大模型-AI集群 (存)

大模型训练の存储优化

ZOMI

大模型业务全流程

数据 & 模型算法

2. 数据处理

- 开源数据
- 数据预处理
- 向量数据库

3. 模型算法

- LLM模型架构
- 多模态一切皆Tokens

模型训练 & 微调

4. 模型训练

- 混合精度
- 梯度检查
- 梯度累积
- …

5. 分布式并行

训练集群稳定性

6. 模型微调

- 全参微调
- 低参微调
- 指令微调

模型验证 & 推理部署

7. 模型验证

- 下游任务
- 测评标准
- bench mark

8. 推理与智能体

- 量化压缩
- 推理加速
- 9. Agent 智能体

集群算力准备

1. AI 集群建设

- 计算、存储、网络
- AI 集群机房建设
- AI 集群上线与运维

关于本内容

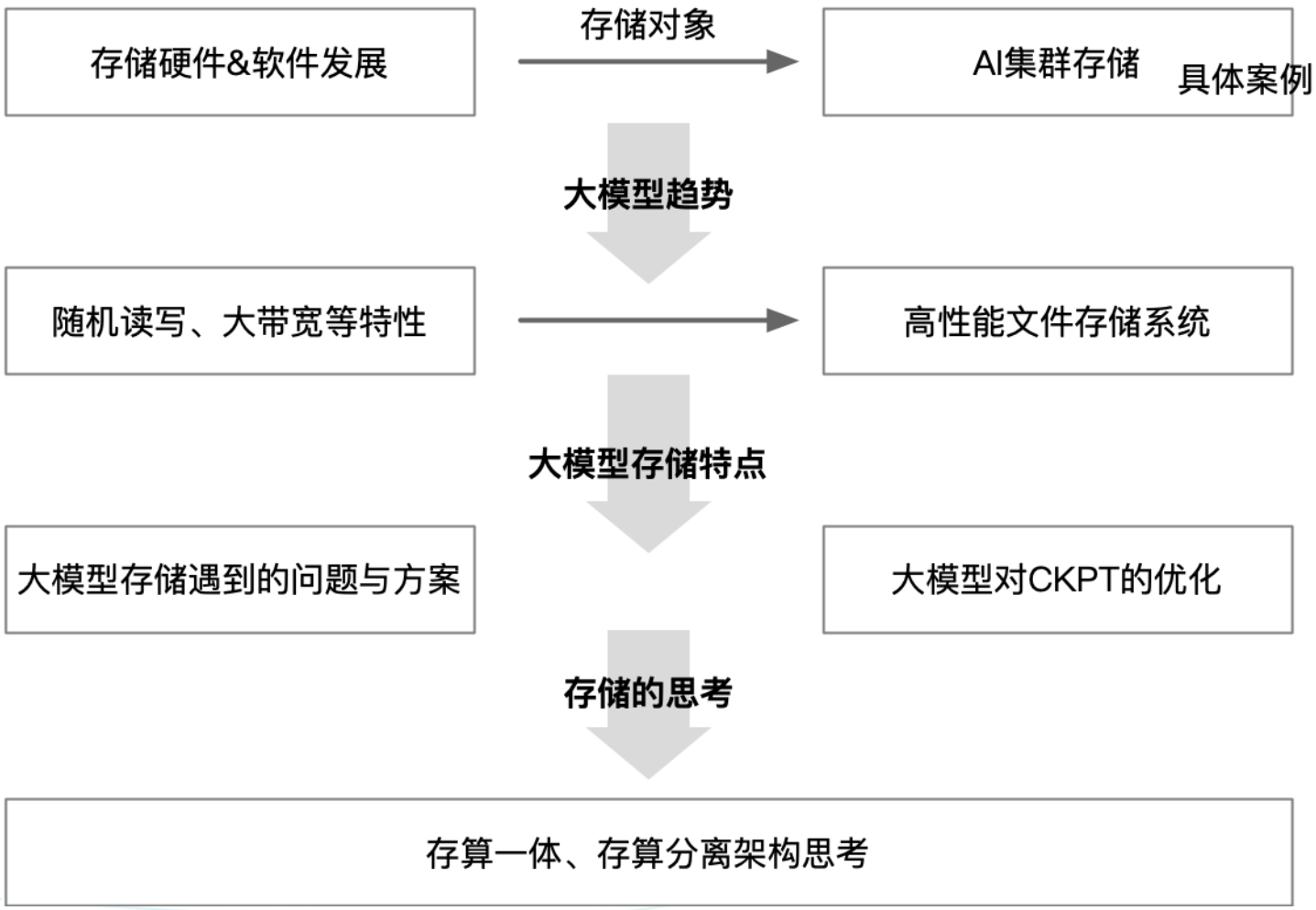
1. 内容背景

- AI 集群 + 大模型

2. 具体内容

- **数据存储现状和场景：**存储软件类型、存储硬件类型的发展
- **大模型对存储的挑战：**存储性能指标、存储遇到大模型挑战与新机会点
- **大模型训练CKPT优化：**大模型训练过程、CKPT过程分解、CKPT优化
- **大模型时代对存储的思考：**什么样的存储架构才是AI大模型时代的选择？

关于本内容


存储硬件&软件发展
存储对象
AI集群存储
具体案例
大模型趋势
随机读写、大带宽等特性
高性能文件存储系统
大模型存储特点
大模型存储遇到的问题与方案
大模型对CKPT的优化
存储的思考
存算一体、存算分离架构思考

大模型 & 存储技术架构

存储技术架构

大模型&应用

- LLAMA等LLM
- 多模态大模型
- AI Agent智能体

大模型训练

- 多级存储
- CheckPoint
- 训练数据

存储对象

存储协议

- 文件存储
  - NFS
  - HDFS
- 块存储
  - NVMe
  - SCSI
- 对象存储
  - Tensor
  - Vector
- 其他
  - Table

连接方式

- SAN
- NAS
- DAS

存储架构

- 集中式存储
- 分布式存储

硬件介质

- 磁带
- 光盘
- 机械硬盘
- 固态硬盘

HUAWEI LOGO 变化

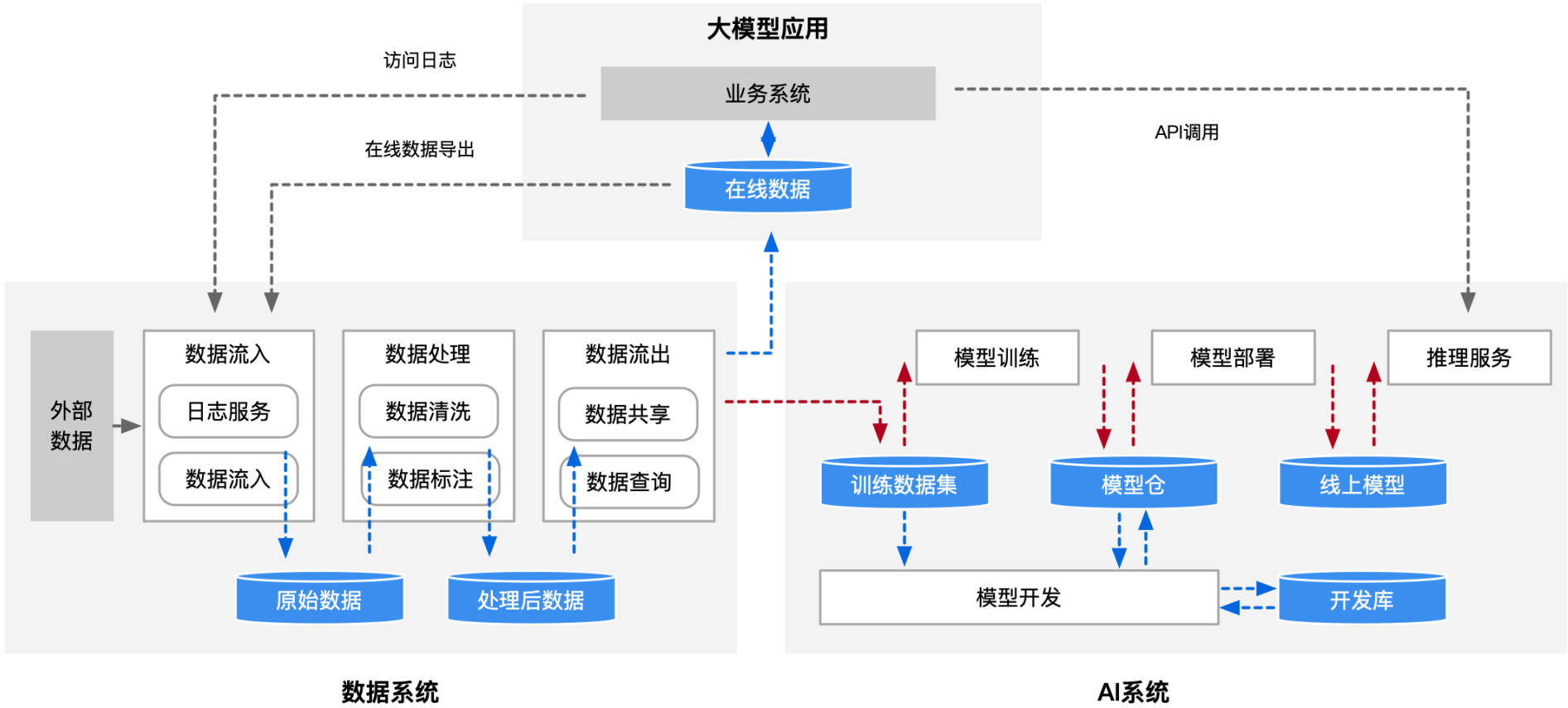
Thought

1. 对象存储（数据湖）能够很好存储公开数据集、训练数据、模型结果等数据，但是高吞吐能力给小文件读取带来的提升有限。面对海量小文件读取，有哪些解决思路呢？
2. 训练场景下，几千台计算节点会同时读取一批小数据集，如何缩短数据集的加载时长？

1. 训练数据存储优化

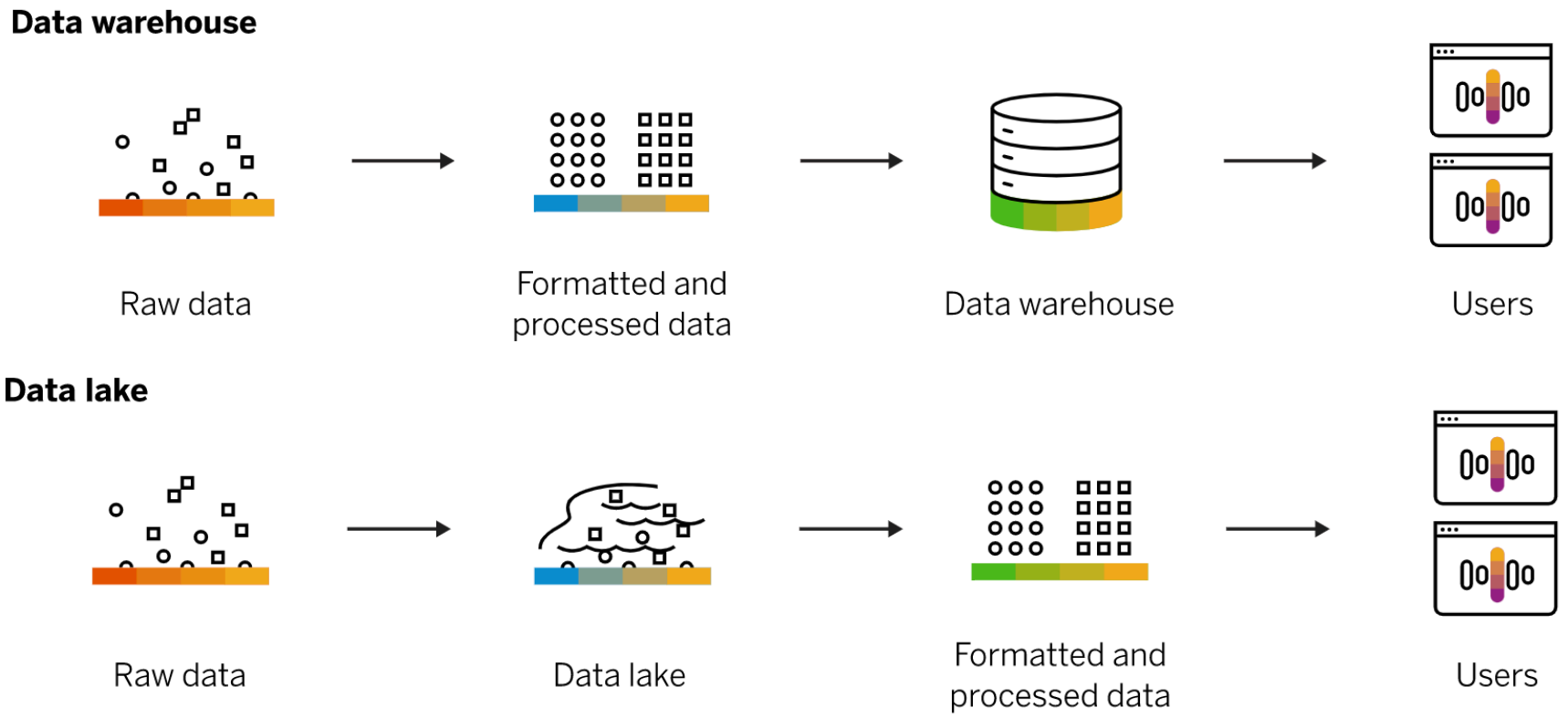
存储的数据流动

- 1）大模型训练过程语料数据持续更新，2）训练数据在不同数据预处理流程中频繁流动。有必要针对非结构化和半结构化数据提供专用存储系统，例如数据湖。



数据湖

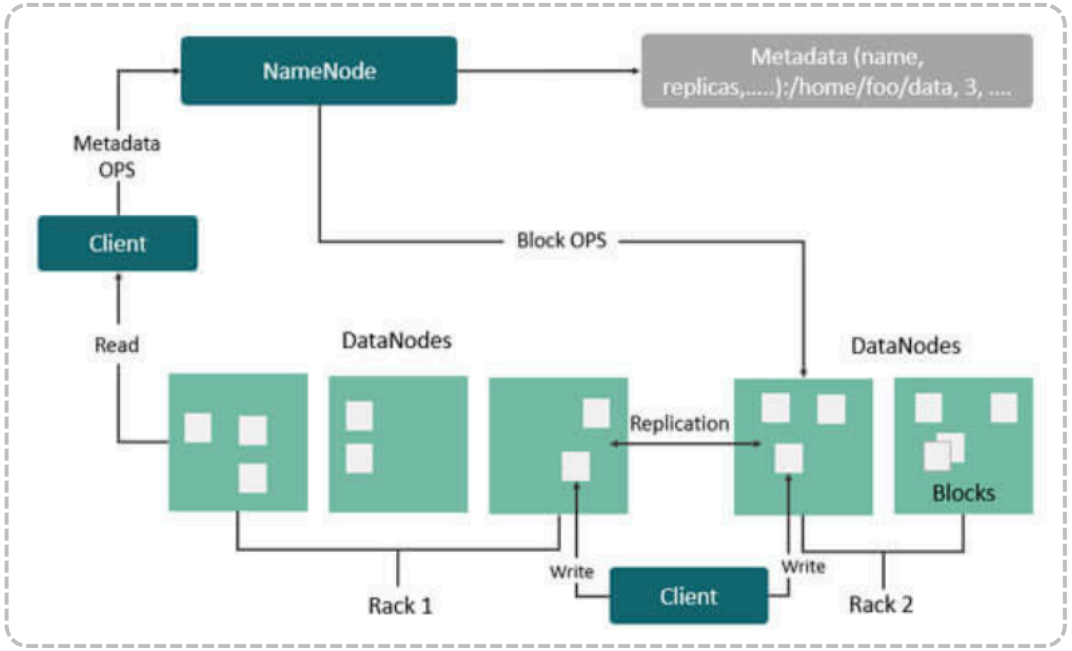
- 集中式存储库，允许以任意规模存储结构化和非结构化数据。可以按原存储数据，运行不同类型的分析处理算法。



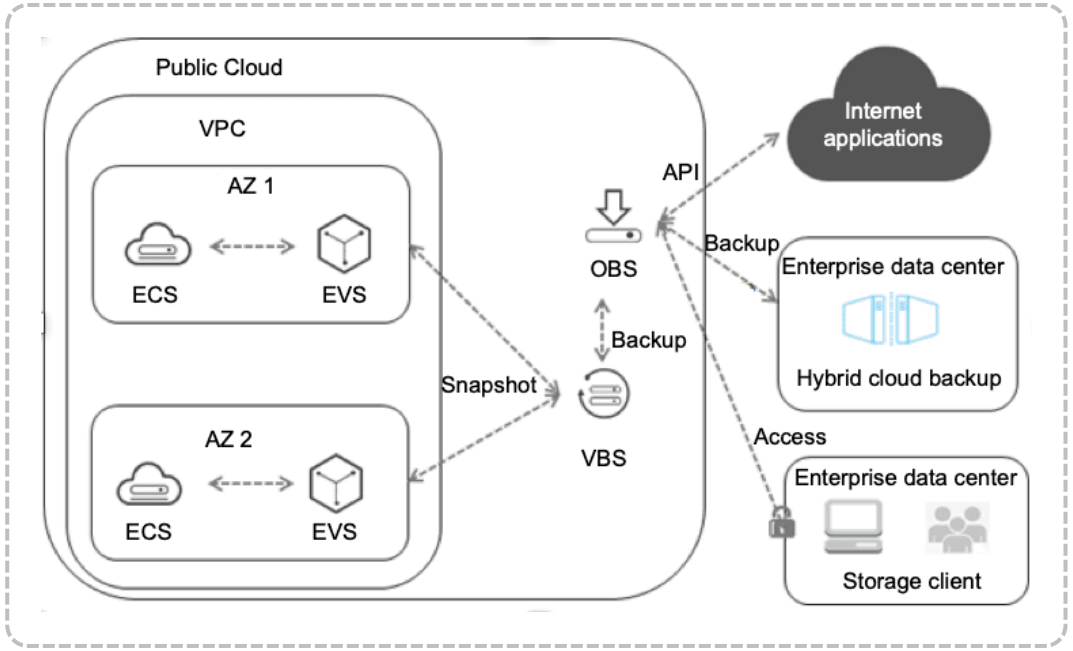
数据湖的存储类型：文件系统 vs 对象存储

1. **扩展能力**：HDFS 采用集中式数据，规模受限；对象存储采用分布式数据可水平扩展，单集群可支持万亿文件、EB 级规模。

文件存储（HDFS）



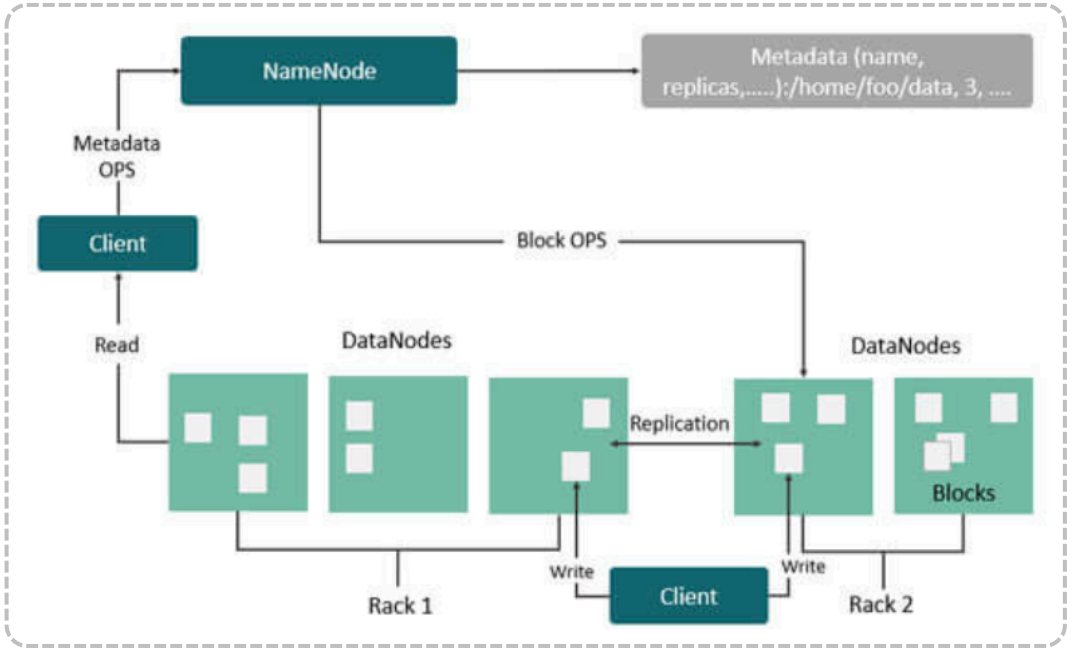
对象存储



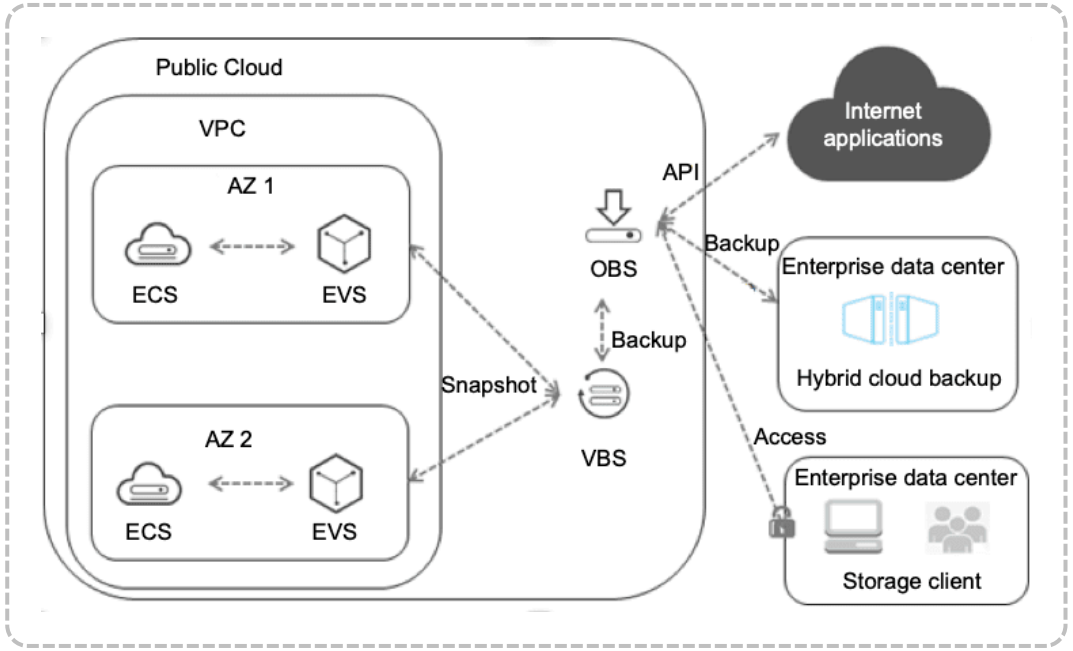
数据湖的存储类型：文件系统 vs 对象存储

1. **存储成本**：HDFS 诞生于存算一体的时代；对象存储天然面向存算分离设计，结合 EC 编码、分级存储等丰富的能力，可以实现大规模数据长期保存的更优成本。

文件存储（HDFS）



对象存储

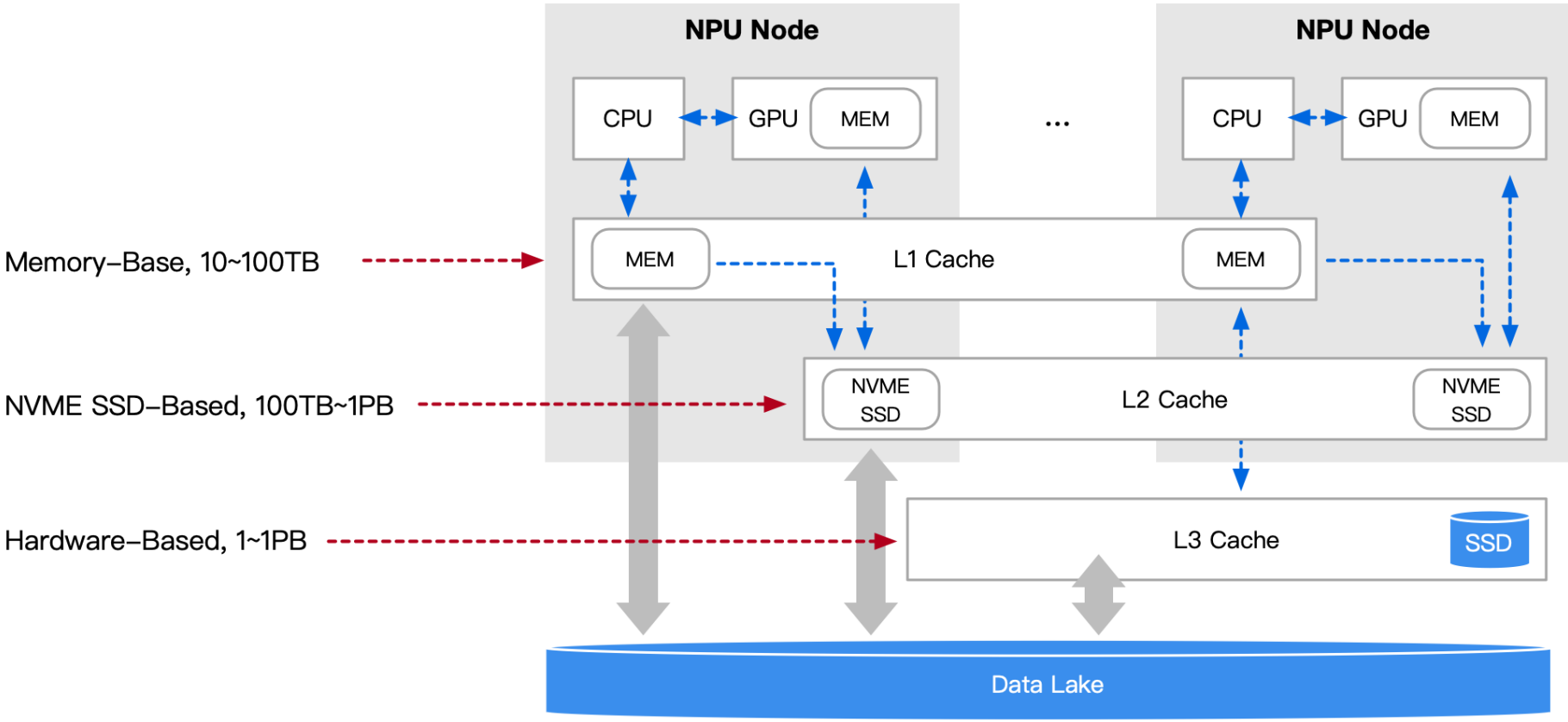


数据湖的存储类型：文件系统 vs 对象存储

	文件系统	对象存储
扩展性	• NameNode 单点，层级命名空间 • 限制单集群上限10亿文件，PB级别容量	• 无架构扩展瓶颈 • 单集群可支持万亿规模文件，EB级别容量
成本	• 3X 副本，存算一体架构，存储成本和计算成本同步增长	• EC 编码，1.5X 副本容灾 • 存算分离架构，存储成本个存储容量相关
可用性	• 可容忍2个副本损坏	• 取决于 EC 校验块数量，18+6 可容忍 6 个块故障
吞吐	• 集群规模相关	• 和集群规模强相关
生态	• 传统大数据领域接受度高	• 存算分离架构流行，AI 领域接受度高

优化方案：统一对象存储到数据湖

- 将公开数据集、训练数据、模型结果统一存储到对象存储中（数据湖），实现不同形态的数据统一存储和高效流转。



2. 训练流程

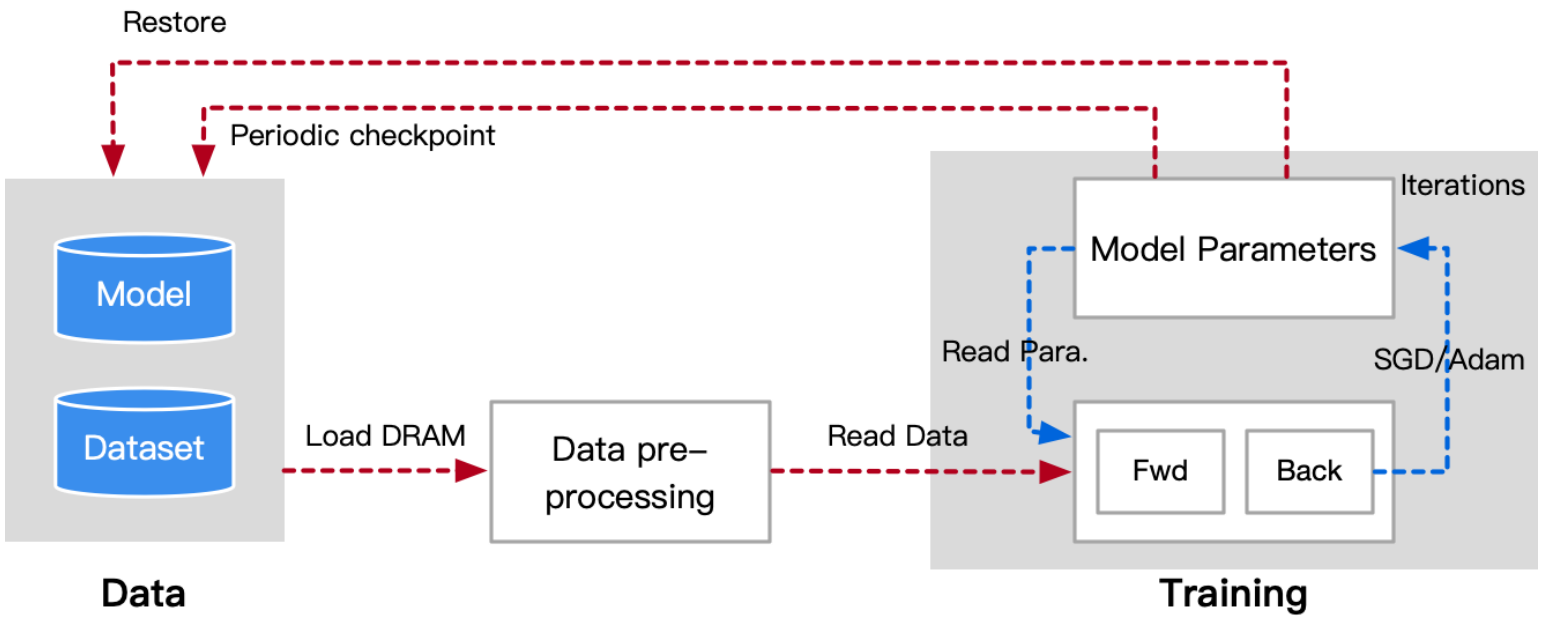
存储优化

训练流程分析

- 每轮 epoch 需要对数据集进行 Shuffle，然后将数据划分为 N batch，每次读取 1 batch 进行一次训练迭代。周期性保存 Checkpoint 用于故障快速恢复。

```
# trainging Epoch
for each in epoch:
    # List files in the datasets
    # pre-processing of datasets
    for mbs in batch:
        # Read files of the batch
        # Training
        pass

    checkpoint.save()
```



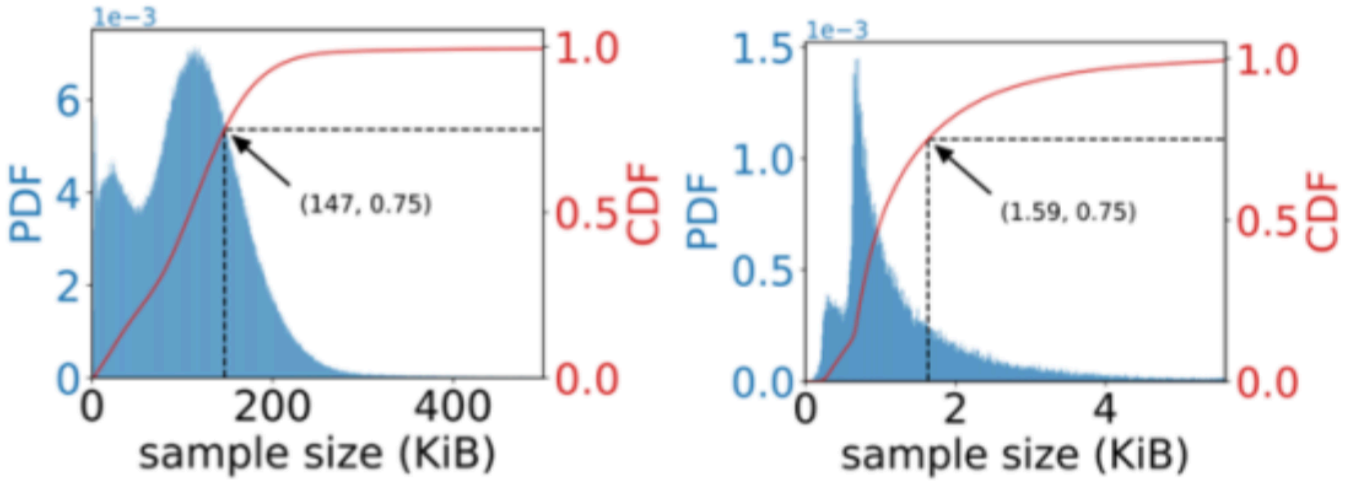
优化分析 I：数据清洗过程

Shuffle 和 Read 数据读取对存储的操作

- Shuffle：元数据操作，主要是针对数据 List
- Read：元数据操作 & 数据本身的操作

对于大量小文件，延时和吞吐是性能关键

- 小文件：文件越小，元数据操作耗时占比越大
- 指标：延时、吞吐是降低数据清洗过程关键



(a) ImageNet dataset　　(b) IMDB movie review dataset

优化分析 II：小文件对象存储

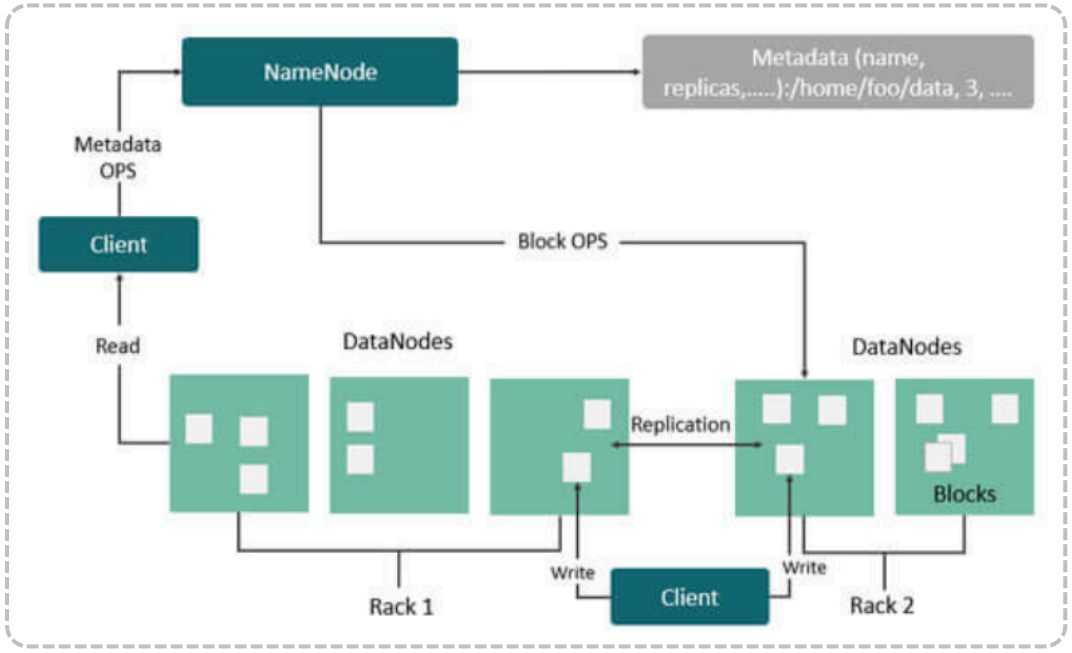
元数据性能影响

- 对象存储采用 key-value 方式维护元数据，很好解决小文件规模扩展问题

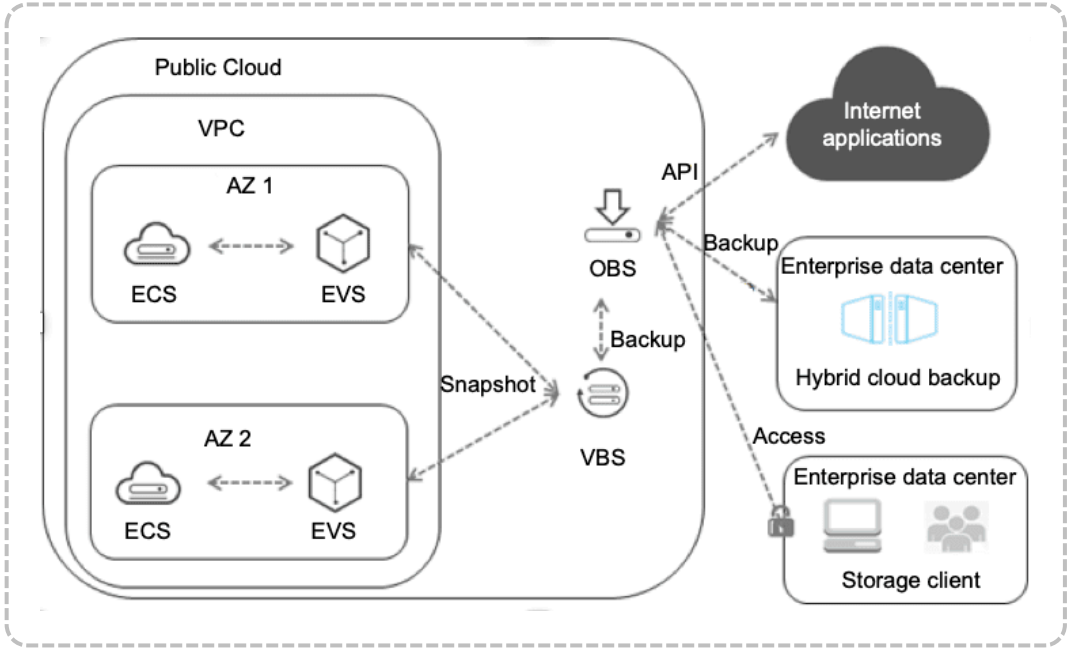
操作 & 协议路径的影响

- 数据清洗 Shuffle 用到的 List 操作性能延时偏高
- 对象存储协议访问路径长，小文件频繁读取延时大

文件存储（HDFS）



对象存储



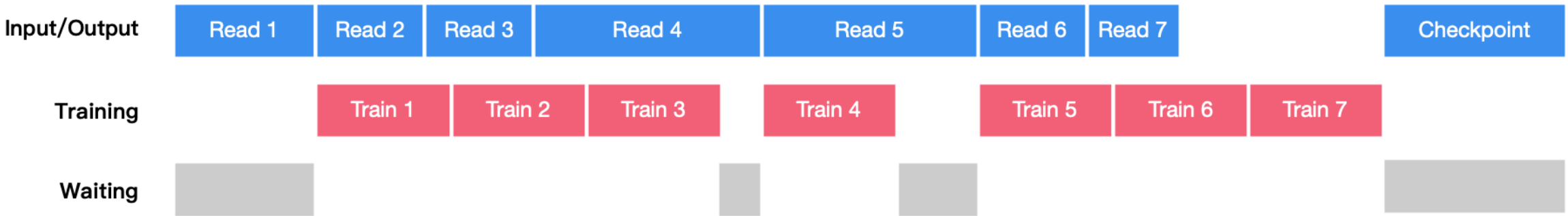
优化分析 III：存储数据加载

- 训练执行与 IO 时序关系：每轮 Epoch 耗时由数据 Shuffle 时间、计算执行时间、数据读取等待。

$$P = T_{training} / T_{epoch}$$

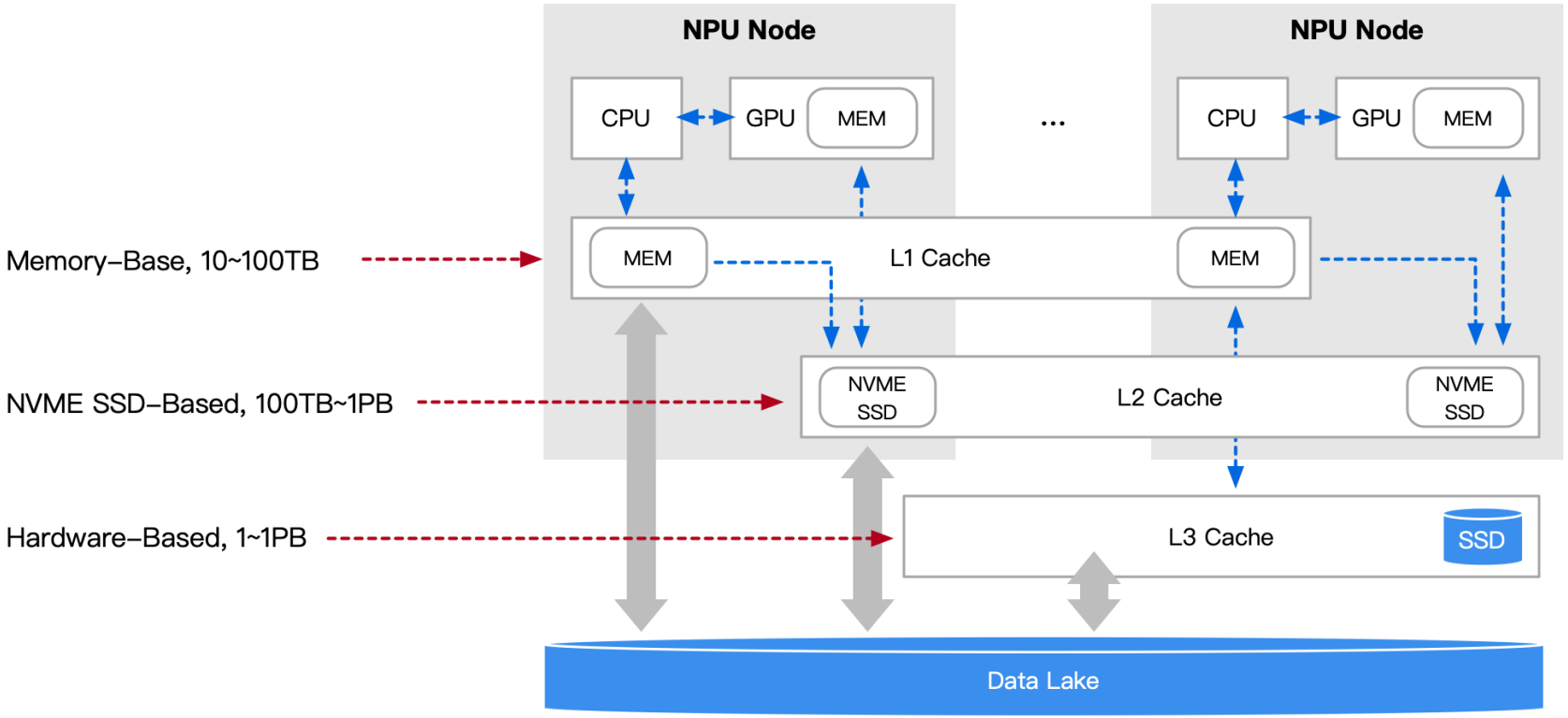
$$P = T_{training} / (T_{wait} + T_{training})$$

$$P = T_{training} \Big/ (T_{shuffle} + \sum T_{wait\ read} + \sum T_{checkpoint} + T_{training})$$



优化方案：多层数据缓存

- 提供多层缓存加速，分层次提供远端、近端的数据读取性能。即按需将热数据缓存到 GPU 内存和 Local 本地盘中，利用数据本地性提供高性能访问。e.g.，训练先将 Checkpoint 写到性能相对容易保证的本地存储，再向远端对象存储服务器（数据湖）上传。



2 优化方案：训练数据加载

- 尽量提高训练效率 and/or 利用率，减少 NPU 空闲，主要优化集中在：
  1. 优化数据清洗过程，数据搬运和处理与计算重叠；
  2. 优化读取过程，让每 Epoch 读取数据耗时小于计算耗时，使得 I/O 时间被计算时间隐藏；
  3. 优化 Checkpoint 保存和加载过程，缩短 Checkpoint 耗时，减少训练中断时间；

$$P = T_{training} \Big/ (T_{shuffle} + \sum T_{wait\ read} + \sum T_{checkpoint} + T_{training})$$

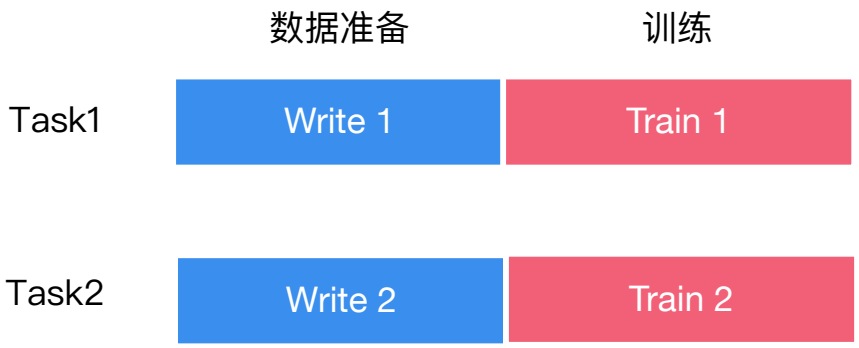
- $T_{shuffle}$：将占比控制在 Epoch 总时长的较小比例
- $T_{checkpoint}$：让每 Epoch 读数据小于计算耗时，读取被计算掩盖
- $T_{wait\ read}$：缩短读取耗时，减少训练等待时间

3 优化方案：调度优化减少数据等待

- 将数据集作为抽象实体统一管理
- 让非竞争性算力资源流水线化，提升计算利用率

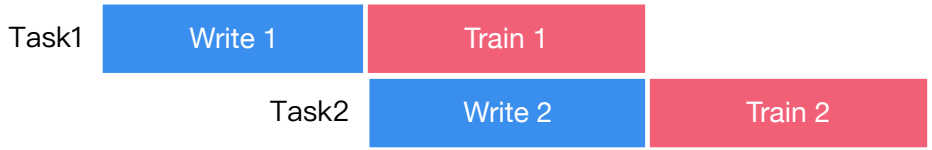
Task1 和 Task2 竞争同一计算资源

从存储对象（数据湖）读取数据



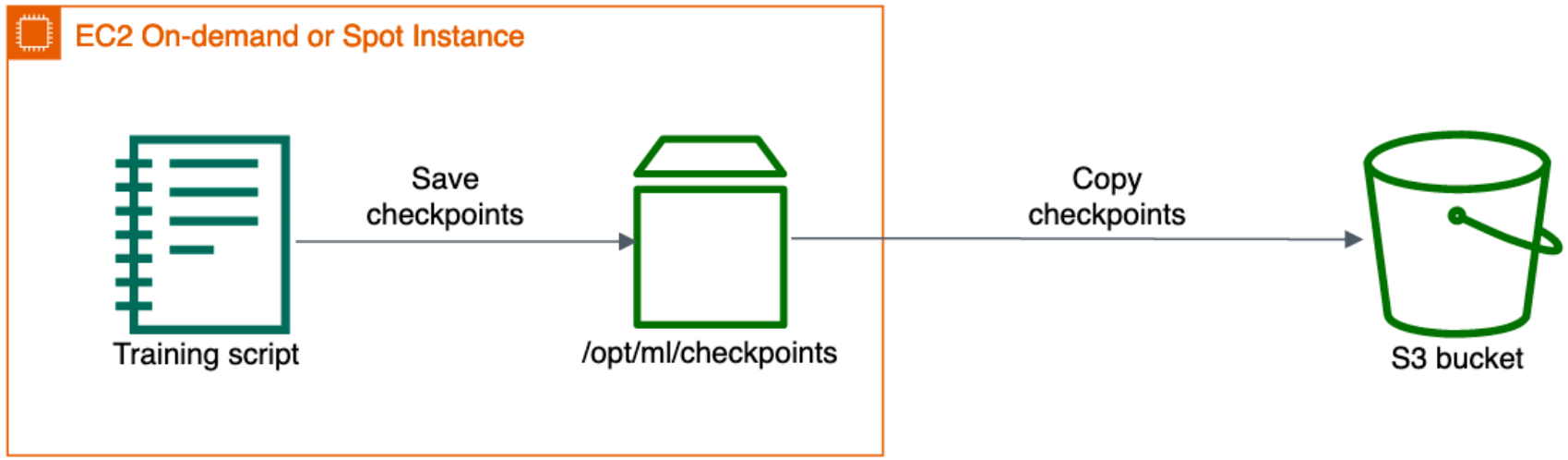
调度策略 1：Task1 和 Task2 作为整体调度

调度策略 2：Task1 和 Task2 流水线调度



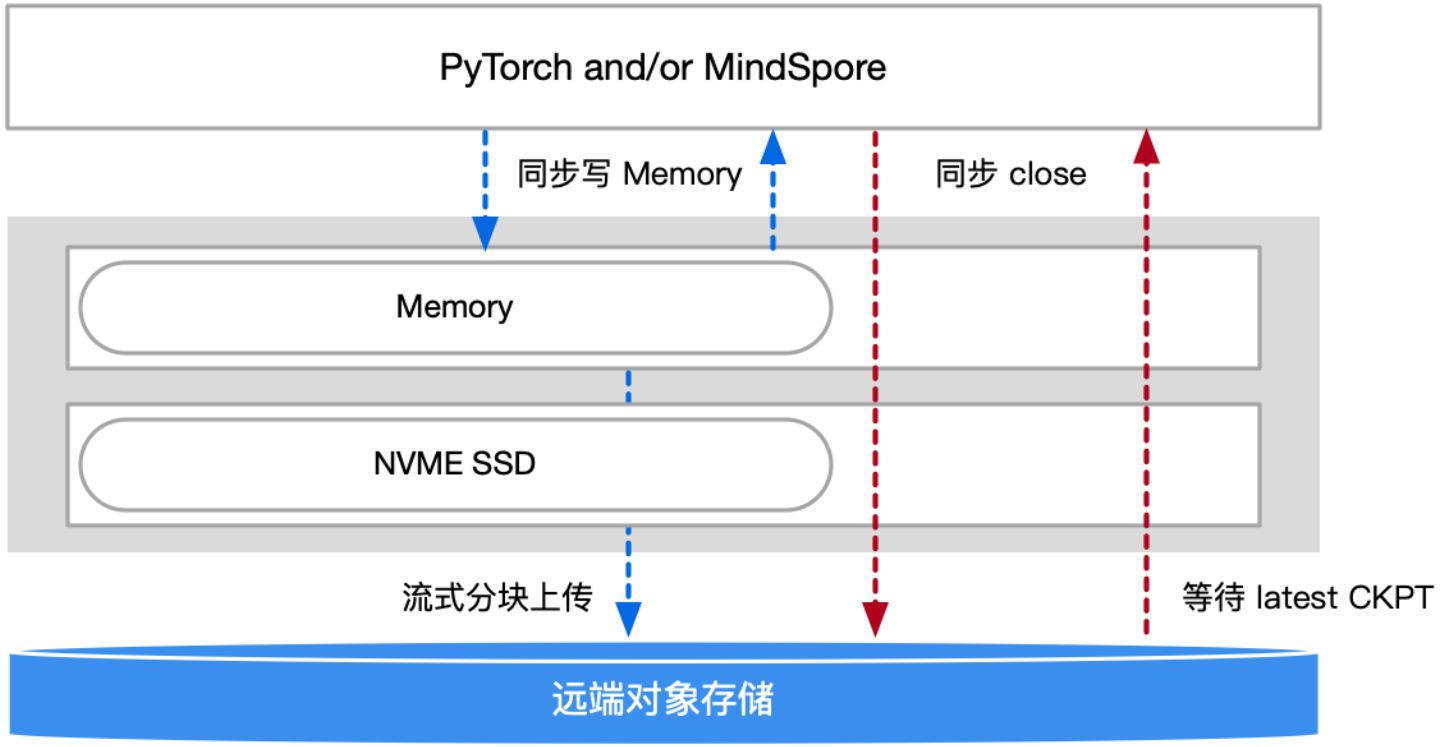
4 同步的存储加速

- 与训练数据以小文件为主不同，大模型单个节点 checkpoint > 100GB。多个训练节点同时写，需要恢复时又同时读，对存储提出了很高的吞吐要求。
- Checkpoint 保存和加载期间整个训练是中断的，因此提高吞吐，将 Checkpoint 耗时控制在尽量小的占比，对于减少训练等待、提高计算资源利用率非常重要。



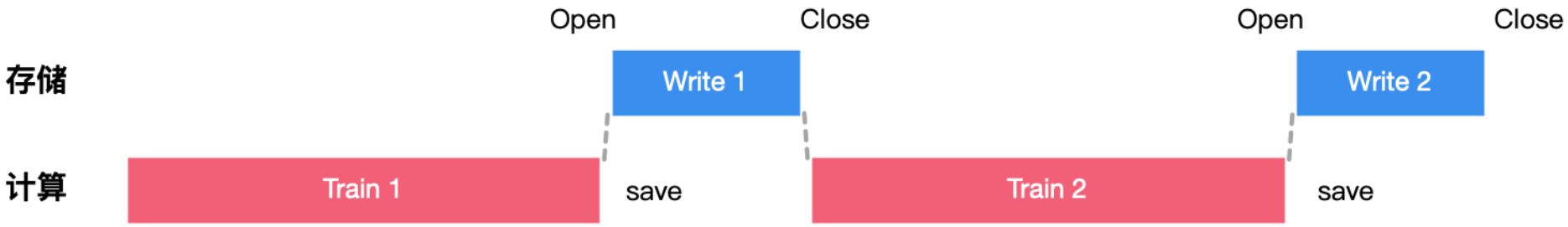
4 同步的存储加速

- Checkpoint 直接写入 CPU 内存或 NVME SSD， 并采用流式 & 分块上传的方式，无需等 Checkpoint 全部写完就开始向数据湖上传。



4 同步的存储加速

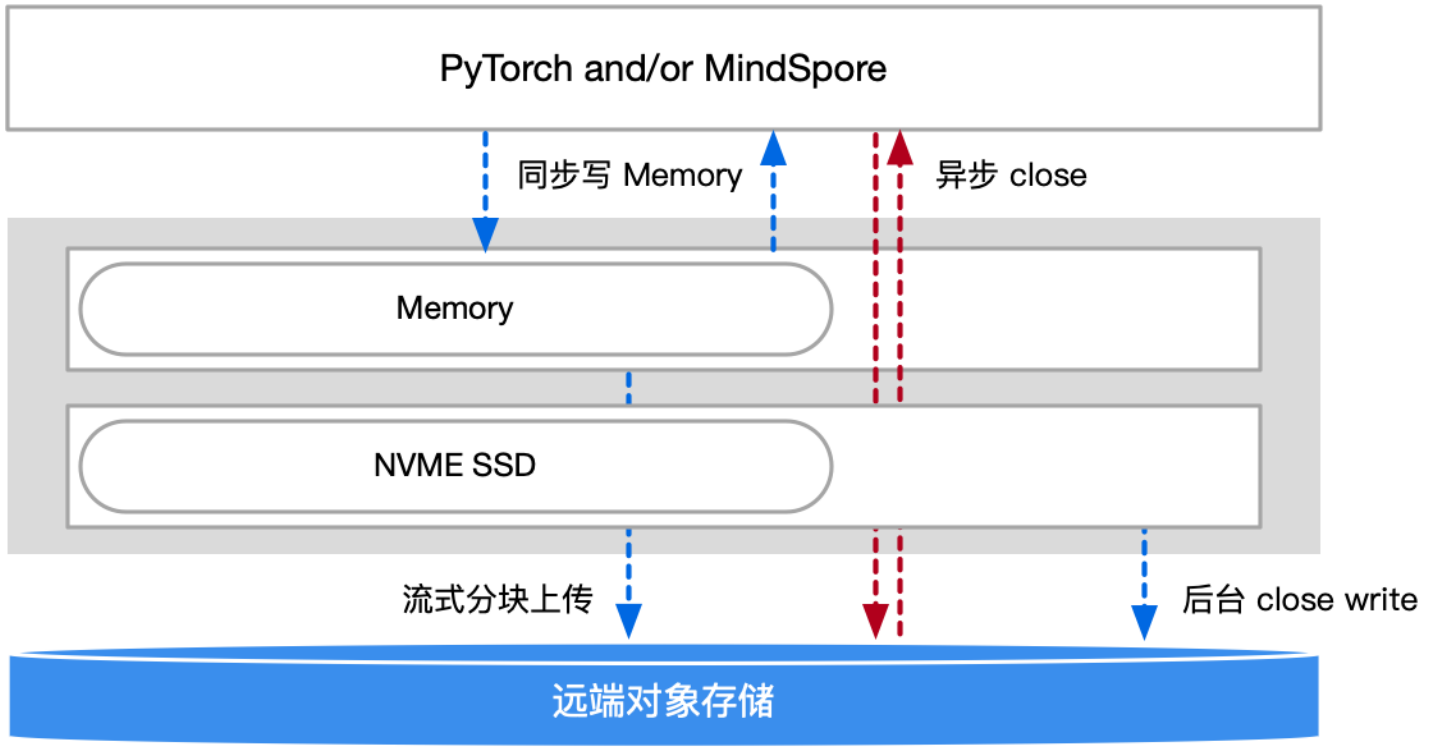
- Checkpoint 直接写入 CPU 内存或 NVME SSD， 并采用流式 & 分块上传的方式，无需等 Checkpoint 全部写完就开始向数据湖上传。
- **缺点**：需要保证 close 时数据已经完整写入对象存储，吞吐瓶颈仍然可能出现在上传带宽限制。



5 异步 Checkpoint 保存 & 加载

- 训练过程 Checkpoint 会周期性保存，发生故障恢复不是常态。

1. 异步写来突破上传对象存储的带宽限制。训练程序不用等待数据上传完成，即可恢复训练。其耗时取决于 NVME SSD 写入能力，极大缩短 Checkpoint 写入时间。
2. 对于 Latest Checkpoint 采用异步写的同时，让其驻留在 CPU 内存，当训练需要恢复时直接读取，解决 Checkpoint 快速加载问题。



小结&思考

小结

1. 通过对象存储提供的数据湖，解决训练数据的存储问题。

2. 训练流程中分析E2E流程，提出多层数据缓存、训练数据异步加载、调度优化、同步存储加速和异步 Checkpoint 保存 & 加载等新的优化手段。

思考

1. 数据存储的优化手段比个人 PC 优化手段丰富有效，但是带来的代价是什么呢？e.g.，例如华为云的数据存储 OBS 桶？昇腾的 MindX 组件？

2. 大模型训练的时候，模型太大需要切分， 因此 Checkpoint 是根据模型并行（MP）、数据并行（DP）来切分成不同的块，因此需要考虑哪些情况进行保存和加载？e.g.，某个节点损坏了怎么办？

Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

Bring AI System to every person, home and organization for a fully connected, intelligent world.

ZOMI

Course chenzomi12.github.io

GitHub github.com/chenzomi12/DeepLearningSystem