



第八章

新型分布式系统

第八章、新型分布式系统

01

8.1 分布式图计算

02

8.2 分布式机器学习

03

8.3 联邦计算

04

8.4 云计算与云原生

05

8.5 边缘计算



新型分布式系统

- 发展背景
 - 物联网、大数据与人工智能快速发展
 - 任务负载、用户需求和部署环境更加复杂
- 新型分布式系统涌现
 - 分布式图计算
 - 分布式机器学习
 - 联邦学习
 - 云计算与边缘计算
- 技术演进特点
 - 继承传统分布式系统关键技术
 - 面向数据关联性计算迭代性环境异构性进行针对性优化
- 发展趋势
 - 系统架构不断演进数据划分、通信优化、容错与调度能力持续增强



8.1 分布式图计算

- 面向大规模图数据的分布式计算系统
 - 针对节点数和边数极大的图结构数据
 - 将图数据划分并存储在多个计算节点上
 - 通过并行计算提升整体处理能力
- 支持高度关联、迭代式的数据处理任务
 - 图计算中节点之间存在复杂依赖关系
 - 计算过程通常需要多轮迭代逐步收敛
 - 典型算法包括 PageRank、最短路径、连通分量等
- 大数据与人工智能时代的重要计算基础
 - 为社交网络分析、推荐系统、知识图谱提供支撑
 - 是分布式机器学习和图神经网络的重要底层平台
 - 促进大规模数据价值的挖掘与利用

应用背景

- 现实系统中大量数据以图结构形式存在
 - 社交网络（用户关系）
 - 推荐系统（用户-物品关系）
 - 知识图谱（实体-关系）
- 图规模持续增长
 - 节点、边数量呈指数级上升



传统单机计算难以满足需求

为什么需要分布式系统

- 存储瓶颈
 - 单机内存与磁盘容量有限，无法容纳超大规模图数据（海量节点与边）
 - 数据需要频繁分块与外存访问，效率低下
- 计算瓶颈
 - 图算法通常需要多轮全图遍历
 - 单机计算能力难以在可接受时间内完成任务
- 性能瓶颈
 - 图计算过程中存在大量随机内存访问，频繁的数据访问与节点间通信
 - I/O 和通信延迟成为系统主要开销
- 可扩展性需求
 - 数据规模和计算负载持续增长，需要通过增加节点实现横向扩展
 - 支持计算与存储资源的动态扩展

图计算的基本特征

- 强关联性
 - 图数据中节点之间存在复杂关联关系
 - 计算过程难以完全并行化
- 迭代计算特性
 - 计算通常以多轮迭代方式进行
 - 每轮计算依赖前一轮的中间结果
- 通信密集型计算
 - 需要频繁进行节点间数据交换
 - 通信和同步成本高于计算成本
- 系统性能受限于最慢节点
 - 同步计算模式下存在“短板效应”
 - 节点异构性进一步加剧性能不均衡

分布式图计算的定义与区别

■ 分布式图计算的定义

- 将大规模图数据划分到多个计算节点
- 各节点并行执行图计算任务
- 通过节点间通信与同步完成整体计算
- 对外表现为统一的逻辑计算系统

■ 与传统分布式批处理的区别

- 批处理系统：以独立任务为主，任务间依赖较少
- 图计算系统：任务之间高度依赖，需要频繁通信与同步
- 图计算对系统提出更高要求：通信效率，同步机制，容错能力

■ 分布式图计算系统按架构可分为以下几类

- 基于迭代 MapReduce 模型的系统
 - 基于 BSP 模型的类 Pregel 系统
 - 基于 GAS 模型的类 GraphLab 系统
 - 其他架构的分布式图计算系统
- ## ■ 不同架构针对图计算的特点提出不同优化思路

基于迭代 MapReduce 模型的系统

- 早期图计算系统依赖现有分布式计算平台
 - 典型代表：MapReduce
- 优点
 - 可直接利用成熟的分布式计算框架
 - 具有一定的可扩展性
- 局限性
 - MapReduce 仅支持粗粒度两阶段计算
 - 原生模型不适合迭代计算
- 性能瓶颈
 - 图算法通常需要多轮迭代
 - MapReduce 每轮迭代需启动新作业
 - 中间结果需频繁进行磁盘读写
 - 节点间数据传输开销大
 - 对稀疏图和细粒度计算效率较低

支持迭代优化的 MapReduce 衍生系统

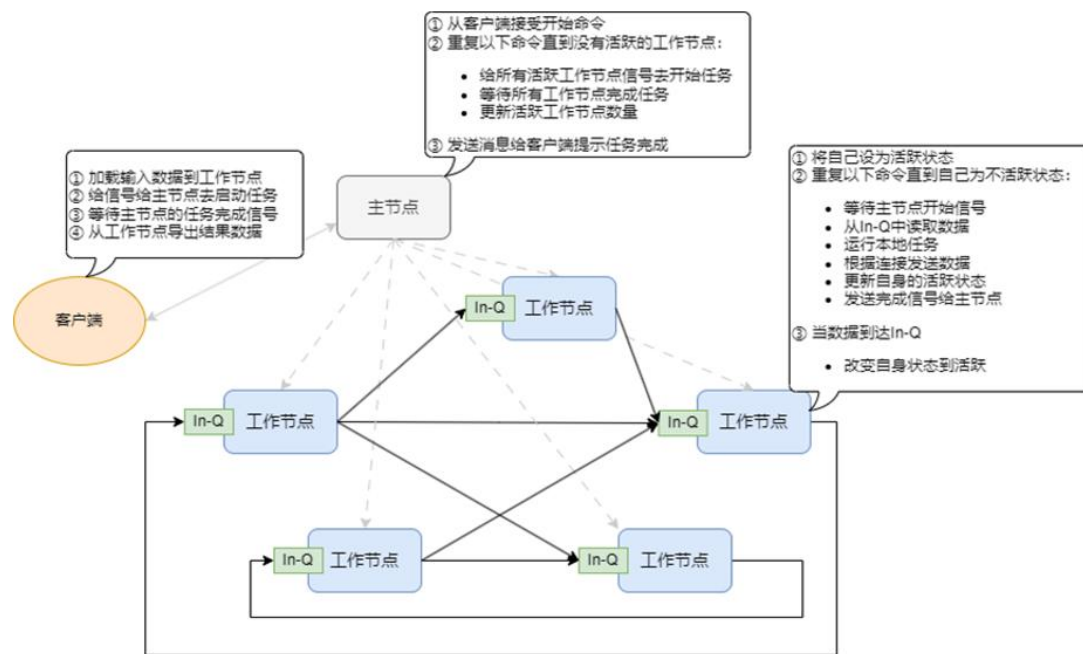
- 为解决迭代效率问题，提出改进型系统
- Haloop
 - 缓存固定数据与 Reducer 输出
 - 提升迭代计算性能
- GBASE
 - 提供图数据分块、索引、压缩与存储机制
 - 提出新的邻接矩阵存储方式
 - 提升空间利用率与查询效率
- Pegasus, Surfer等
- 目标
 - 减少不必要的作业启动和数据读写开销

小结：

- 可利用原有分布式平台
- 具备一定扩展性
- 仍难以满足
 - 图数据的高度耦合性
 - 计算的强迭代性
 - 高性能计算需求

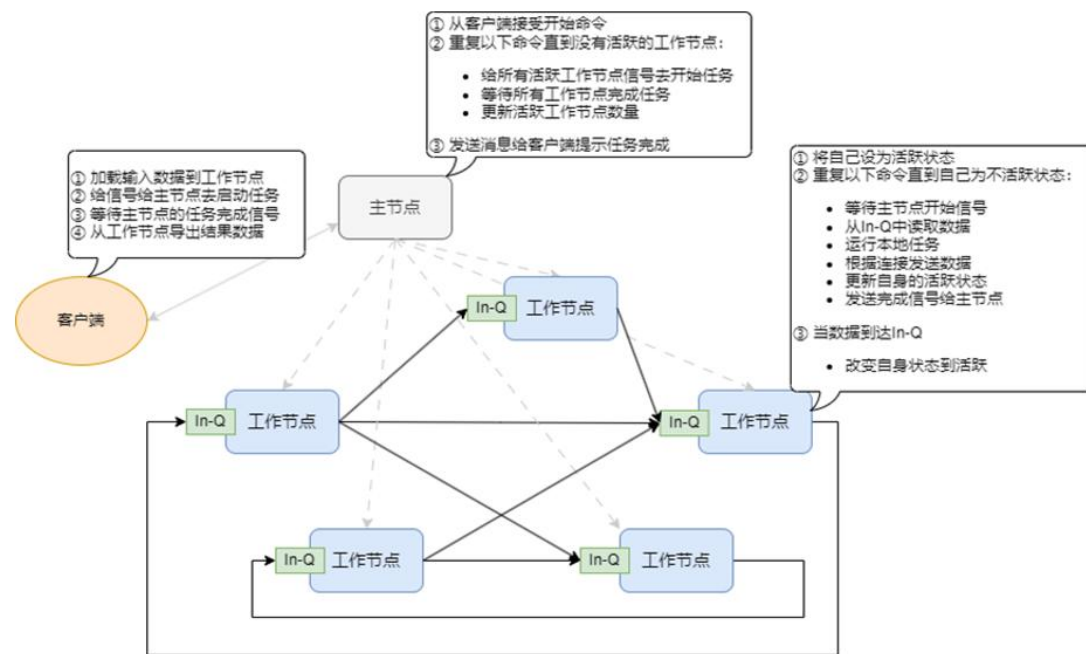
基于 BSP 模型的 Pregel 架构

- 为克服 MapReduce 在迭代计算上的缺陷
Google 于 2010 年提出 Pregel 框架基于整体同步并行 (BSP) 模型
- 代表系统
- Pregel, Pregel+, Pregelx
- BSP模型基本思想:
 - BSP 是一种基于消息通信的并行计算模型
 - 计算过程由一系列超步 (Superstep) 组成
 - 每个超步之间通过全局同步分隔



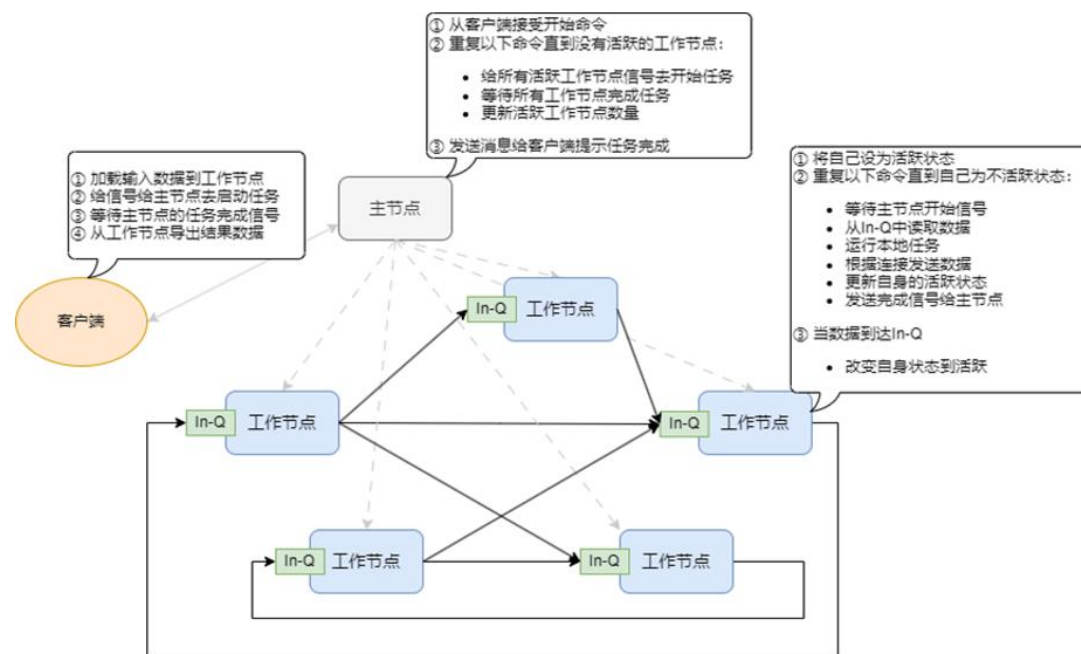
基于 BSP 模型的 Pregel 架构

- 本地计算 (Local Computation)
 - 每个节点仅访问自身负责的图数据
 - 独立完成顶点计算任务
 - 支持节点内部的并行执行
- 消息通信 (Message Passing)
 - 节点通过消息机制与邻居节点通信
 - 消息携带计算结果或状态信息
 - 下一轮计算依赖收到的消息
- 路障同步 (Barrier Synchronization)
 - 所有节点完成当前阶段后统一同步
 - 防止节点之间计算进度不一致
 - 保证迭代计算的正确性



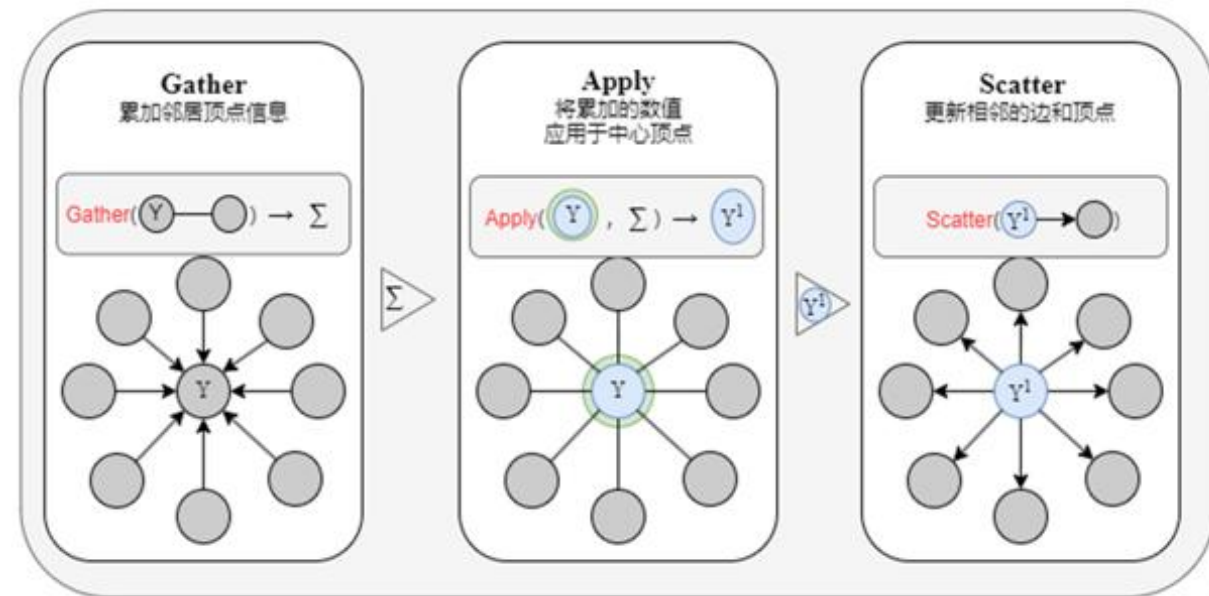
Pregel 架构特点与优势

- 单作业、多超步完成迭代计算
- 节点通过消息传递交换中间结果
- 避免频繁磁盘读写
- 相比 MapReduce，迭代效率显著提升
- 适合大规模静态图计算



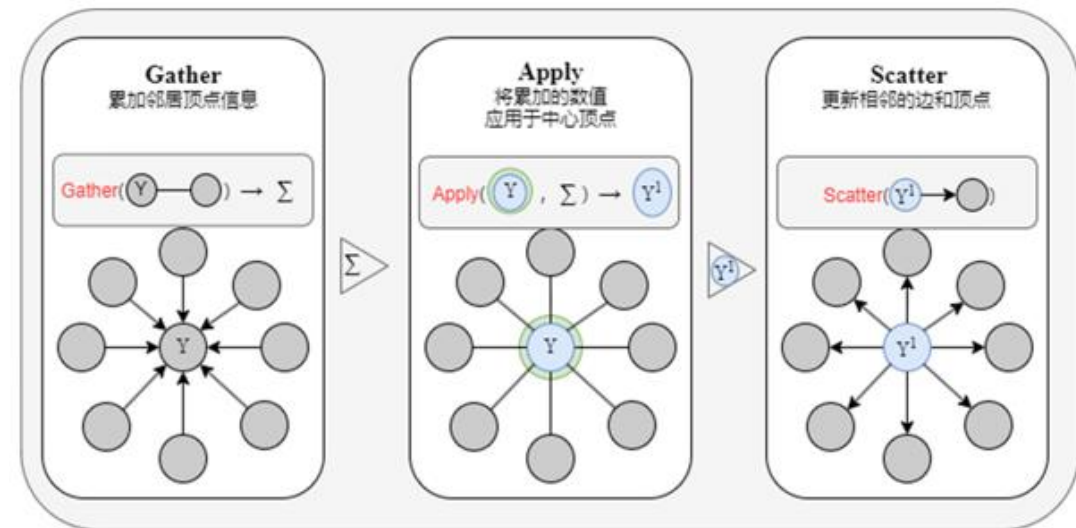
基于 GAS 模型的 GraphLab 架构

- GraphLab 提出的细化顶点中心模型
- 目标
 - 提高计算并行性
 - 优化不均匀图数据处理
- 代表系统：
 - GraphLab、PowerGraph、PowerLyra



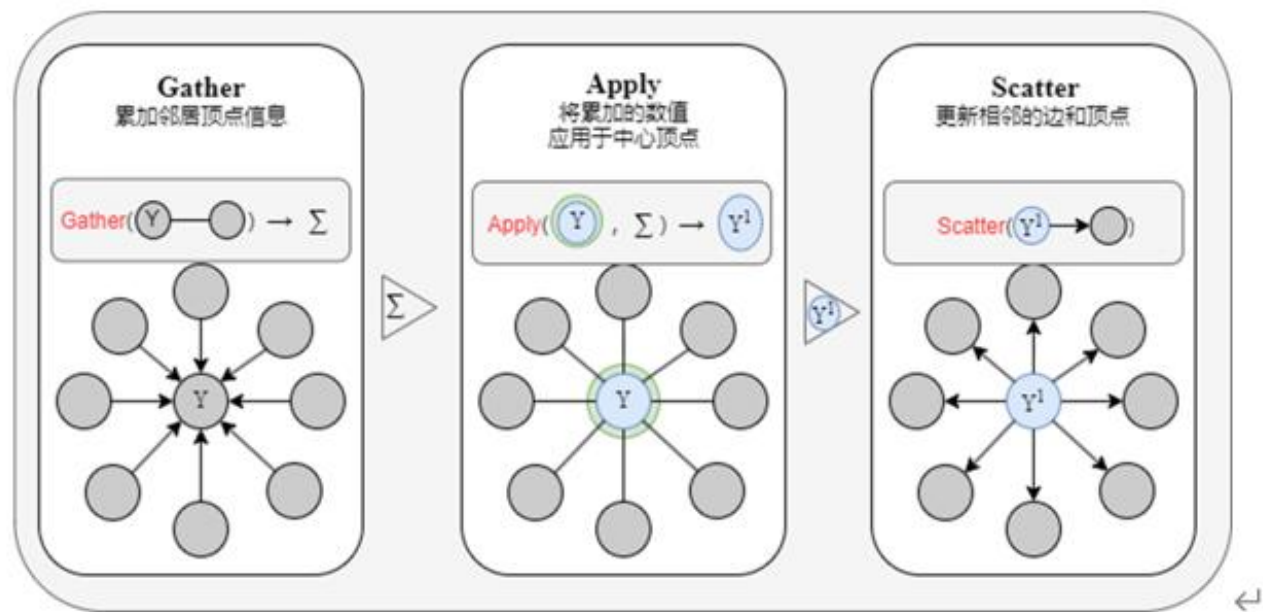
GAS 模型的三阶段计算流程

- Gather (收集)
 - 收集邻居顶点信息
- Apply (应用)
 - 更新当前顶点状态
- Scatter (分发)
 - 向邻居发送更新结果
- 明确划分计算逻辑，提高灵活性



GAS 架构的特点与适用场景

- 支持更高并发度
- 适合幂律分布图数据
- 高度连接节点处理效率高
- 编程模型更灵活，但实现复杂度较高

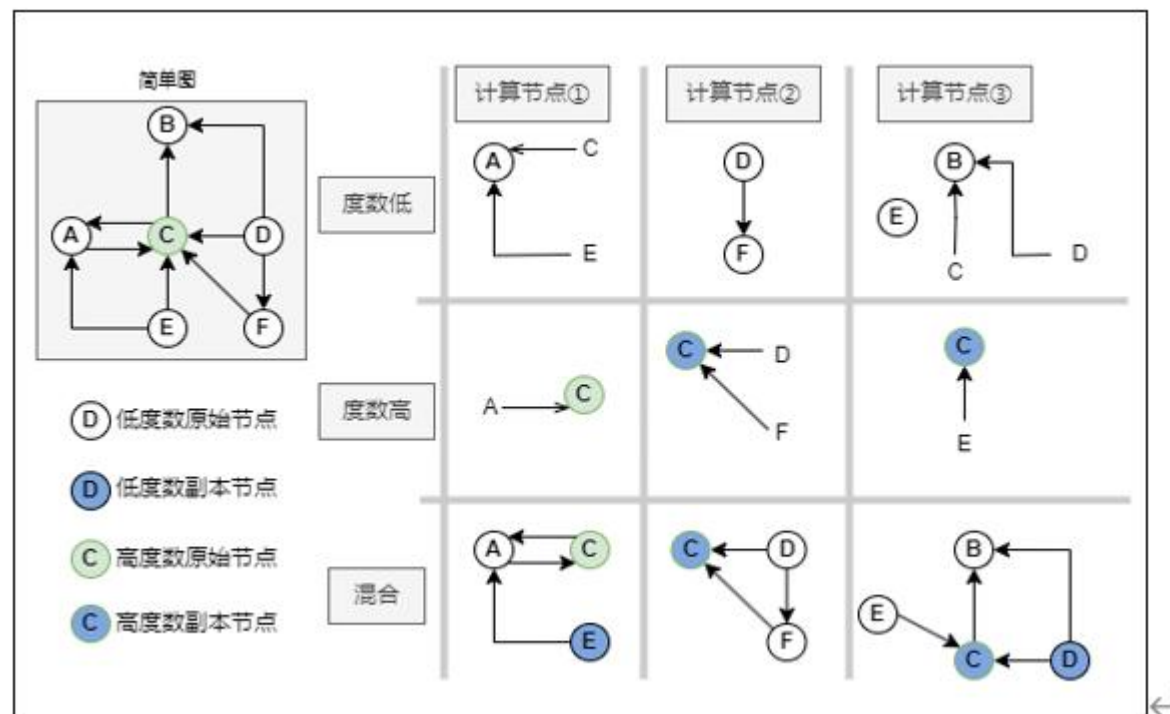


其他分布式图计算架构

- 在 BSP / GAS 基础上的扩展与融合
 - 结合不同计算模型的优势，在性能、灵活性与易用性之间进行权衡，面向更丰富的应用场景
- 代表系统：
 - Trinity：基于内存云的图计算平台，支持在线查询与离线计算
 - GraphX：构建在 Spark 之上，结合 Pregel 模型与 Spark 内存计算

8.1.3 关键技术--数据划分

- 基于顶点的划分（切割边）
 - 顶点及其邻边存储在同一节点
 - 计算以顶点为中心，消息显式传递
 - 切割边越多，通信开销越大
 - 在幂律分布图中易产生负载不均
- 基于边的划分（切割顶点）
 - 边分布在不同节点，顶点被复制
 - 减少通信量，负载更均衡
 - 编程复杂度较高，副本增加开销
- 混合划分
 - 区分高度数与低度数顶点
 - 兼顾负载均衡与通信效率
 - 代表系统：PowerLyra



混合划分

8.1.3 关键技术---同步策略

同步执行模型

- 并发工作进程按全局同步顺序执行
- Pregel 系统采用 超步 (Superstep)
- 每个超步包括
 - 计算阶段
 - 通信阶段
 - 路障同步
- 特点
 - 状态一致性强
 - 编程与调试简单
- 优点
 - 计算过程清晰
 - 易于实现和调试
 - 适合计算与通信密集型算法
- 缺点
 - 负载不均会导致“慢节点”问题
 - 性能受最慢节点限制
 - 需要较高内存开销保存中间状态

异步执行模型

■ 基本特点

- 不存在全局同步屏障
- 各节点可独立推进计算
- 不需要等待其他节点完成当前迭代

■ 执行方式

- 节点在本地数据可用时立即执行计算
- 计算结果可立即传播给相关节点
- 有效缓解同步模型中的“慢节点”问题

■ 代表系统

- Maiter
 - 基于异步迭代计算模型
 - 通过增量更新减少不必要的计算

■ 主要问题

- 锁竞争
 - 邻接顶点并发访问共享状态
- 通信频繁
 - 消息发送次数显著增加
- 收敛性问题
 - 执行顺序不确定，可能影响算法收敛

混合执行模型

■ 基本思想

- 结合同步执行与异步执行的优点
- 根据运行状态动态切换执行策略

■ 执行特点

- 在局部区域采用异步计算
- 在全局阶段进行必要的同步
- 动态调整顶点处理顺序

■ 优势

- 减少同步等待时间
- 提高计算并发度
- 加快整体收敛速度

■ 代表系统

- GRACE：通过动态调度优化顶点更新顺序
- PowerSwitch：根据运行时特征在同步与异步模式间切换

通信开销与优化目标

■ 图计算中的通信特性

- 在分布式图计算中，通信开销通常高于计算开销
- 节点之间需要频繁交换中间计算结果
- 网络延迟和带宽成为系统性能瓶颈

■ 主要原因

- 数据关联性强
 - 图结构中顶点与边高度耦合
 - 计算过程中依赖大量邻居信息
- 消息频繁传递
 - 迭代式算法需要多轮通信
 - 消息粒度细、小而多

■ 通信优化目标

- 减少消息数量
 - 合并消息，避免不必要的中间结果传输
- 降低通信频率
 - 减少同步次数，提高每次通信的有效负载

消息缓存机制

■ 消息通信方式

- 图计算系统通过显式通信接口进行消息传递
- 消息在节点之间传输，用于交换中间计算结果
- 通常采用批量发送与接收方式

■ 常用通信协议

- MPI、Thrift、ActiveMQ

■ 消息缓存策略

- 按来源缓存
 - 将来自同一发送节点的消息集中存放
 - 减少多次发送带来的通信开销
- 按目标缓存
 - 将发往同一目标节点的消息合并处理
 - 便于批量传输与消息聚合

■ 优点

- 减少通信次数
- 降低通信开销

推送 / 拉取策略

■ 推送 (Push) 模式

- 活跃顶点将更新信息主动发送给邻居顶点
- 由源顶点触发通信操作
- 适合活跃顶点数量较少的阶段
- 需要对邻居更新加锁以保证一致性

■ 拉取 (Pull) 模式

- 顶点从其邻居顶点获取所需信息
- 由目标顶点触发数据读取
- 适合活跃顶点数量较多的阶段
- 可减少锁竞争和写冲突

■ 推送与拉取结合

- 降低锁开销
- 减少中间消息

■ 典型应用场景

- BFS: 早期阶段使用 Push, 后期阶段使用 Pull
- PageRank: 根据活跃顶点比例切换策略

容错机制概述

- 容错机制是分布式图计算系统的重要组成部分
 - 用于应对节点故障、网络异常等问题
 - 保证系统计算的正确性与可靠性
- 分布式图计算中的错误处理流程
 - 故障检测
 - 系统恢复
- 容错机制的两类关键措施
 - 故障发生前的预防措施
 - 故障发生后的系统恢复



故障发生前的预防措施

- 状态保存与检查点机制
 - 在计算过程中定期保存系统状态
 - 状态包括
 - 顶点状态
 - 消息状态
- 系统实现示例：
 - Pregelix：支持在任意超步边界保存检查点，将检查点写入分布式文件系统（HDFS）
 - Piccolo：提供同步与异步检查点接口，同步检查点适合迭代算法（如 PageRank），异步检查点适合长时间运行任务
- 其他预防手段：消息记录，数据复制，周期性快照

故障发生后的系统恢复

- 基于检查点的恢复机制
 - 故障节点被标记并隔离
 - 系统从最近一次检查点恢复计算状态
 - 重新调度失败节点上的任务
- 系统实现示例：
 - Pregelix：将故障机器加入黑名单重新加载最近检查点状态
 - PowerGraph：利用数据图快照进行恢复在恢复后重新安排计算任务
- Trinity 的恢复策略：
 - 根据查询类型与故障情况重新加载数据或重启故障机器
- 适用于在线查询与离线计算场景

代表系统

Pregel

- 系统定位
 - Google 提出的分布式图计算系统
 - 面向大规模图数据的高效迭代计算
- 系统架构与模型
 - 采用 Master-Worker (Master-Slave) 架构, 基于 BSP (整体同步并行) 模型, 引入超步 (Superstep) 概念
- 编程模型特点
 - 以顶点为中心的计算模型
 - 每个超步中执行用户定义的顶点计算函数
 - 通过消息在超步之间传递中间结果
- 系统优势
 - 编程模型简单、表达能力强

Distributed GraphLab

- 系统定位
 - GraphLab 的分布式扩展版本
 - 面向高并发图计算场景
- 核心设计
 - 引入类似 BSP 的处理流程
 - 采用数据版本控制减少网络拥塞
 - 使用流水线分布式锁降低延迟影响
- 执行引擎与容错机制
 - 染色引擎与锁引擎
 - 支持同步与全异步快照
 - 基于 Chandy-Lamport 快照算法
- 特点: 在保证一致性的同时提升系统性能

代表系统

PowerLyra

- 系统定位
 - 基于 GAS 模型的分布式图计算系统
 - 面向幂律分布和高度倾斜图数据
- 核心思想
 - 区分低度数顶点与高度数顶点
 - 低度数顶点：集中计算，减少通信
 - 高度数顶点：分布式计算，平衡负载
- 关键技术
 - 混合划分策略（顶点切割 + 边切割）
 - 平衡 p 路混合切割算法
 - 在 GAS 各阶段减少不必要通信

GRAPE

- 面向大规模图的高性能分布式计算引擎
- 支持直接并行化现有顺序图算法
- 基于部分评估与增量计算
- 减少跨节点通信和调度开销

GraphScope

- 阿里巴巴开发的新一代大图计算系统
- 图分析引擎继承 GRAPE
- 支持遍历、模式匹配、迭代算法与采样
- 提供统一编程接口，支持与机器学习平台集成

第八章、新型分布式系统

01

8.1 分布式图计算

02

8.2 分布式机器学习

03

8.3 联邦计算

04

8.4 云计算与云原生

05

8.5 边缘计算



8.2 分布式机器学习

- 如何管理大规模模型与海量数据的训练执行？
- 管理维度：在任务执行过程中，协调各节点之间的计算逻辑与数据流动
- 目标：
 - 克服单机性能瓶颈
 - 保持系统高效运作
 - 确保模型精度与收敛性

8.2.1 背景与概念



- 模型规模剧增：从千万级参数跃升至**万亿级**（如GPT-3.5约2万亿参数，GPT-4规模更大）
- 数据量庞大：ImageNet包含1400万张图像，医疗数据集MIMIC-III包含4.6万患者信息
- 硬件瓶颈：单机在**存储、计算、通信能力**上存在明显不足
- 核心目标：通过分配计算资源与任务，聚合训练结果，在保证精度和收敛性的基础上，提升**训练速度**

任务分配方式：数据并行

- 核心思想：样本数据分配到不同工作节点，各节点使用相同的模型副本
- 流程：
 - 数据切分：训练数据 N 分配给 k 个节点，每块大小 N/k
 - 局部计算：各节点计算局部梯度 G_i
 - 全局聚合：通过通信机制对梯度进行聚合 $\Sigma G_i/k$
 - 模型更新：依据聚合梯度更新参数，开启新一轮迭代

任务分配方式：模型并行

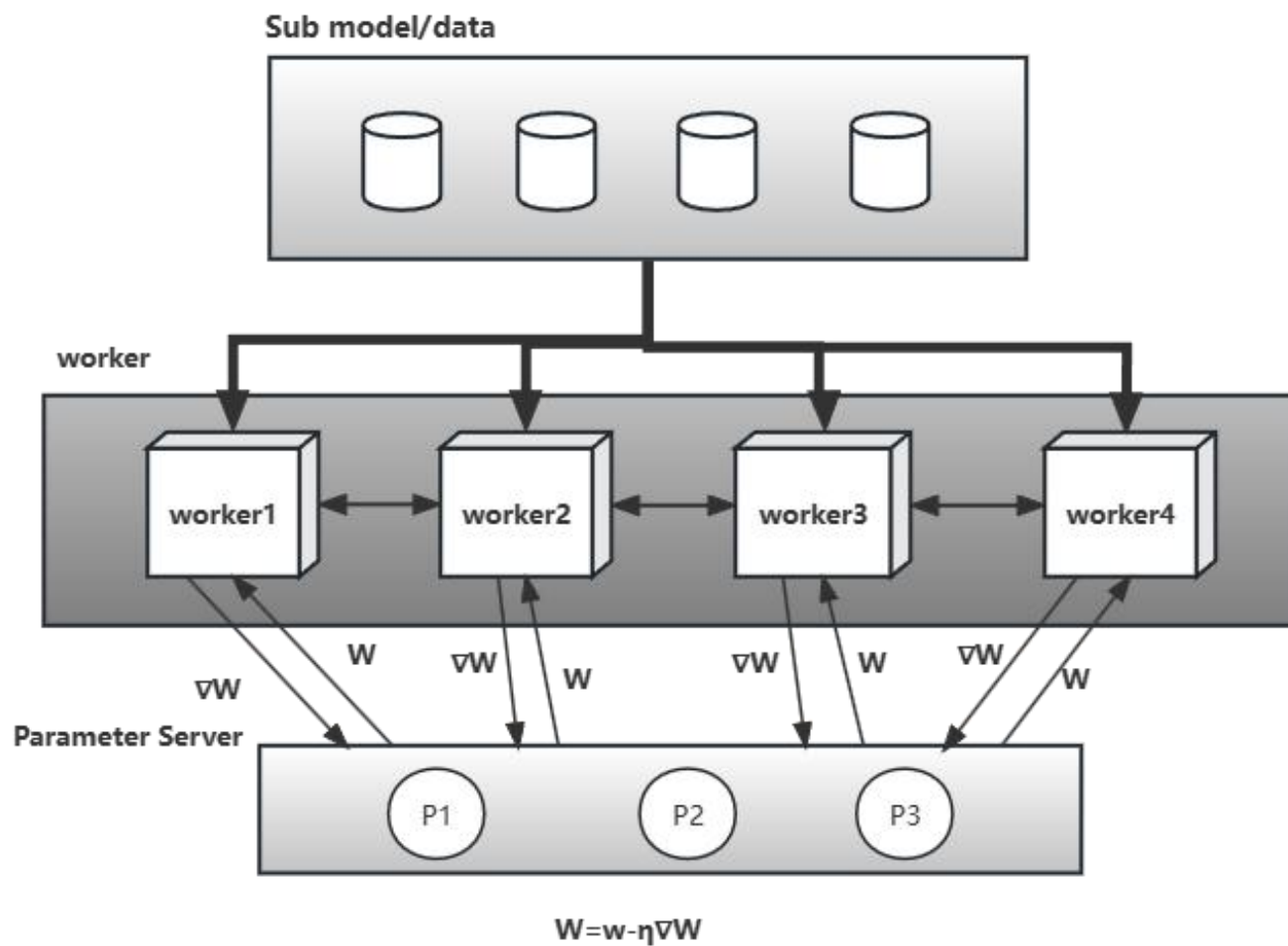
- 核心思想：模型拆分为不同算子并分配到各节点，各节点使用相同的数据副本
- 适用场景：
 - 存在大型算子或模型参数量巨大
 - 不同算子的计算结构差异明显
- 适用场景：通常以神经网络的层为单位进行划分
- 聚合方式：每个节点负责部分参数，通过参数并集操作完成模型更新

8.2.2 主要架构

参数服务器架构

- 节点类型:
 - 工作节点 (Worker): 负责模型训练、计算梯度、推送(Push)梯度与拉取(Pull)参数
 - 服务器节点 (Server): 负责聚合梯度、更新参数并进行全局一致性约束
- 运行机制: 服务器维护 W 的分区, 工作节点计算 ∇W 后传给服务器, 服务器执行 $W = W - \eta \nabla W$
- 优势: 良好支持大模型参数存储, 方便实现同步或异步并行训练

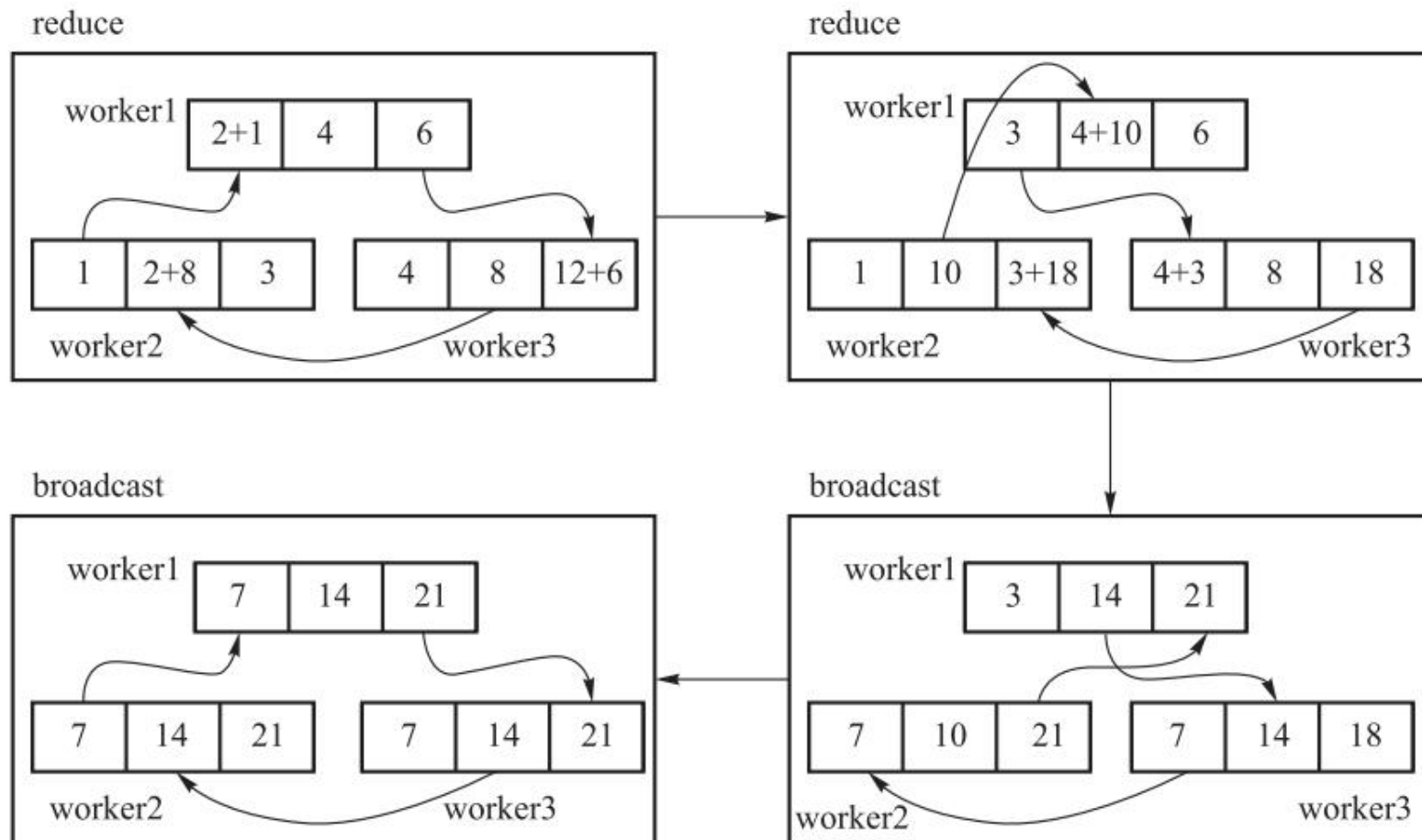
参数服务器架构



AllReduce 架构

- 去中心化设计：解决了中心化架构的问题
- 通信机制：每个节点仅与前向/后向节点通信，经网络带宽阻塞过多步迭代使各节点均具备完整梯度信息
- Reduce-Broadcast 实现
 - Reduce阶段：梯度分区分段累加
 - Broadcast阶段：迭代传递使所有节点获得完整梯度信息
- 支持框架：TensorFlow、PyTorch、MXNet等主流框架均支持

AllReduce 架构参数更新



数据流架构 (Data Flow)

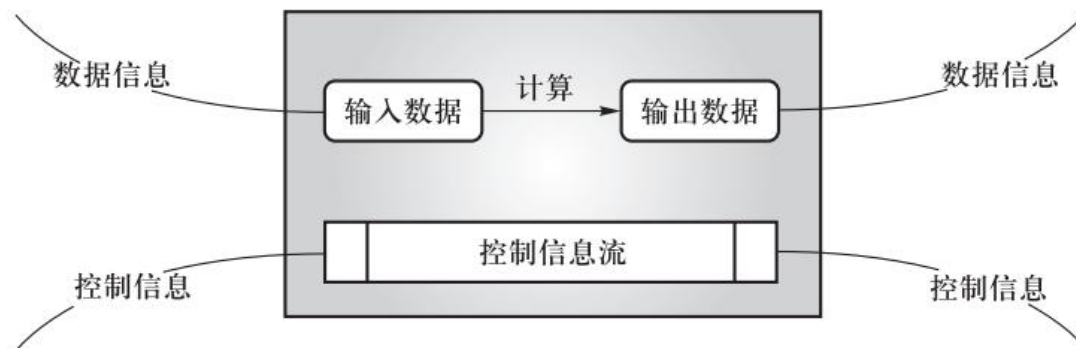
- 将分布式训练抽象为**计算图**

- 将机器学习过程抽象为 DAG (有向无环图)
- 节点代表算子 (数据处理), 边代表数据流动

- 节点组成:

- 控制信息流: 决定任务触发、数据完整性判断及下游调度
- 计算数据流: 负责数据接收、初始化及中间结果传递

- 优势: 极佳的模型并行适应性 (如 Spark DAG, TensorFlow)



8.2.3 关键技术

参数信息同步

- 整体同步并行 (BSP)
 - 阻塞等待所有节点梯度，确保高精度收敛，但存在“木桶效应”
- 异步并行 (ASP)
 - 无须等待，计算资源利用率高，但可能导致梯度抖动或不收敛
 - 优势：极佳的模型并行适应性（如 Spark DAG, TensorFlow）
- 延时同步并行 (SSP)
 - 核心思想：设置延迟阈值 s ，控制最快与最慢节点的迭代差距
 - 约束： $t_{fast} - t_{slow} \leq s$ 兼顾训练速度与精度收敛

通信优化与容错机制

■ 通信优化策略

- 时域滤波：增加通信间隔（多次更新后再通信）或非对称 Push/Pull 频率
- 空域滤波：模型量化、低秩化处理、梯度模型过滤（设置阈值 t ）

■ 容错与可靠性

- 备份节点：监控主节点状态，故障时无缝接管任务
- 检查点 (Checkpoint)：定期保存模型参数，实现故障后的增量恢复
- BML 算法：通过一阶邻居节点均匀划分故障任务进行梯度聚合

资源管理与超参数调优

- 任务调度器
 - 监测节点状态，负责计算任务的重分配
 - 支持 Yarn, Kubernetes 等现有资源管理器复用
- 分布式超参数搜索
 - 网格搜索、随机搜索、贝叶斯优化
 - Ray Tune (支持 Early-stopping 早期停止策略以节省资源)

8.2.4 代表系统

■ Spark MLlib

- 基于RDD算子接口，提供更高级的机器学习管道
- 继承Spark平台高可扩展性优势

■ Multiverso (微软)

- 基于PS架构实现
- 模型存储：服务器端使用HashMap、Vector、Matrix格式存储
- 优化：使用线程池并行响应请求，分包聚合提高通信效率

■ TensorFlow (谷歌)

- 基于数据流的分布式框架，极受欢迎
- 包含大量op算子（数值、矩阵、神经网络等），与机器学习结合紧密

■ PaddlePaddle (百度)

- 支持万亿级稀疏参数模型训练
- 近期提出4D混合并行策略，支持千亿级稠密参数模型

第八章、新型分布式系统

01

8.1 分布式图计算

02

8.2 分布式机器学习

03

8.3 联邦计算

04

8.4 云计算与云原生

05

8.5 边缘计算



8.3 联邦学习

- 数据孤岛挑战：商业竞争、政策监管与隐私保护导致数据无法直接共享
- 在不共享用户隐私数据的前提下，如何实现跨机构、跨设备的分布式计算？
- 目标：
 - 保护数据隐私安全
 - 打破数据孤岛，实现联合建模
 - 提升全局模型性能与准确率

8.3.1 背景与概念

- 联邦学习 (FL) 定义：
 - 谷歌于 2016 年提出
 - 旨在不共享隐私数据的前提下进行联合建模
- 核心逻辑：
 - 数据不动模型动：数据保留在本地边缘设备或服务器
 - 参数交互：仅交换模型参数，通过服务器聚合生成全局模型
- 应用实例
 - 医疗机构在不泄露患者病历的前提下，联合训练医学影像诊断模型

联邦学习分类

1. 按客户端特征

■ 跨孤岛联邦 (Cross-silo):

- 对象：组织或数据中心（如医疗机构）
- 特点：数量少、数据量大、计算能力强、稳定性高

■ 跨设备联邦 (Cross-device):

- 对象：大规模移动设备或 IoT 设备（如手机键盘）
- 特点：数量极多、单设备数据少、稳定性低、网络环境复杂

2. 按数据特征分布

■ 水平联邦学习 (Horizontal FL):

- 用户特征重叠多，样本空间重叠少（如不同地区的银行）

■ 垂直联邦学习 (Vertical FL):

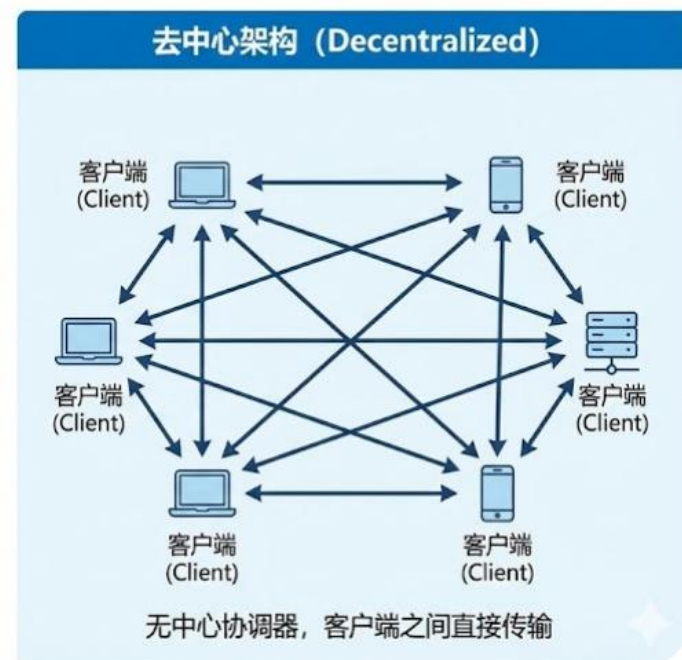
- 样本重叠多，用户特征重叠少（如同一地区的银行与电商）

■ 联邦迁移学习 (Federated Transfer Learning):

- 样本与特征重叠均较少，利用迁移学习克服数据匮乏

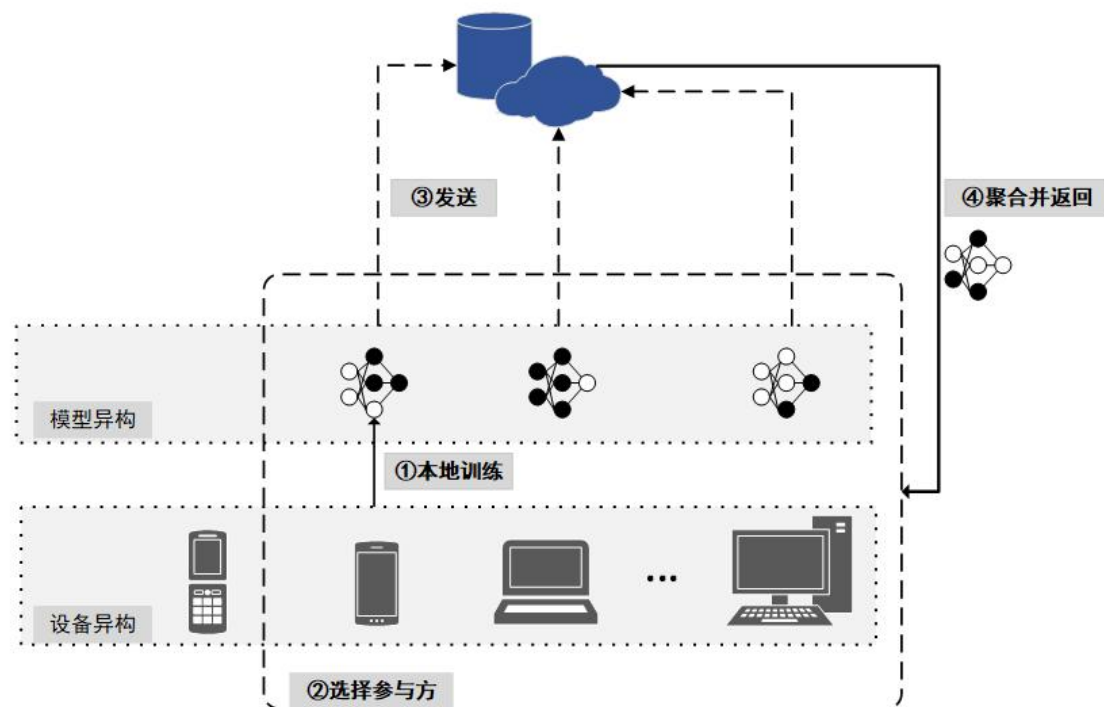
8.3.2 主要架构

- 客户端-服务器架构 (Client-Server):
 - 设有中央服务器负责协调与聚合, 通信开销较低
 - 主流实现: FedAvg 算法
- 去中心架构 (Decentralized):
 - 无中心协调器, 客户端之间直接传输更新, 安全性更高
 - 局限: 需要更复杂的通信协议与更多计算资源



联邦学习训练过程

- 本地训练：客户端利用本地数据更新模型
- 选择参与方：服务器按算法筛选参与节点
- 参与方发送：加密上传本地参数至服务器
- 聚合与分发：服务器聚合生成全局模型并下发更新



联邦学习 vs. 传统分布式学习

- 隐私安全：通信内容必须采用加密技术保护
- 数据异构 (Non-IID)：数据分布不均且非独立同分布，要求负载均衡优化
- 系统异构：设备计算、通信及存储能力差异巨大，存在“长板等待”风险
- 自主权：参与方可随时自主加入或退出训练

8.3.3 关键技术

通信优化

- 算法优化： 增加本地训练次数以减少全局通信轮次，或采用自适应加权聚合
- 客户端选择： 优化采样概率与策略，识别并剔除与全局趋势不一致的无关更新
- 模型压缩：
 - 压缩感知(CS)：对权重向量进行稀疏化与测量矩阵压缩
 - 模型精简：训练小型化核心集网络

负载均衡

- 资源分配优化：利用强化学习（如 Q-learning）动态管理设备的数据量与资源使用，降低延迟
- 基于资源的权衡：
 - FedCS 策略：优先选择能在截止期限内完成任务的活跃节点
 - 数据质量评估：根据数据量、类别平衡度得分进行概率采样

动态退出容错

- 异步联邦学习(Async FL):
 - 如FedAsync, 使用时间相关参数对设备贡献加权, 缓解退出/加入带来的波动
- 负面影响缓解:
 - 寻找“朋友”: 使用数据分布相似的节点更新作为丢失节点的替代品
 - MIFA 算法: 利用记忆中的最新更新纠正梯度偏差, 降低不活动节点影响

8.3.4 代表性联邦学习系统

- Spark MLlib
 - 基于DAG (有向无环图) 调度与 RDD (弹性分布式数据集) 接口层次
 - 继承 Spark 平台的高可扩展性, 提供高效的分布式 ML 解决方案
- Multiverso (微软)
 - 基于PS (参数服务器) 架构实现
 - 服务器端采用 HashMap、Vector、Matrix 格式, 关键数据存于连续内存
 - 消息驱动 + 线程池并行响应
- TensorFlow (谷歌)
 - 基于数据流 (Dataflow)
 - 内置大量数值计算、矩阵及神经网络算子, 支持灵活的任务分配与调度

8.3.4 代表性联邦学习系统

■ PaddlePaddle (百度)

- 万亿级稀疏参数及千亿级稠密参数
- 提出 4D 混合并行 策略，拥有丰富的 CV 和 NLP 模型库
- 支持动态图、静态图及两者间的转换

■ MindSpore (华为)

- 支持动静态图统一
- 模型层、表达层、编译优化层、运行层
- 内置数据并行、半自动并行、自动并行

第八章、新型分布式系统

01

8.1 分布式图计算

02

8.2 分布式机器学习

03

8.3 联邦计算

04

8.4 云计算与云原生

05

8.5 边缘计算



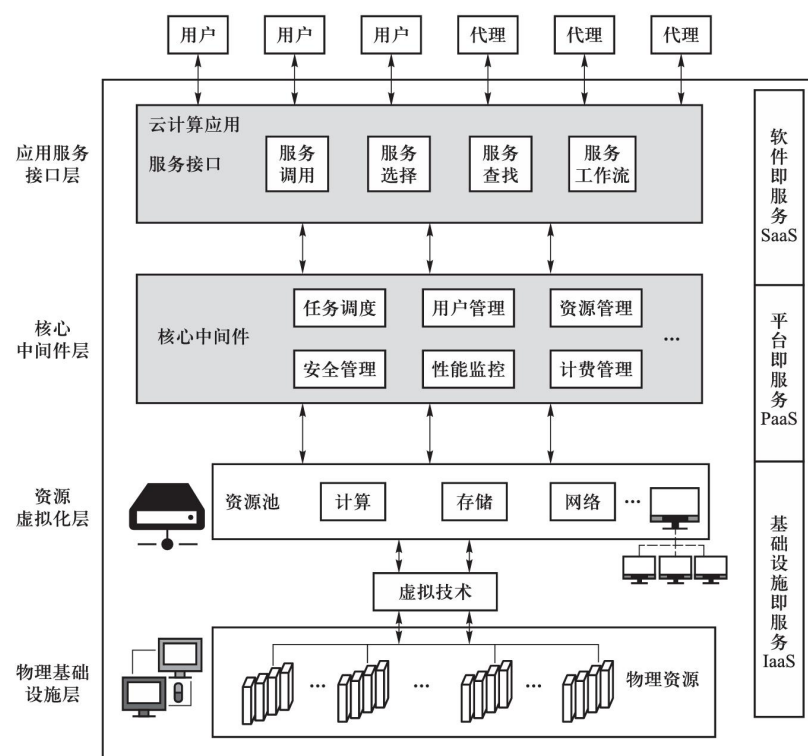
8.4.1 云计算与云原生

- 云计算 (Cloud Computing):
 - 一种按需提供的、可配置的计算资源共享池（包括网络、服务器、存储、应用和服务）
 - 用户可以随时随地、便捷地通过网络访问这些资源
- 核心特征：
 - 按需自助服务 (On-demand Self-service): 用户无需人工干预即可获取资源
 - 广泛的网络接入 (Broad Network Access): 通过标准机制（网络）访问
 - 资源池化 (Resource Pooling): 多租户模型，动态分配物理和虚拟资源
 - 快速弹性 (Rapid Elasticity): 资源可根据需求快速扩展或收缩
 - 可计量服务 (Measured Service): 自动监控和优化资源使用（按使用付费）

8.4.1 云计算与云原生

■ 服务模型：

- IaaS (基础设施即服务)：提供计算、存储、网络等基础资源
- PaaS (平台即服务)：提供开发、运行、管理应用的平台
- SaaS (软件即服务)：提供通过互联网访问的应用程序



8.4.1 云计算与云原生

- 云原生 (Cloud Native)定义：
 - 一种构建和运行应用程序的方法，旨在充分利用云计算模型的优势（弹性、可扩展性、自动化）
- CNCF (云原生计算基金会) 定义：
 - 云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用
- 四大支柱：
 - 容器化 (Containerization)：应用的轻量级封装
 - 微服务 (Microservices)：应用架构的松耦合
 - 动态编排 (Dynamic Orchestration)：自动化部署和管理（如Kubernetes）
 - DevOps：开发与运维的协作文化，持续交付

8.4.2 云计算主要架构

■ 分层架构

- 体系物理资源层：服务器、存储设备、网络设备等硬件基础设施
- 虚拟化层：通过 Hypervisor 等技术将物理资源抽象为虚拟资源（虚拟机），实现资源池化和隔离
- 平台层：提供操作系统、数据库、中间件、运行时环境等，屏蔽底层细节
- 应用层：运行在云平台上的各种用户应用程序（SaaS应用）

■ 架构特点

- 资源抽象：对用户屏蔽物理硬件的复杂性
- 多租户隔离：保证不同用户的数据和计算安全

8.4.2 云原生主要架构

■ 云原生技术栈

- 容器 (Container): 作为标准交付单位, 包含应用及其依赖, 保证环境一致性
- 服务网格 (Service Mesh): 处理服务间通信 (流量管理、安全、监控), 将网络功能下沉到基础设施层
- 不可变基础设施 (Immutable Infrastructure): 不仅是对服务器进行修补, 而是直接替换, 提高部署的可靠性
- 声明式API (Declarative API): 描述系统的期望状态, 由系统自动调整至该状态 (如K8s的YAML配置)

■ 存算分离架构

- 将计算资源 (CPU/内存) 与存储资源 (磁盘) 解耦, 分别独立扩展, 适应大数据和AI场景下的弹性需求

8.4.3 关键技术——虚拟化与容器

- 虚拟化技术 (Virtualization)
 - 通过虚拟机监视器 (Hypervisor) 在物理机上运行多个虚拟机 (VM)
 - 特点：强隔离性，运行完整的操作系统，开销相对较大
- 容器技术 (Containerization)
 - Docker：最流行的容器引擎，利用Linux内核特性 (Cgroups, Namespaces) 实现进程级隔离
 - 优势：
 - 轻量级：共享主机内核，启动秒级
 - 可移植性：“一次构建，到处运行”
 - 高效：资源利用率高

8.4.3 关键技术——容器编排

- 容器编排
 - 随着容器数量增加，需要自动化工具来管理容器的部署、扩缩容和网络
- Kubernetes (K8s)
 - Google开源的容器编排系统，事实上的标准
 - 核心概念：
 - Pod：K8s的最小部署单元，包含一个或多个容器
 - Service：定义一组Pod的访问策略（负载均衡）
 - Deployment：声明式管理应用的副本和更新
 - 功能：服务发现、负载均衡、自动扩缩容、滚动更新、自我修复

8.4.3 关键技术——微服务与服务网格

- 微服务架构 (Microservices)
 - 将单体应用拆分为一组小型、独立部署的服务，服务间通过轻量级机制 (HTTP/REST, gRPC) 通信
 - 优势：独立开发部署、技术异构、故障隔离
- 服务网格 (Service Mesh)
 - 背景：微服务数量增多导致服务治理（熔断、限流、监控）复杂
 - 定义：负责服务间通信的基础设施层
 - Sidecar模式：在每个应用容器旁部署一个代理容器（如Envoy），接管所有网络流量
 - 代表系统：Istio、Linkerd

8.4.3 关键技术——无服务器计算

- 无服务器计算 (Serverless)
 - 开发者只需关注代码（函数），无需管理服务器基础设施
 - FaaS (Function as a Service):
 - 事件驱动：由HTTP请求、定时器或消息队列触发执行
 - 自动扩缩容：根据请求量自动从0扩展到N，再缩容回0
 - 按量付费：按实际运行时间（毫秒级）和内存收费
 - BaaS (Backend as a Service)：直接使用云端提供的后端服务（如数据库、认证）
 - 代表：AWS Lambda, Azure Functions, Google Cloud Functions

8.4.4 代表系统

■ 公有云平台

- Amazon Web Services (AWS): 全球最大的云服务提供商, 提供EC2, S3, Lambda等丰富服务
- Microsoft Azure: 与企业级软件结合紧密, 提供混合云解决方案
- Google Cloud Platform (GCP): 在大数据、AI和容器技术 (K8s) 方面具有优势
- 阿里云 (Alibaba Cloud): 亚太地区领先, 支撑双11等大规模并发场景

■ 云原生开源项目

- Kubernetes: 容器编排
- Prometheus: 监控与报警
- Envoy / Istio: 服务网格
- Helm: K8s包管理工具

第八章、新型分布式系统

01

8.1 分布式图计算

02

8.2 分布式机器学习

03

8.3 联邦计算

04

8.4 云计算与云原生

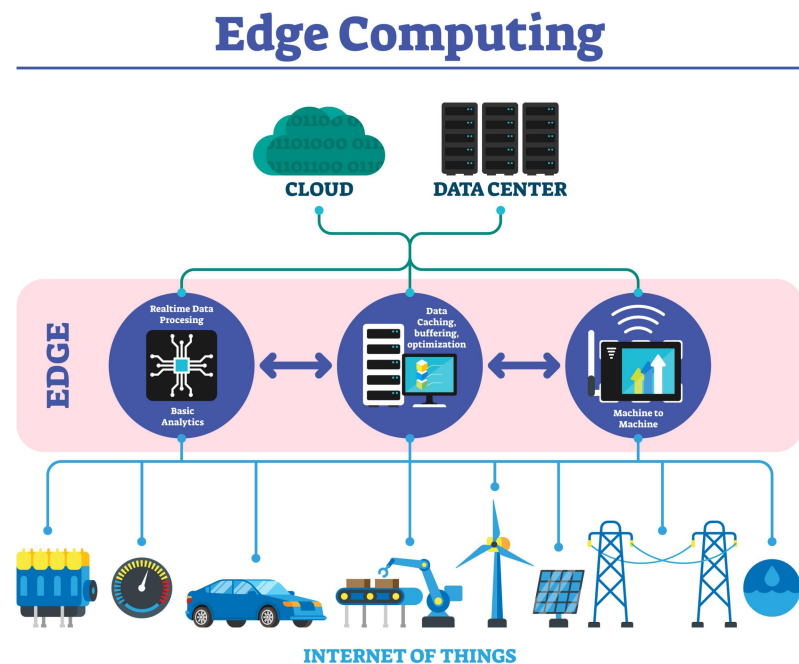
05

8.5 边缘计算



8.5.1 背景与概念

- 核心特征：
 - 资源下沉： 计算/存储资源向数据源头迁移
 - 边缘部署： 建立小型化数据中心，减少传输冗余
- 核心优势：
 - 低延迟： 响应时间从百毫秒级降至毫秒级
 - 省带宽： 流量本地消化，减轻骨干网压力
 - 高安全： 敏感数据本地处理，无需全部上云



8.5.1 边缘计算核心概念

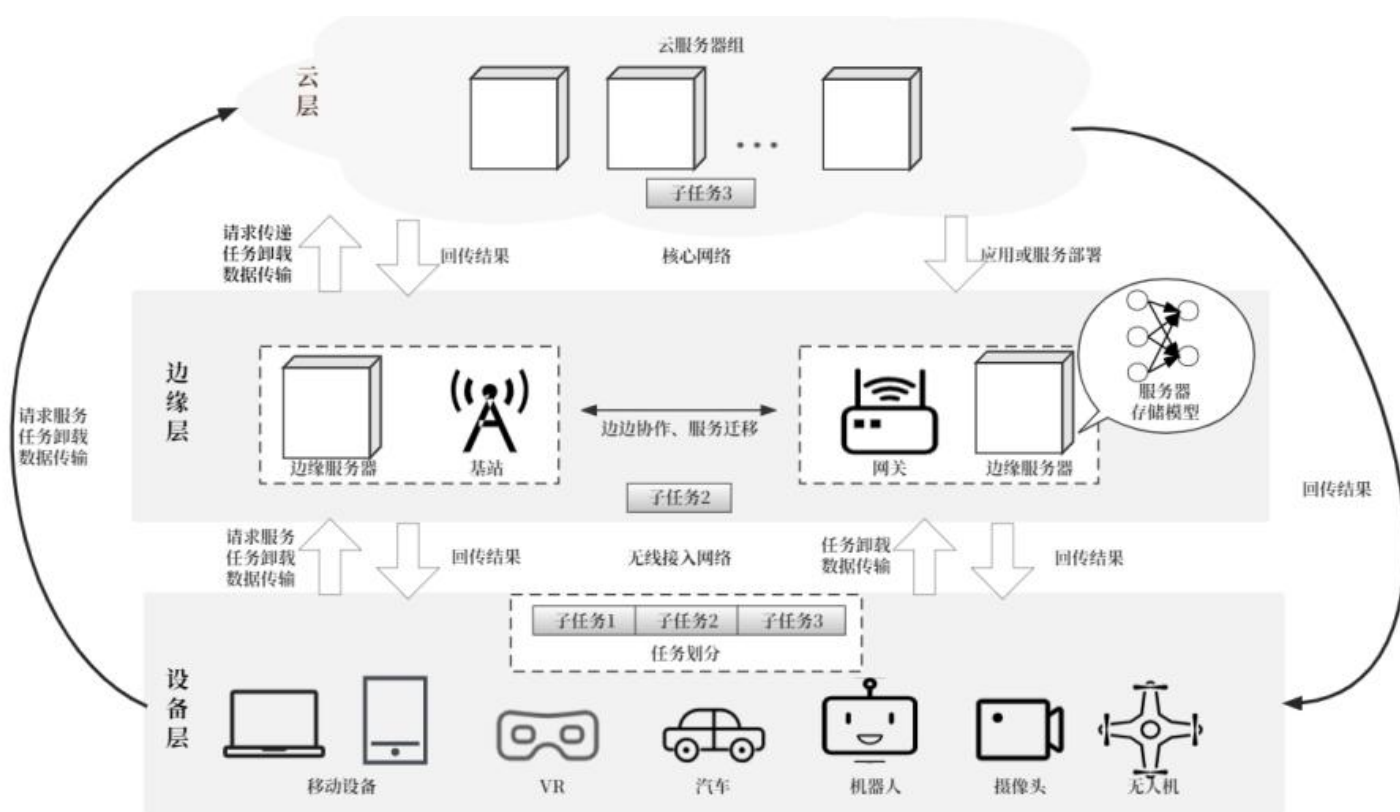
- 1. 基础设施层 (Infrastructure)
 - 边缘设备：数据的源头（传感器、摄像头等），负责数据采集与任务卸载决策
 - 边缘服务器：近场计算节点，负责实时数据处理与分析，降低延迟
 - 边缘网络：支撑近场数据传输与处理的通信纽带
- 2. 逻辑组件层 (Logical Components)
 - 边缘网关：位于中间层，负责数据的预处理、过滤与聚合
 - 边缘计算平台：简化应用开发与部署，提供存储、分发及安全性保障
- 3. 应用与协同层 (Applications & Collaboration)
 - 边缘应用/智能：在边缘侧运行算法，实现实时决策与定制化服务
 - 边云协同：分级处理体系——边缘预处理，云端深度建模与存储

8.5.1 雾计算、移动边缘计算

- 1. 雾计算 (Fog Computing) —— “更接近地面的云”
 - 核心定义：由 Cisco 提出，在云和终端之间引入“雾层”（基站、网关、局域网节点）
 - 功能分工：
 - 雾层：处理延迟敏感、小规模任务，实现任务下移
 - 云中心：负责计算密集型、全局性的任务
 - 关注点：侧重于云与终端之间的基础设施层构建
- 2. 多接入边缘计算 (MEC) —— “5G 时代的边缘交付”
 - 核心定义：原为移动边缘计算（Mobile Edge Computing），现演进为多接入边缘计算
 - 技术特点：
 - 下沉部署：将任务从核心网转移至接入网侧（如 5G 基站）
 - 灵活载体：可部署于固定场所（车站）或移动载体（智能汽车）
 - 核心价值：结合 5G 技术，极大降低端到端传输延迟，实现近端云服务交付

8.5.2 主要架构

- 设备层：作为数据源头感应环境，并在本地或通过计算卸载执行初步任务
- 边缘层：部署在网络边缘，利用近场服务器为设备提供低延迟的实时计算服务
- 云层：凭借强大的中心化资源，处理复杂业务、全局信息并进行持久化存储



8.5.2 设备层

- 层级定义： 由移动设备、智能汽车、AR/VR 及物联网传感器等终端节点构成
- 核心功能：
 - 环境感知： 持续收集多类型数据，生成实时计算任务
 - 初步处理： 具备基础存储与计算能力，支持任务本地执行
- 关键决策：
 - 计算卸载（Offloading）： 根据任务性质、网络及服务器状态，自主选择本地处理或卸载至边缘/云端
- 设计目标： 满足服务质量要求，通过并行处理提升系统响应速度

8.5.2 边缘层

- 核心地位： 位于设备层与云层之间的中间层，是整个三层架构的控制核心
- 硬件组成： 由边缘服务器、智能网关、基站、路由器等基础设施组成
- 关键能力：
 - 无线接入： 为终端设备提供近场接入技术
 - 本地服务： 预存云端应用镜像，直接响应设备请求，提供更快捷的服务
- 任务处理逻辑：
 - 实时处理： 接收终端数据流与子任务，由边缘服务器就近完成分析
 - 协同卸载： 基于负载均衡和体验质量（QoE）考虑，可将复杂任务进一步卸载至云端

8.5.2 云层

- 基础设施： 由高性能计算单元、大容量存储集群等大规模云计算基础设施组成
- 核心优势： 具备系统中最强大的数据处理能力与海量存储空间
- 职能定位：
 - 复杂任务处理： 负责边缘层难以胜任的复杂计算任务与全局信息分析
 - 大数据中心： 作为数据的最终归宿，进行长期的持久化存储
- 协同与控制：
 - 数据下沉与上传： 接收来自设备层和边缘层的数据流及子任务
 - 全局优化控制： 向边缘层下发控制指令，从全局视角调度资源，实现“云-边-端”高效协同

8.5.2 服务放置与迁移

■ 动态服务放置 (Service Placement)

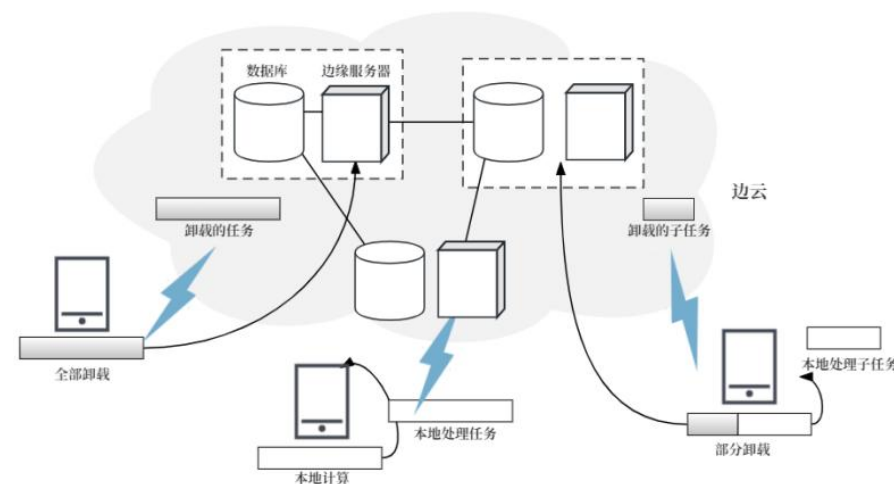
- 定义： 云层根据近期请求模式，统一调整边缘服务器的应用部署
- 目标： 优化应用布局，扩大服务覆盖率，精准满足终端需求
- 视角： 属于全局性的资源调度与重新平衡

■ 服务迁移 (Service Migration)

- 场景： 针对设备移动性（如智能汽车、移动终端）引发的范围切换
- 过程： 将应用从原边缘服务器迁移至离用户更近的新服务器
- 关键挑战：
 - 有状态迁移： 需同步迁移当前任务处理状态及相关业务数据
 - 服务连续性： 确保切换过程中用户感知的服务不中断

8.5.3 关键技术——计算卸载

- 定义： 弥补终端资源不足，通过自主获取网络/节点状态，决策任务的执行位置（本地/边缘/云端）
- 卸载步骤： 1. 任务划分 2. 卸载决策 3. 子任务部署 4. 任务处理 5. 结果回传。
- 卸载类型与任务形态：
 - 完全卸载： 针对原子任务，整体迁移至边缘或云端
 - 部分卸载： 针对复合任务，拆分为子任务进行分布式处理（需考虑执行依赖顺序）
- 优化目标与算法：
 - 目标： 最小化延迟、降低能耗、控制成本、提高系统效能
 - 主流方法： 凸优化、启发式算法、博弈论、机器学习



8.5.3 关键技术——计算卸载优化策略

- 基于启发式算法：
 - 利用改进的蚁群优化算法（RTACO），通过复用仿真结果实现负载均衡下的实时任务分配
- 基于博弈论：
 - 构建非合作或联合合作博弈模型，通过纳什均衡或讨价还价方案实现多用户环境下的资源分配优化
- 基于机器学习：
 - 结合深度强化学习（DDPG/DQN）与经验回放机制，针对信道和任务状态实现自适应的任务卸载与功率调度

8.5.3 关键技术——动态服务放置

- 核心定义： 云端根据用户需求，对边缘服务器上的应用进行自适应配置，确保任务能卸载到合适的节点
- 三个子问题：
 - 应用缓存： 决定哪些应用应从云端下沉至边缘侧（如缓存“热门”请求应用）
 - 资源分配： 在边缘服务器资源限制内，优化多应用的承载布局
 - 请求转发： 若本地未命中应用，则将请求调度至邻近边缘节点或云端
- 解决方案分类：
 - 集中式方案： 由云端或集中协调器决策。具备全局视图，性能接近最优
 - 分布式方案： 边缘服务器根据本地视图自主决策，支持多节点间共享视图与协作
- 优化目标： 综合平衡服务覆盖范围、QoS（服务质量）及资源利用率

8.5.3 关键技术——动态服务放置优化策略

- 基于启发式算法 (Heuristic-based)
 - 离散蝴蝶优化算法 (DBAO): 引入遗传算子与莱维飞行, 解决局部最优问题, 提升 workflow 调度的收敛速度
 - 改进遗传算法 (GASISMEC): 综合考量用户移动路径、干扰及负载, 在多项式时间内求解近似最优的服务选择方案
- 基于图论方法 (Graph Theory-based)
 - 图着色模型 (CSP): 针对密集蜂窝网络, 利用图着色实现并行的吉布斯采样, 达成去中心化的协作放置
 - 图割与最大流 (Graph Cut): 将成本优化问题构建为图模型, 通过最大流算法求解最小切割问题, 实现总成本的最优控制

8.5.3 关键技术——服务迁移

- 核心动因： 应对用户/设备的移动性，防止因脱离边缘覆盖范围而导致服务中断，确保 QoS 连续性
- 与服务放置的区别：
 - 服务迁移： 单个设备移动触发，涉及任务状态（Context）的实时转移
 - 服务放置： 周期性或需求变化触发，侧重于应用镜像（Image）的缓存配置
- 迁移分类：
 - 无状态迁移： 仅需在目标节点重新部署应用
 - 有状态迁移： 需同步转移运行状态（中间数据），在目标节点无缝恢复执行
- 迁移决策：
 - 迁移策略： 选择最优目标服务器进行状态交接
 - 不迁移策略： 若频繁移动导致迁移成本过高，则维持原位运行，通过核心网转发数据

8.5.4 代表系统

- EdgeX Foundry（工业互联标准化）
 - 工业物联网标准化、互操作生态系统、统一通用API、统一通用API
- Azure IoT Edge（云原生功能下沉）
 - 云原生逻辑下沉、模块化容器实例、云边编程模型统一、低延迟应用迁移
- CORD（电信中心机房改造）
 - 电信机房数据中心化、SDN/NFV/云技术集成、网络功能虚拟化、移动/住宅/企业多场景支持
- Apache Edgent（轻量级边缘流处理）
 - 轻量级运行时、边缘流式数据分析、拓扑式处理模型、本地实时反馈

本章小结

系统类型	核心设计目标	关键技术亮点
分布式图计算	高效处理非结构化图数据	顶点/边分割、BSP/GAS 模型
分布式机器学习	支持大规模参数与并行训练	参数服务器 (PS)、All-Reduce 架构
联邦学习	隐私保护下的数据联合训练	安全多方计算、同态加密、本地迭代
云计算	资源的按需分配与极致扩展	虚拟化、IaaS/PaaS/SaaS 模式
边缘计算	满足实时性与带宽优化	计算卸载、服务放置、云边协同

第八章 完