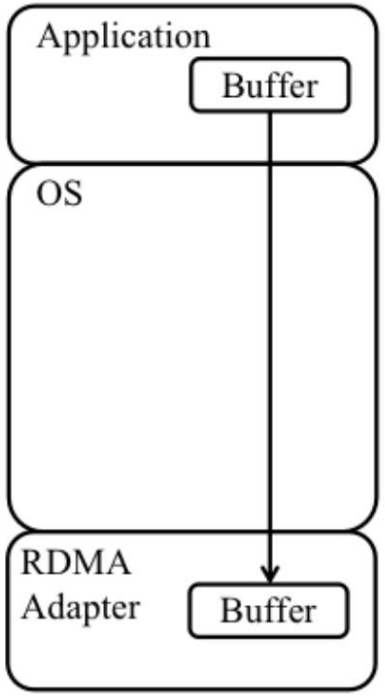
- ✓ 传统消息通信多基于TCP/IP协议，需要操作系统建立Socket来建立通信连接完成数据传输，增加了通信延时
- ✓ **RDMA**：允许机器直接读写另一个机器的内存而不需要通过操作系统层，有两种模式：消息模式和内存模式
- ✓ **IP over IB**：将IP数据包封装到InfiniBand卡，使得TCP/IP协议可以直接在IB的上层去跑，而不需要任何的代码修改，但是不能避开主机OS
- ✓ 利用RDMA对GPU进行直连，允许RDMA可以直接访问GPU内存而不是通过主机内存，来提高通信吞吐量



(a) TCP/IP Socket



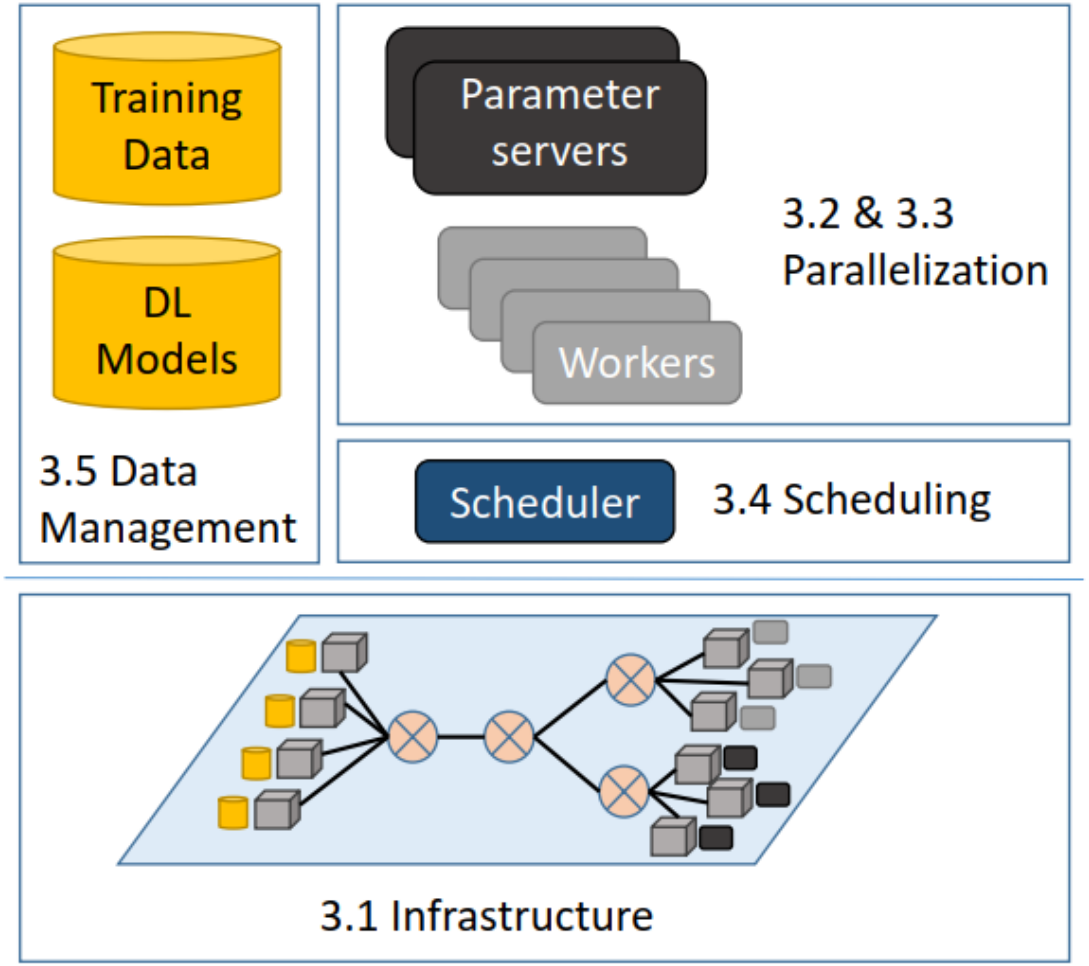
(b) IPoIB



(c) RDMA

一、分布式深度学习综述

[RUBEN MAYER et al., 2019, Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools]



分布式深度学习框架

与其他工作所不同的是：

- ✓ 把不同训练任务的调度问题也作为了scalable分布式深度学习一个重要问题。
- ✓ 将训练数据、深度学习模型的管理也作为重要研究内容，包括了数据标注、数据预处理、海量数据存储，以及模型的跟踪、存储、索引和共享、查询、分析。

一、分布式深度学习综述

- Meng Wang et al., 2020, A Survey on Large-scale Machine Learning
- JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learning
- Shuo Ouyang et al., 2020, Communication optimization strategies for distributed deep neural network training: A survey
- RUBEN MAYER et al., 2019, Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools

二、自动并行技术及已有框架

- 自动并行问题定义
- MindSpore自动并行
- 自动并行框架FlexFlow
- 分布式训练框架Whale

三、优化思考

- 并行搜索空间建模
- 边训练边搜索模式
- 自感知自优化配置

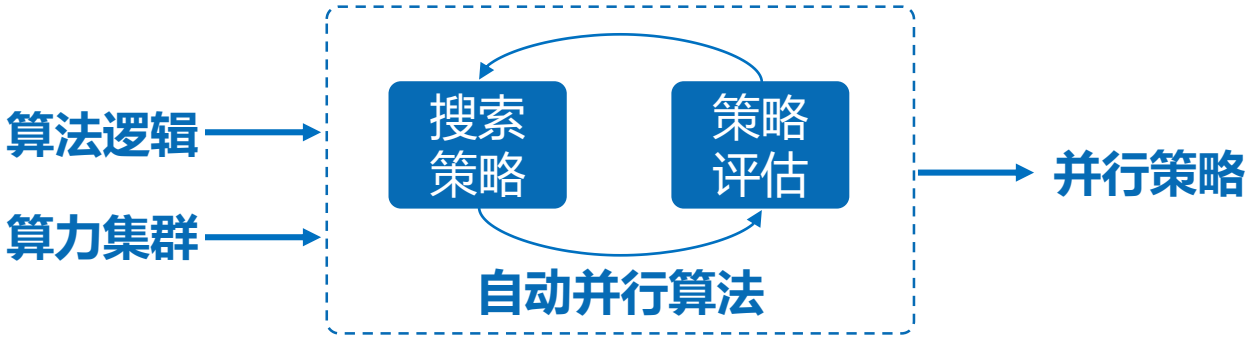
二、自动并行技术及已有框架——问题定义

分布式深度学习中自动并行问题：

- ✓ 面向算法开发者，如何将深度学习算法逻辑自动化并行到算力集群进行更高效率训练和推理的过程。这个自动化并行过程对于算法开发者几乎是无感知的。
- ✓ 输入：算法逻辑、算力集群
- ✓ 输出：并行策略

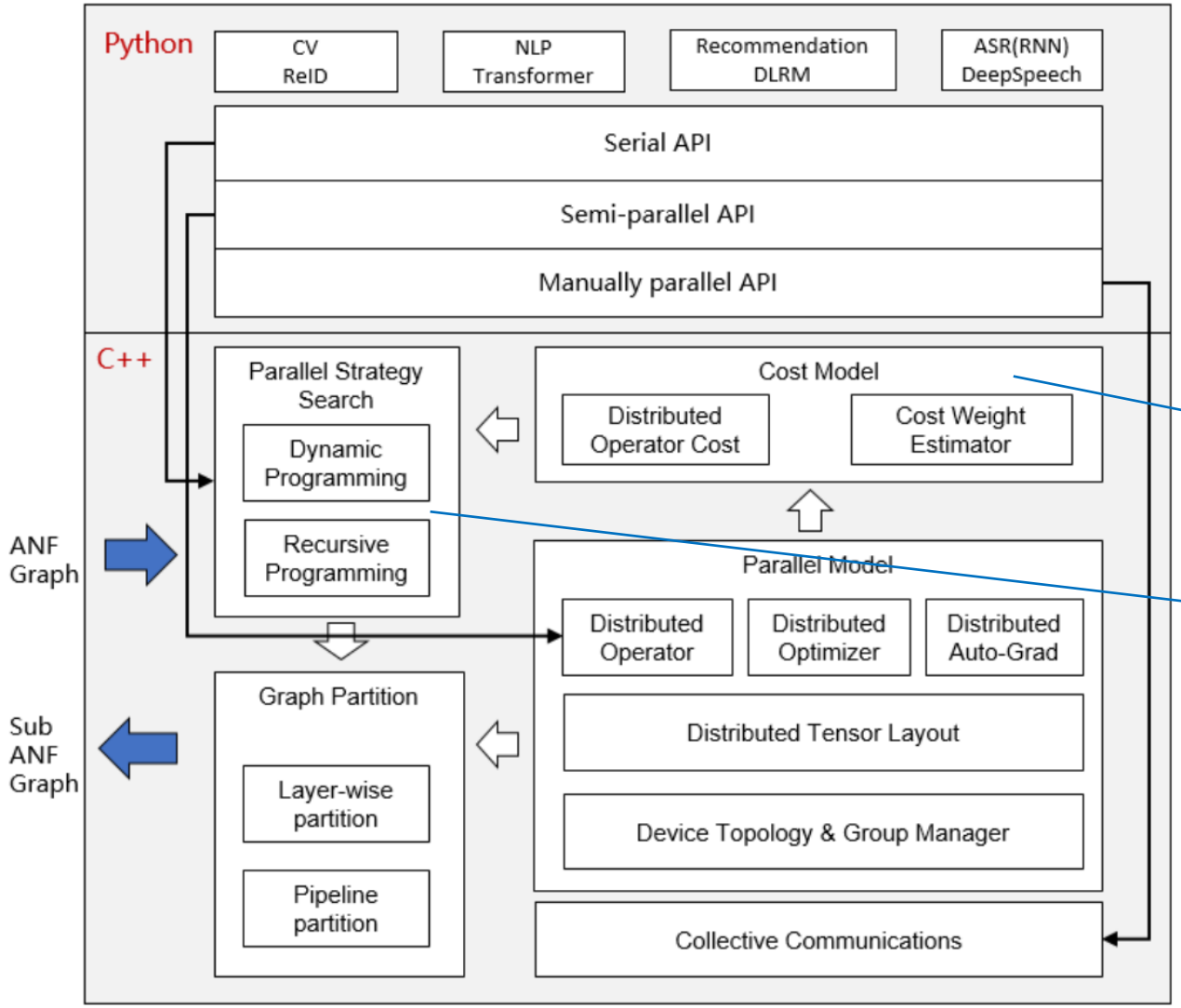
需要解决的问题：

- ✓ 算法逻辑：如何表示？以什么样的粒度表示？与上层框架的表示层如何对接，还是需要重新设计框架？算法逻辑可并行程度如何？如何评估切分的有效性？
- ✓ 算力集群：如何表示？如何建模计算、通信、内存、拓扑等资源？
- ✓ 并行策略：如何表示？如何与底层runtime适配？



常见自动并行逻辑

二、自动并行技术及已有框架——MindSpore



- ✓ MindSpore具备手动、半自动、全自动并行能力
- ✓ 采用动态规划算法进行并行策略搜索
- ✓ Cost Model建模相对简单

自动并行Cost Model（如何评估或计算一个计算图的并行执行性能）

基于动态规划的并行策略搜索

二、自动并行技术及已有框架——MindSpore

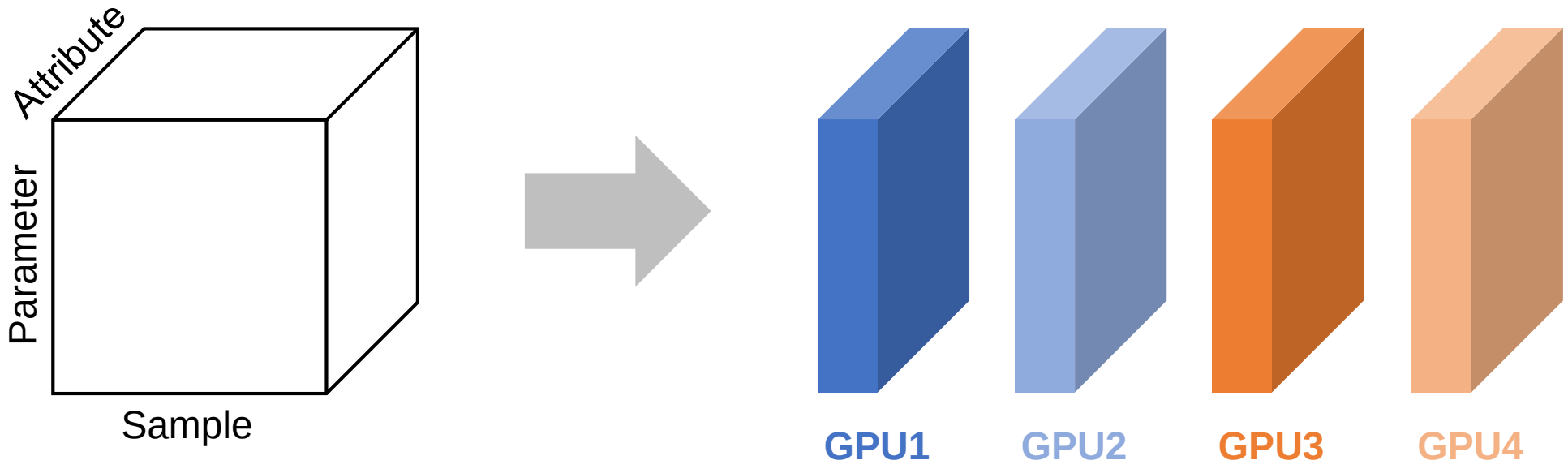
```python
class DenseNet(nn.Cell):
    def __init__(self):
        super(DenseMutMulNet, self).__init__()
        self.embedding_weight = Parameter(Tensor(12288, 128))
        self.embedding = P.MatMul()
        self.fc1 = nn.Dense(128, 768, activation='relu')
        self.fc2 = nn.Dense(128, 768, activation='relu')
        self.fc3 = nn.Dense(128, 768, activation='relu')
        self.transpose = P.Transpose()
        self.matmul1 = P.MatMul()
        self.matmul2 = P.MatMul()

    def construct(self, x):
        x = self.embedding(x, self.embedding_weight)
        q = self.fc1(x)
        k = self.fc2(x)
        v = self.fc3(x)
        k = self.transpose(k, (1, 0))
        c = self.matmul1(q, k)
        s = self.matmul2(c, v)
        return s

def train_step():
    context.set_auto_parallel_context(parallel_mode=ParallelMode.AUTO_PARALLEL)
    input = Tensor(np.ones([32, 128]).astype(np.float32))
    label = Tensor(np.zeros([32, 768]).astype(np.float32))
    net = DenseNet()
    model = Model(net, opt, loss)
    train(net, input, label)
```

单机模型代码

Auto Parallel

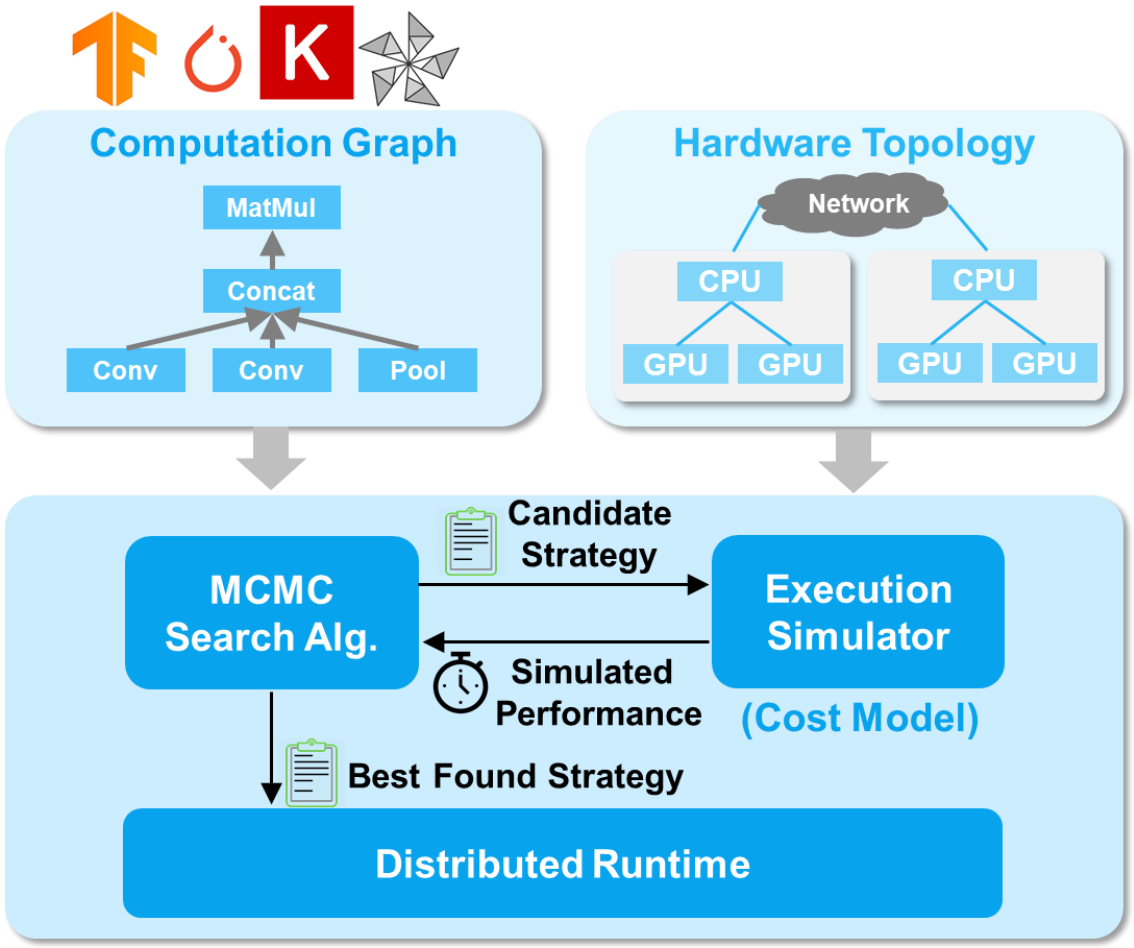
The SOAP Search Space

- **S**amples: partitioning training samples (Data Parallelism)（样本维度切分）
- **O**perators: partitioning ML operators (Model Parallelism) （算子维度切分）
- **A**ttributes: partitioning attributes in a sample (e.g., pixels) （样本属性维度切分）
- **P**arameters: partitioning parameters in an operator（算子的内部参数维度切分）

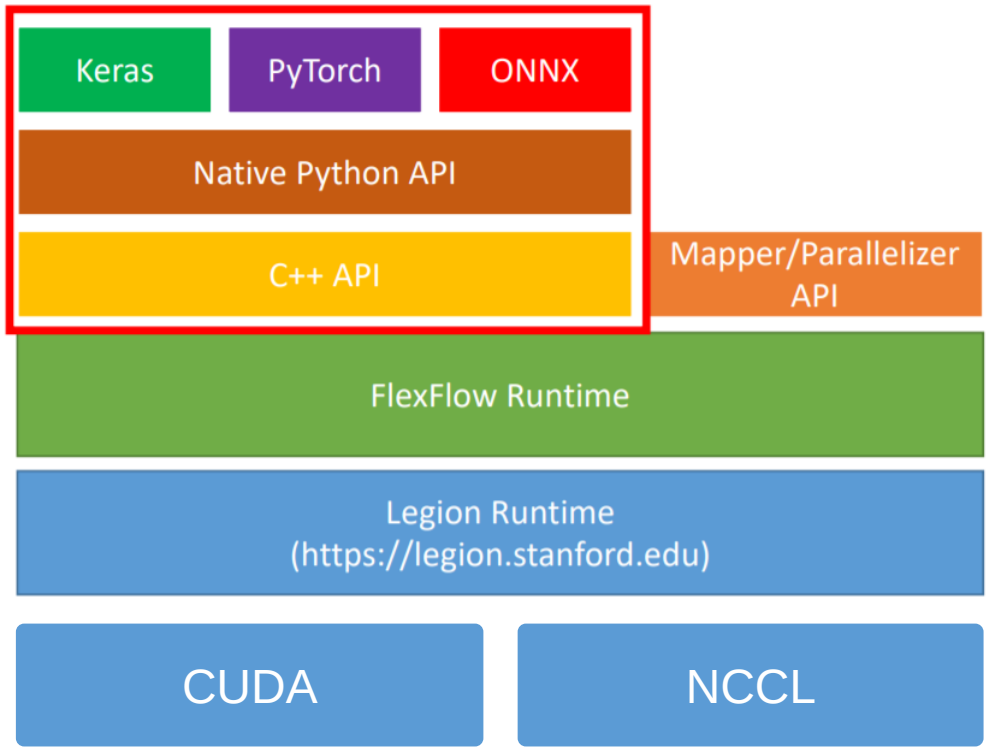


Parallelizing a convolution in ***Sample***

二、自动并行技术及已有框架——FlexFlow

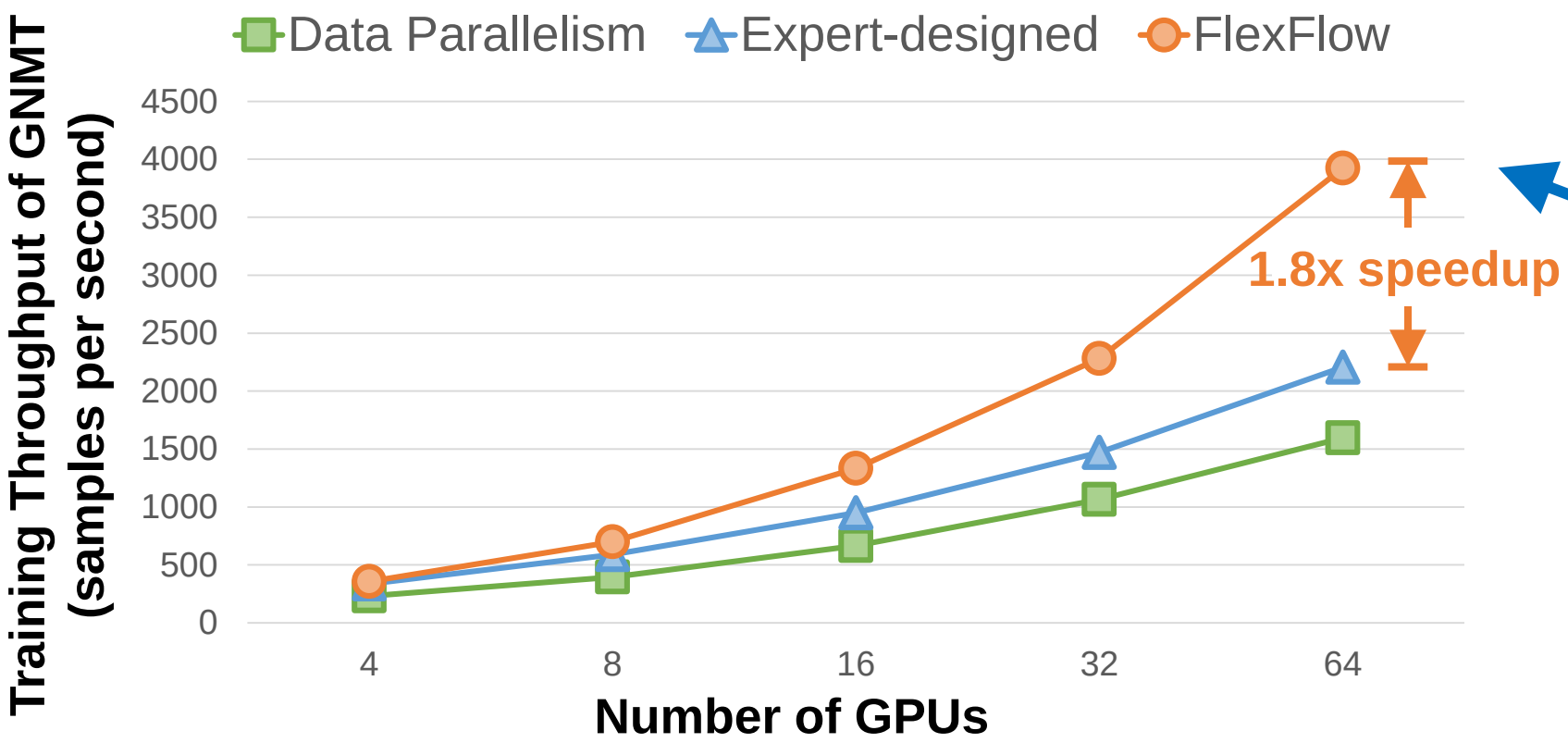


FlexFLow并行搜索逻辑架构



FlexFLow软件栈

二、自动并行技术及已有框架

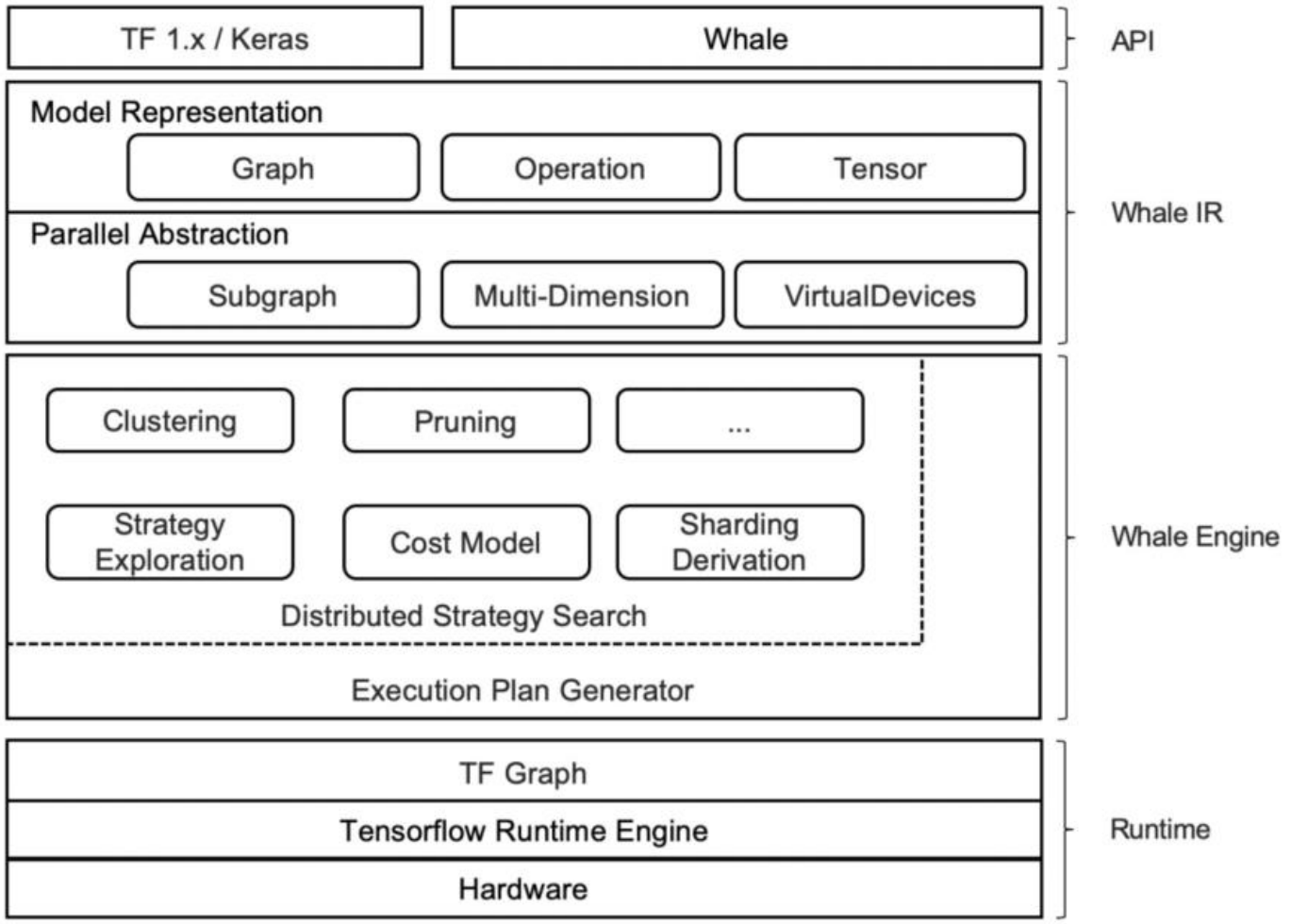


Faster and more scalable strategies than expert-designed

二、自动并行技术及已有框架

序号	模型名称	模型参数量（M）	节点数	单节点GPU数	数据并行效率（sample/s）	自动并行效率（sample/s）	超参数说明
1	AlexNet	60	1	2	2683.58	2902.06	batch size:512 GPU类型：T4 Epoch：5
2	AlexNet	60	1	2	2934.09	3158.45	batch size:1024 GPU类型：T4 Epoch：5
3	AlexNet	60	1	2	2110.85	2135.81	batch size:2048 GPU类型：T4 Epoch：5
4	Inception V3	24	1	2	216.36	215.96	batch size:128 GPU类型：T4 Epoch：5
5	Inception V3	24	1	2	218.72	215.34	batch size:160 GPU类型：T4 Epoch：5
6	Inception V3	24	1	2	228.34	239.96	batch size:200 GPU类型：T4 Epoch：5

二、自动并行技术及已有框架——Whale



Whale Framework

Case 1: Data Parallel

```
with wh.cluster():
  with wh.replica():
    out = Model()
```

Case 2: Large Scale Classification (DP + Op Sharding)

```
cluster = wh.cluster(layout={"all"})
with cluster:
  with wh.replica():
    features = ResNet50(inputs)
  with wh.split():
    logits = FC(features)
    predictions = Softmax(logits)
```

Case 3: Model Parallel

```
with wh.cluster():
  with wh.stage():
    out = ModelPart1()
  with wh.stage():
    out = ModelPart2(out)
```

Case 4: Bert Pipeline+DP

```
with wh.cluster():
  with wh.replica():
    with wh.pipeline(micro_batch=4):
      with wh.stage():
        out = embedding(inputs)
        out = encoder_0_8(out)
      with wh.stage():
        out = encoder_8_16(out)
      with wh.stage():
        out = encoder_16_24(out)
        out = pooler(out)
```

Case 5: Auto Parallel

```
wh.auto_parallel()
out = Model()
```

并行代码示例

一、分布式深度学习综述

- Meng Wang et al., 2020, A Survey on Large-scale Machine Learning
- JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learning
- Shuo Ouyang et al., 2020, Communication optimization strategies for distributed deep neural network training: A survey
- RUBEN MAYER et al., 2019, Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools

二、自动并行技术及已有框架

- 自动并行问题定义
- MindSpore自动并行
- 自动并行框架FlexFlow
- 分布式训练框架Whale

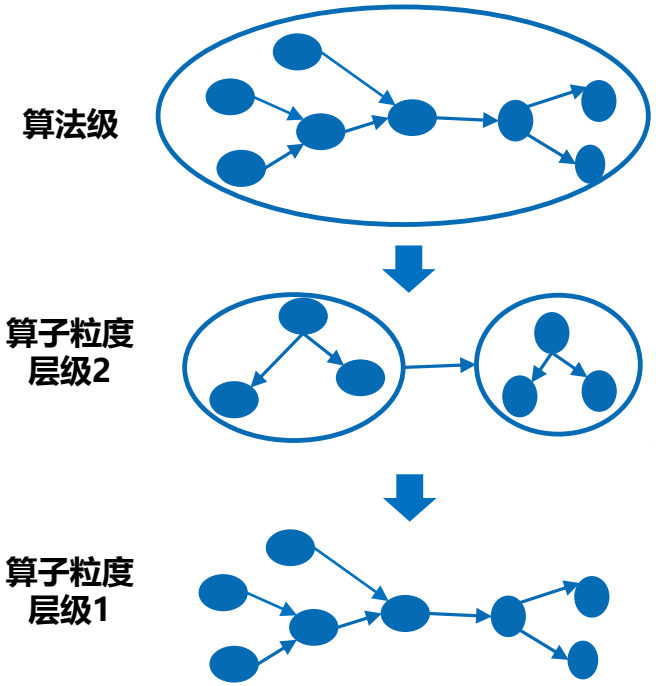
三、优化思考

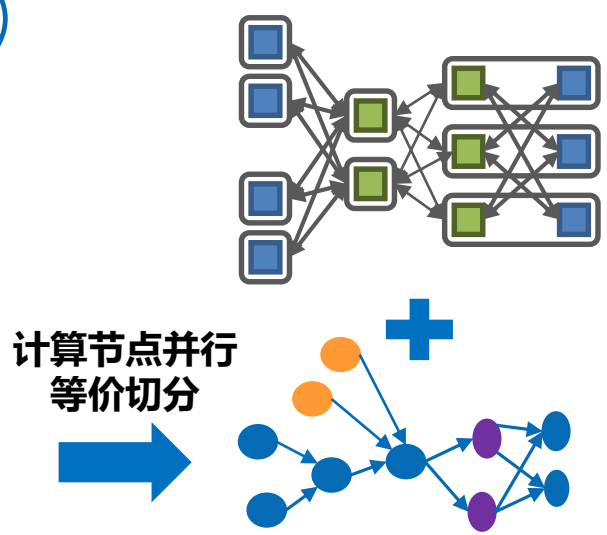
- 并行搜索空间建模
- 边训练边搜索模式
- 自感知自优化配置

三、优化思考

优化方向思考1：有效的并行搜索空间建模

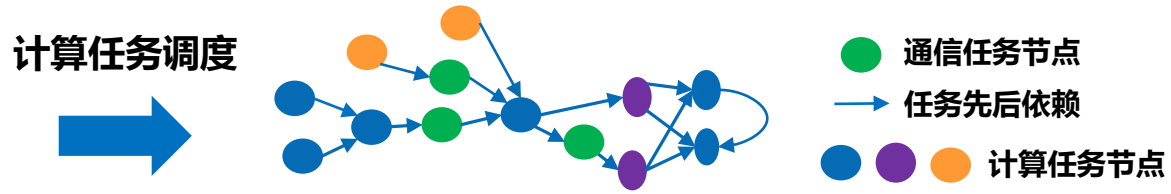
- ✓ FlexFlow从SOAP四个维度对并行搜索空间进行建模，每个算子可能从SOAP四个维度进行并行切分
- ✓ 并行搜索空间的建模直接影响自动并行效率及并行策略搜索复杂度
- ✓ 算子图的可切分维度只需要满足数学计算上的等价切分原则即可，这个角度看数据并行与模型并行没有差别





并行搜索空间由两步确定：

- 算子图并行等价切分：将初始算法图映射到G1
- 任务调度映射：将G1的节点计算任务调度到集群映射为执行任务图T
- 可以认为并行搜索策略空间是：针对算法计算图的图操作变换空间。



算法初始计算图（G） **算法切分计算图空间（G1）** **执行任务图（T）**

三、优化思考

优化方向思考1：有效的并行搜索空间建模

✓ 并行策略搜索问题可以简单表示为：

图变换操作表示为：$G(V_G, E_G) \xrightarrow{f} T(V_T, E_T)$：$T = f(G)$

并行策略执行时间表示为：$time = S\ (T)$

最优并行策略表示为：$\hat{T} = \min_{T \in f(G)} S(T)$

$S.T.\ \ C(t, E_{cluster}) \leq C(E_{cluster})$ 链路带宽限制

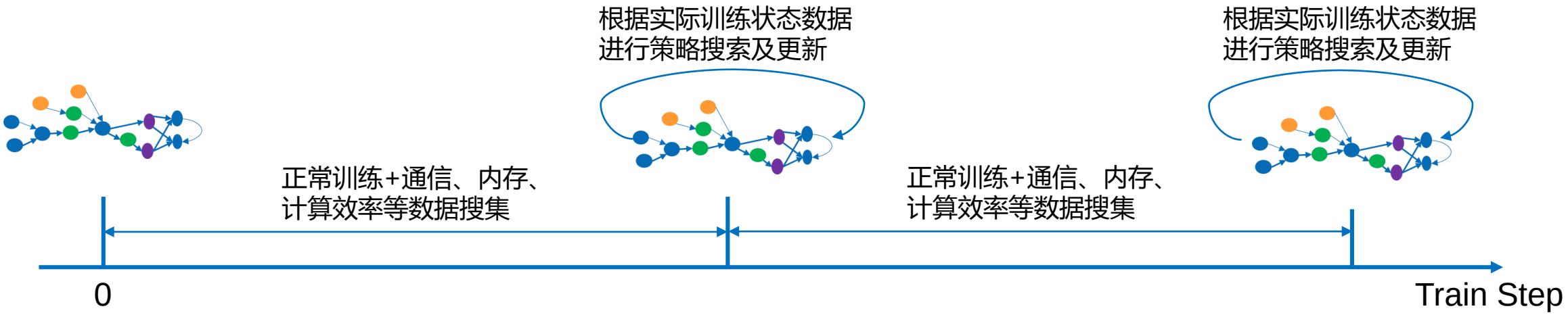
$M(t, N_{cluster}) \leq M(N_{cluster})$ 节点内存限制

✓ 每个图变换操作类似一个决策分支，对于最终并行策略性能产生一定影响

三、优化思考

优化方向思考2：Search while training（边训练边搜索）

- ✓ 现有自动并行由于搜索空间爆炸问题导致在大模型训练场景搜索时间过长
- ✓ 两个指标评价集群并行计算瓶颈点：集群通信链路拥堵程度、集群计算节点计算效率
- ✓ 通过两个指标的评估，将缓解通信拥堵、提高节点计算效率作为下一步并行策略的搜索方向
- ✓ 假设模型自动并行搜索需要100万步，通过边训练边搜索能否将搜索代价降低到10X1万步？

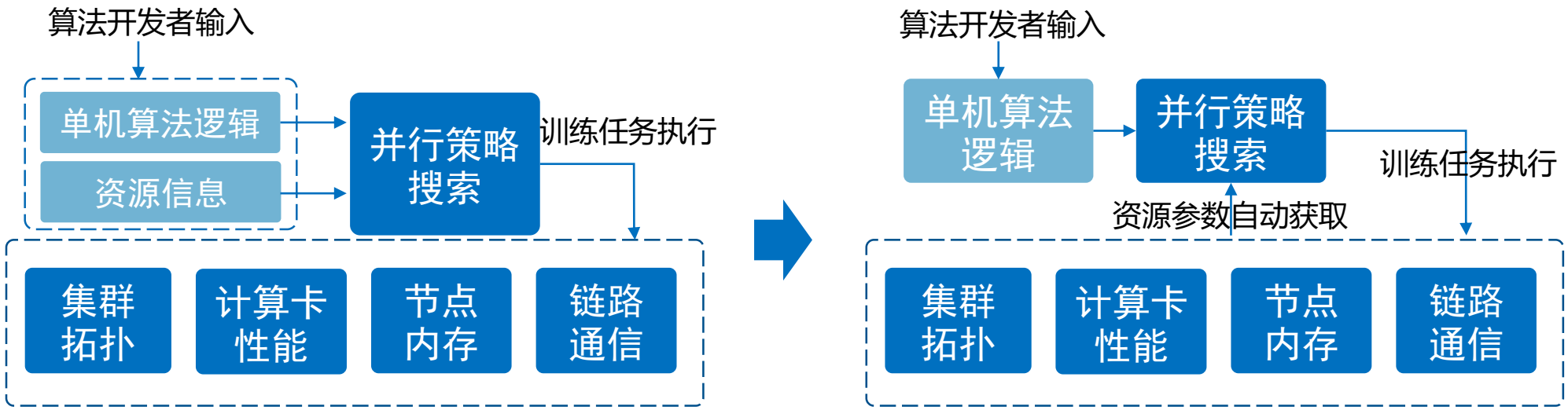


三、优化思考

优化方向思考3：并行超参优化（自感知、自优化能力）

✓ 我们能够通过自动并行搜索算法求解出更好的并行策略，然而仍然面临很多超参的手动调优

✓ 如何根据内存设置等设置Batch size大小？对用户来说如何更好配置集群拓扑、通信等输入参数？



参考文献：

[1] Meng Wang et al., 2020, A Survey on Large-scale Machine Learning

[2] JOOST VERBRAEKEN et al.,2019,A Survey on Distributed Machine Learning

[3] Shuo Ouyang et al., 2020, Communication optimization strategies for distributed deep neural network training: A survey

[4] RUBEN MAYER et al., 2019, Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools

[5] Zhihao Jia et al., 2019, BEYOND DATA AND MODEL PARALLELISM FOR DEEP NEURAL NETWORKS

[6] Ang Wang et al., 2020, Whale: A Unified Distributed Training Framework

[7] Matthias Langer et al., 2020, Distributed Training of Deep Learning Models: A Taxonomic Perspective

感谢聆听
敬请指教