日志+检查点存储助力大模型训练故障高效恢复

刘晓宇

liuxy@zhejianglab.org

2024年11月8日

之江实验室 ZHEJIANG LAB

汇报提纲

- 1. 大模型检查点机制简介
- 2. 研究现状
- 3. 创新实践：TranLogs
- 4. 总结

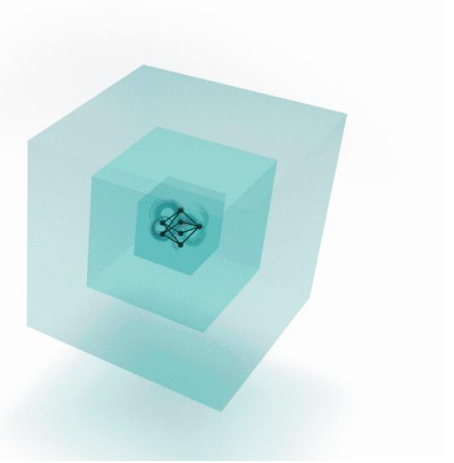
之江实验室 ZHEJIANG LAB

1. 大模型检查点机制简介

1.1 大模型训练

- 大模型训练的复杂性
  - 模型架构复杂：层数深，参数量巨大，训练过程既需大量算力，还需高效的内存管理
  - 计算资源需求：计算资源消耗量巨大（GPT-3，数千NVIDIA V100）
  - 分布式训练：涉及复杂的并行化算法、通信优化、负载均衡等技术
- 深度学习训练过程中故障频发

据OPT-175B的训练报告显示，在训练过程中，由于各种故障造成的训练失败，损失了约**178000**个GPU小时。



1.1 大模型训练

故障案例：大模型训练中的常见问题

模型方面

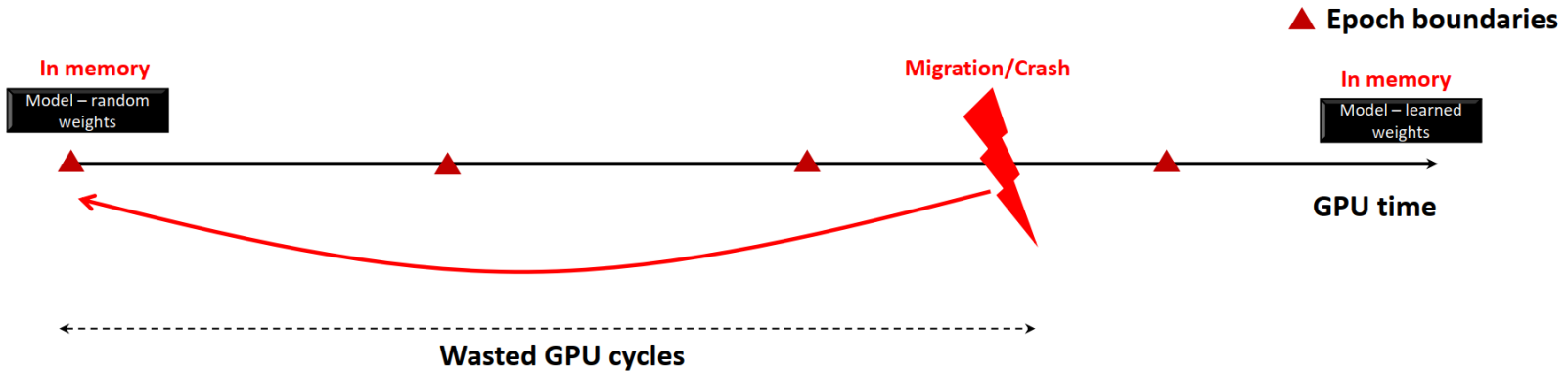
- 梯度消失与梯度爆炸
- 过拟合问题
- 训练不稳定和学习率问题

其他方面

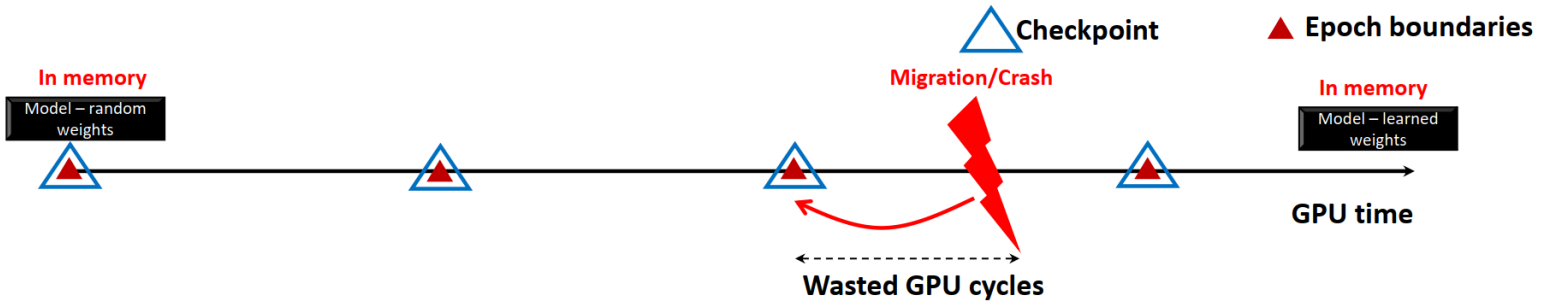
- 内存溢出
- 分布式训练中的同步问题
- GPU/TPU硬件故障
- 网络延迟和通信故障

1.2 大模型检查点机制

- 训练过程中断会清除迄今已学习到的模型参数



- 检查点机制



1.2 大模型检查点机制

检查点机制是大规模深度学习模型训练过程中用于**保存中间模型状态**的重要策略。它通过定期保存模型参数、优化器状态和其他关键信息，在训练中途或结束时可用于**恢复模型**，减少因训练故障导致的时间和资源损失。

检查点内容

- 模型参数
- 优化器状态
- 训练元数据
- 其他信息

重要性

- 容错性与训练恢复
- 节省时间与资源
- 调试与分析
- 长期实验管理

挑战与优化

- 定期检查点的稳态开销
- 固定检查点频率
- 恢复损失

通过合理地设计和优化检查点策略，可以**有效应对**大模型训练中的故障，节省大量的计算资源和时间。

之江实验室 ZHEJIANG LAB

2. 研究现状

2.1 减少稳态开销（Gemini，SOSP 23）

- Motivation
  - 现有检查点解决方案依赖于低带宽的远程存储，恢复速度较慢；
  - 利用本地CPU内存来加速检查点的存储和训练故障恢复。
- 解决问题：1) 如何最大化从CPU内存恢复的成功率；2) 如何将检查点通信与模型训练的通信进行调度，以最小化对训练吞吐量的干扰。
- 关键技术点
  - 检查点的存放策略；
  - 流量调度算法。
- 实现方案：通过分层存储架构、内存检查点存储、优化的检查点放置策略和流量调度算法，确保在大规模深度学习分布式训练中的快速故障恢复，同时不影响训练吞吐量。

2.1 减少稳态开销（Check-N-Run，NSDI 22）

- Motivation：保存和恢复检查点对存储带宽和容量提出了很高的要求。
- 思路
  - 每次检查点迭代只更新模型的部分内容；
  - 压缩检查点文件大小。
- 关键技术点
  - 差异化检查点；
  - 量化技术；
  - 解耦检查点与训练过程。
- 实现方案：1) 跟踪模型状态；2) 量化；3) 分布式存储。

2.1 减少稳态开销（ByteCheckpoint，2024）

- Motivation：1）额外I/O开销；2）Checkpoint重新切分困难；3）不同训练框架Checkpoint模块割裂。
- 思路
  - 元数据/张量数据分离：不同训练框架中的模型以及优化器的张量切片存储在storage文件中，元信息存储到全局唯一的metadata文件中；
  - 异步张量合并（Asynchronous Tensor Merging）技术：针对不同训练框架运行时的不规则张量切分问题，采用异步P2P通信，分配到不同进程上进行。
- 关键技术点：I/O性能优化技术
  - Checkpoint存储优化：1）流水线执行，2）避免内存重复分配，3）负载均衡；
  - Checkpoint读取优化——零冗余加载。

2.2 动态调整检查点频率（CheckFreq，FAST 21）

- Motivation
  - 固定周期的检查点机制只在每个训练周期结束时保存模型状态，丢失历史计算较多；
  - 利用频繁的、精细粒度的检查点机制来减少中断带来的计算损失。
- 思路：在保持低开销的同时提高检查点的频率。
- 关键技术点
  - 系统化在线剖析；
  - 两阶段检查点机制；
  - 可恢复数据迭代器。
- 实现方案：确定检查点频率，并行处理检查点，并进行自适应调整。

2.3 降低恢复损失（JIT，EuroSys 24）

- Motivation
  - 关键的GPU状态仅在可跟踪的短时间间隔内进行更新；
  - 具有相同状态的多个副本的可用性。
- 思路：只在发生故障后才进行及时检查点。
- 关键技术点
  - Domain-aware Device API Interception.
- 两种实现方案：用户级和透明级。

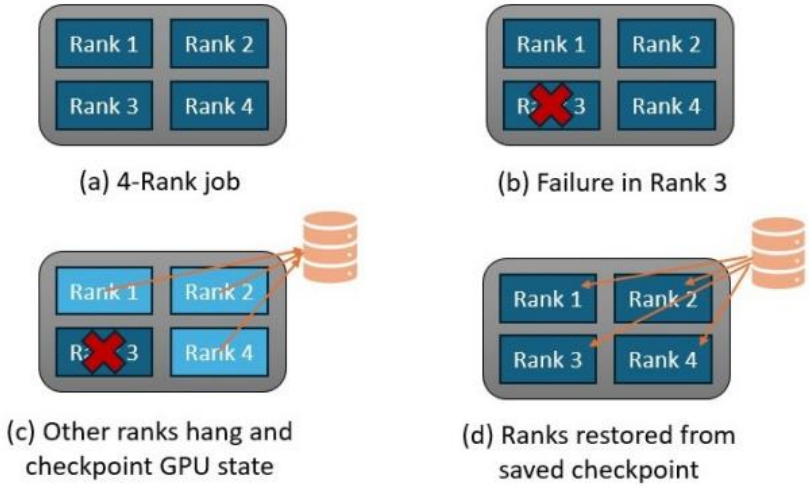


Figure 1. Just-in-time checkpointing

2.4 小结

解决方案

需解决问题

减少**稳态开销**

动态调整检查点频率

降低恢复**损失**

利用训练机器的本地内存缓存检查点文件，降低IO时间【Gemini, 2023】
通过擦除编码为deep-learning-based recommendation model实现高效容错【Tianyu, 2023】
利用量化技术和增量迭代的方式缩减检查点文件大小【Check-N-Run, 2022】
流水线并行、零冗余加载等技术加速Checkpoint存储/读取过程【ByteCheckpoint，ByteDance】
为使用PyTorch的分布式大规模模型训练作业提供高效的检查点解决方案【Nebula, Microsoft】
其他解决思路【参考文献5-8】

加速 缩减

利用强化学习指导检查点间隔的调整【DACM, 2023】
通过在线分析算法，动态调整检查点频率【CheckFreq, 2021】
基于处理率、检查点开销和平均故障间隔时间调整间隔【Zhuang Y, 2018】
通过故障概率调整检查点间隔【Akber, 2018】

按需

即时检查点【JIT, 2024】

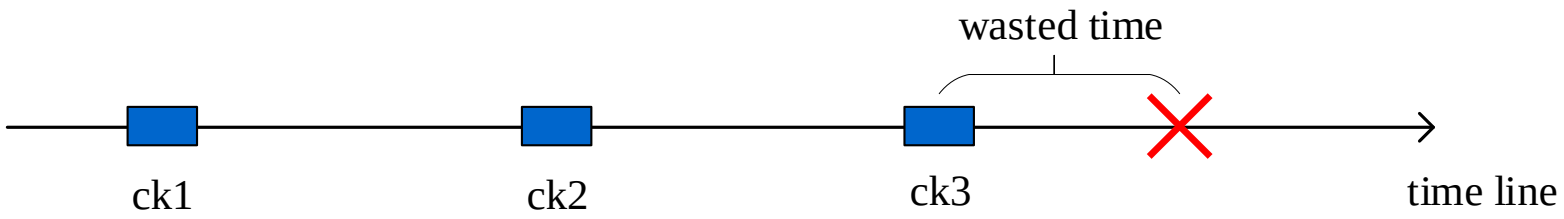
即时

之江实验室 ZHEJIANG LAB

3. 创新实践：TranLogs

3.1 研究背景

- 深度学习训练过程中**故障频发**（据OPT-175B的训练报告显示，在训练过程中，由于各种故障造成的训练失败，损失了约178000个GPU小时）。
- 检查点机制对于训练精度恢复的**不确定性**（与故障时间密切相关）。
  - 最理想的故障时间为刚进行过检查点记录，仅会损失短时间的历史训练信息；
  - 最糟糕的故障时间为即将进行检查点记录，会损失近乎一个检查点周期的历史训练信息。



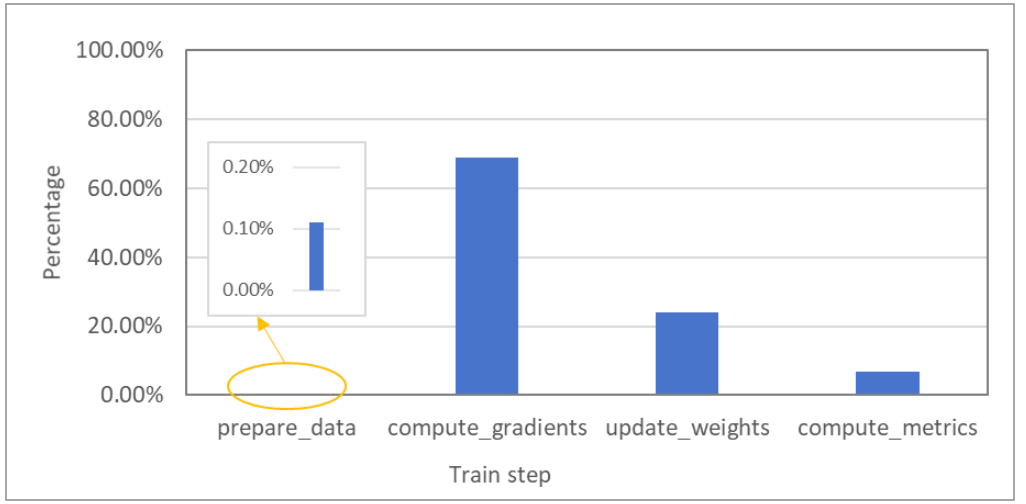
3.1 研究背景

- 文件系统中的**日志机制**是一种用于提高数据一致性和系统恢复能力的技术。其主要思想是通过记录文件系统操作的日志来确保在系统崩溃或意外断电后，可以恢复到一致的状态。
- 实现日志机制的文件系统，在遇到系统崩溃时，可以通过日志进行快速恢复，保证数据的完整性。
- 特点：事务性、顺序写入、恢复能力。
- 类型
  - 写前日志（WAL, Write-Ahead Logging）；
  - 写后日志（WAL, Write-Behind Logging）。

3.2 研究动机（Motivation）

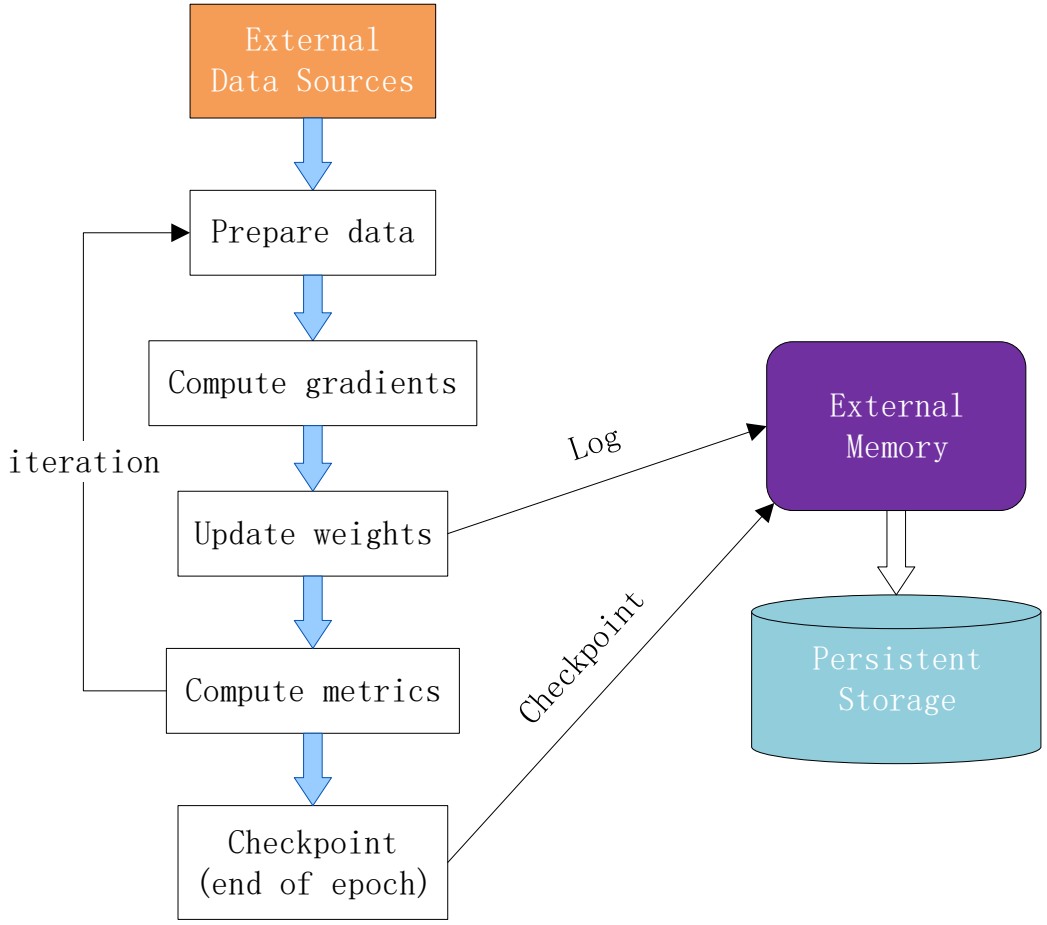
- 主要有以下几方面
  - 检查点机制对于训练精度恢复的不确定性；
  - 低周期检查点的高成本（以GPT-3，1750亿参数，float32为例）；
    - 单个检查点大小为7000GB，频率为10/100，所需空间为70000GB（约68.4TB）
    - 优化器状态，合计总存储空间为140000GB（约136TB）
  - 基于Bootstrap方法的训练过程耗时分析；
  - 时间和空间之间的权衡（tradeoff）。

不同策略	存储空间	节省训练时间 (占一个更新过程)
存储gradients	1.82GB	68.99%
存储weights	1.85GB	93.11%



3.3 设计和实现 – 核心思想

- 检查点机制与日志相结合
  - 训练过程中使用日志存储参数更新记录，检查点文件记录模型每轮次最新的完整状态；
  - 由“研究动机”部分关于训练步骤耗时的分析，计算梯度是较为耗时的步骤，TranLogs通过将该步骤中的更新记录到日志中，在恢复时从文件中读取该步骤的结果。

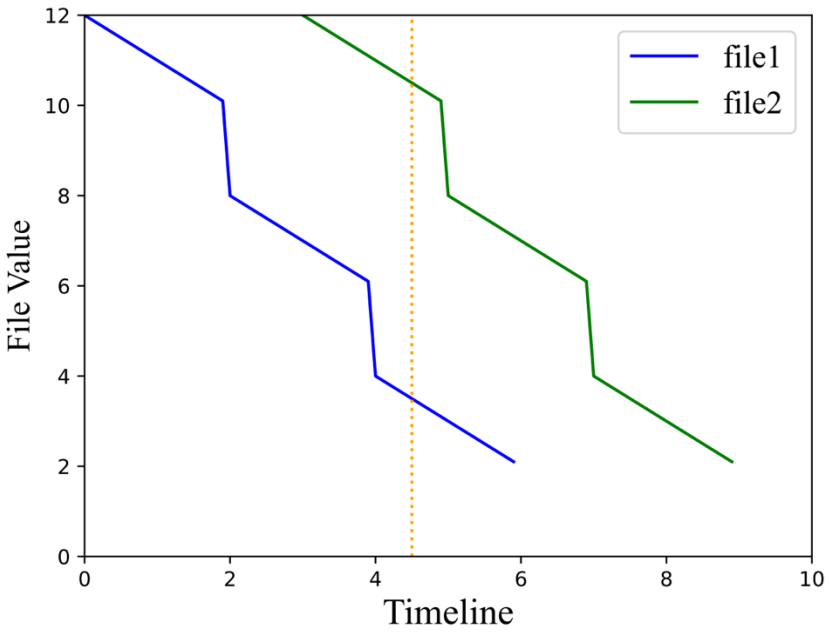


3.3 设计和实现 – 详细设计

- 日志高效迭代
  - 日志文件的价值变化理论，对于日志文件的价值估计如下：

$$V_i = (-t/t_e) \times u_{i-1} - \alpha_i$$

  - 日志文件价值随时间变化示意图



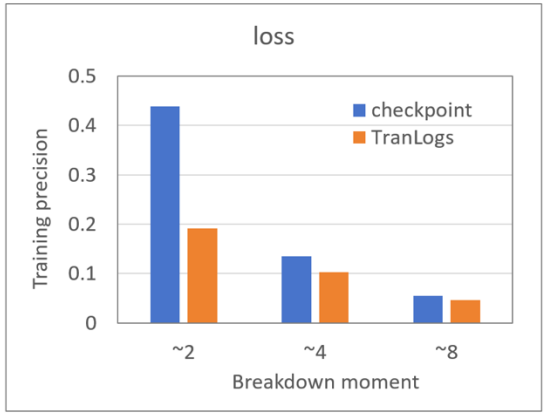
- 异步持久化策略：在写入日志文件时发挥作用，首先将文件写入内存缓存层，然后异步写入磁盘进行持久化。

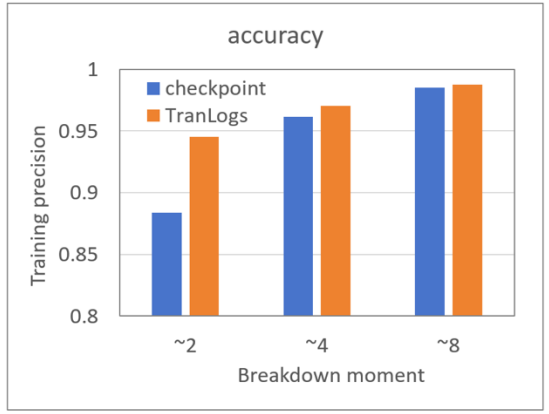
3.3 设计和实现 – 具体实现

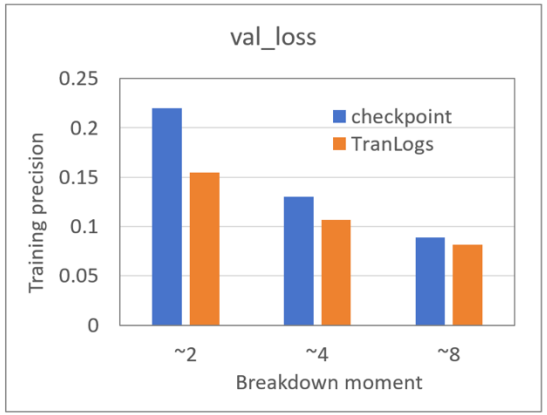
- 基于典型的深度学习训练框架Tensorflow和Keras进行TranLogs的实现
  - 对原框架中提供的训练函数进行修改，在其基础上添加训练过程记录日志和基于日志进行恢复所需要的操作代码。
  - 日志高效迭代策略模块，该模块是一个后台监控程序，用来监控日志文件所占存储空间，并进行日志文件的处理。
  - 异步持久化策略，该部分借助于Alluxio分布式缓存系统实现，Alluxio提供了ASYNC_THROUGH写入类型。

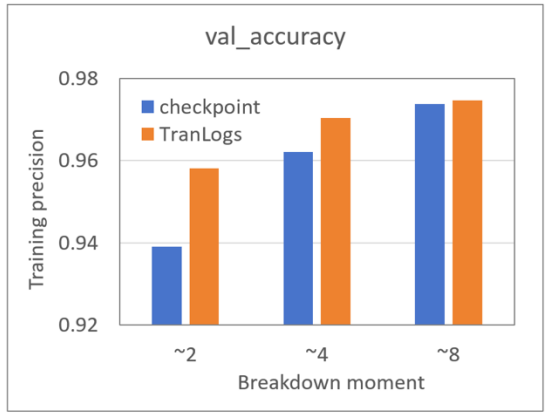
3.4 实验结果

- 对比实验：TranLogs与不采用日志做辅助的检查点机制（Checkpoint）对比
  - 在不进行重复迭代计算的前提下，实验结果显示，TranLogs可以将模型训练恢复到更接近故障时刻已达到的状态。
  - loss和accuracy最佳可分别实现约24%和6%的提升。



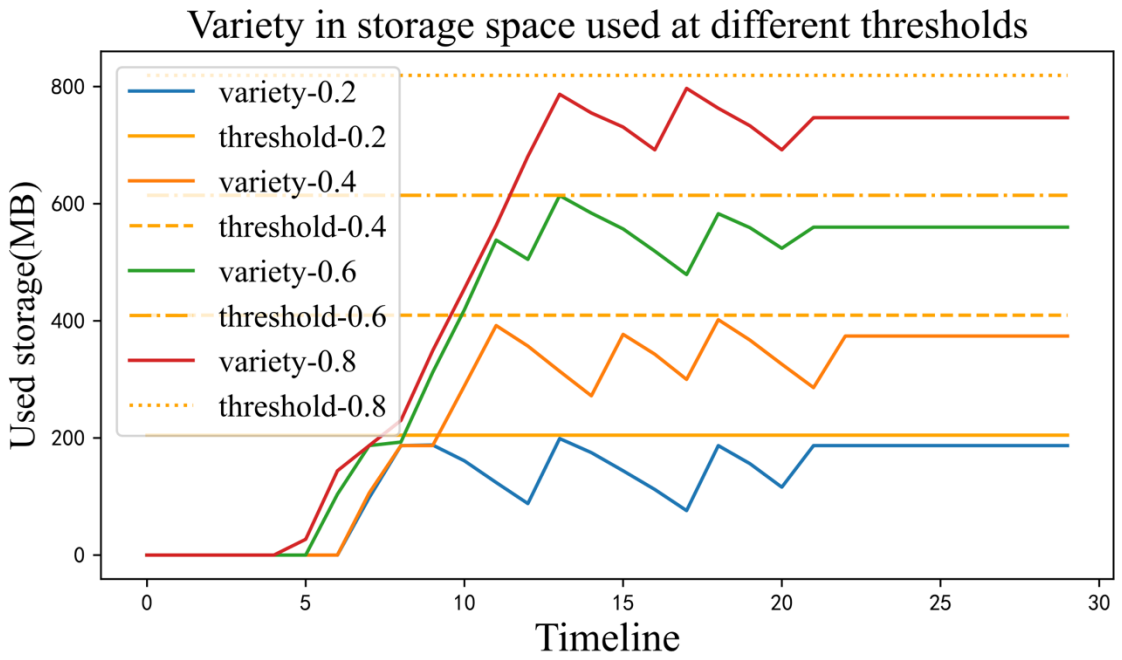




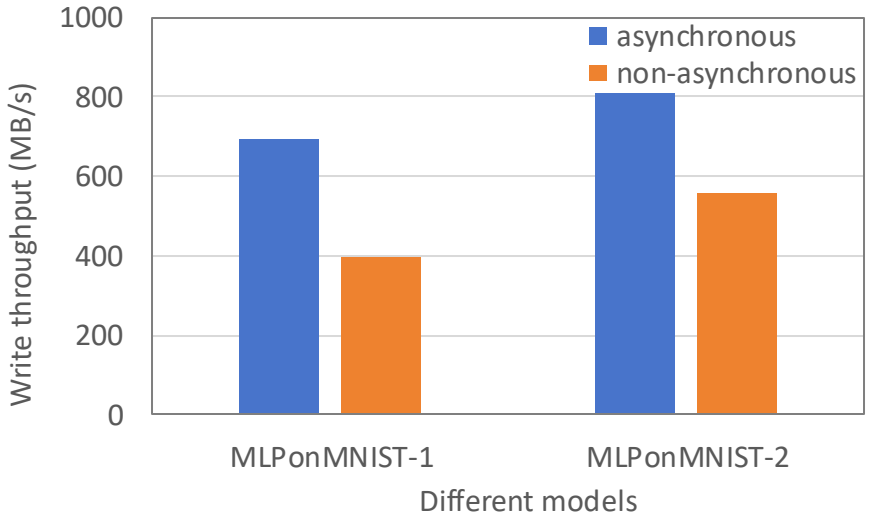


3.4 实验结果

- 日志高效迭代策略测试：在不同阈值下，日志文件所占存储空间的变化。



- 异步持久化策略测试：比较采取异步持久化策略和不采取该策略的情况下，日志文件的读写效率，测试针对不同结构的模型日志文件开展，结果如下图所示。



3.5 小结

- 提出一种基于日志方案的模型无损恢复机制TranLogs，以解决检查点机制难以在发生故障时将模型训练状态**无损恢复**至出故障时刻的问题。
- TranLogs通过使用**日志加检查点**文件的方式，探索了在深度学习训练故障恢复过程中降低训练损失的可能性。
- 文章中进行了实际模型训练过程不同阶段的耗时等分析，在这些理论分析的基础上，确定了基于日志的故障恢复策略的实施细节。
- 实验表明，与不采用日志辅助的检查点机制相比，TranLogs可以在不进行重复迭代计算的前提下，将模型训练恢复到**更接近故障时刻**已达到的状态，loss和accuracy最佳可分别实现约24%和6%的提升。

之江实验室 ZHEJIANG LAB

4. 总结

4. 总结

研究现状

- 减少稳态开销
- 动态调整检查点频率
- 降低恢复损失

创新实践

- 提出基于日志方案的无损恢复机制TranLogs
- 使用日志+检查点文件的方式
- 理论分析→实施细节
- 实验验证

持续探索大模型检查点机制存储优化的更多可能性，

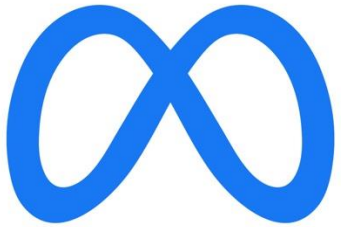
为大模型训练提供高效能存算技术支撑。

参考文献

1. Wang Z, Jia Z, Zheng S, et al. Gemini: Fast failure recovery in distributed training with in-memory checkpoints[C]//29th Symposium on Operating Systems Principles (SOSP 23). ACM, 2023: 364-381.
2. Zhang T, Liu K, Kosaian J, et al. Efficient Fault Tolerance for Recommendation Model Training via Erasure Coding[J]. Proceedings of the VLDB Endowment, 2023, 16(11): 3137–3150.
3. Eisenman A, Matam K K, Ingram S, et al. Check-N-Run: A checkpointing system for training deep learning recommendation models[C]//19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). USENIX, 2022: 929-943.
4. Microsoft. Boost Checkpoint Speed and Reduce Cost with Nebula[Online]. Azure Machine Learning, August 29, 2024. Accessed on: September 23, 2024. Available: https://learn.microsoft.com/enus/azure/machine-learning/reference-checkpoint-performance-for-largemodels?view=azureml-api-2&tabs=PYTORCH.
5. Chen M, Hua Y, Bai R, et al. A Cost-Efficient Failure-Tolerant Scheme for Distributed DNN Training[C]//2023 IEEE 41st International Conference on Computer Design (ICCD). IEEE, 2023: 150-157.
6. Nicolae B, Li J, Wozniak J M, et al. Deepfreeze: Towards scalable asynchronous checkpointing of deep learning models[C]//2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). IEEE, 2020: 172-181.
7. Agrawal A, Reddy S, Bhattamishra S, et al. DynaQuant: Compressing Deep Learning Training Checkpoints via Dynamic Quantization[J]. arXiv preprint arXiv:2306.11800, 2023.
8. Cores I, Rodríguez G, Martín M J, et al. Improving scalability of application-level checkpoint-recovery by reducing checkpoint sizes[J]. New Generation Computing, 2013, 31: 163-185.
9. Zhang Z, Liu T, Shu Y, et al. Dynamic Adaptive Checkpoint Mechanism for Streaming Applications Based on Reinforcement Learning[C]//2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS 22). IEEE, 2023: 538-545.
10. Mohan J, Phanishayee A, Chidambaram V. CheckFreq: Frequent,FineGrainedDNN Checkpointing[C]//19th USENIX Conference on File and Storage Technologies (FAST 21). USENIX, 2021: 203-216.
11. Zhuang Y, Wei X, Li H, et al. An optimal checkpointing model with online OCI adjustment for stream processing applications[C]//27th International Conference on Computer Communication and Networks (ICCCN 18). IEEE, 2018: 1-9.
12. Akber S M A, Chen H, Wang Y, et al. Minimizing overheads of checkpoints in distributed stream processing systems[C]//7th International Conference on Cloud Networking (CloudNet 18). IEEE, 2018: 1-4.
13. Gupta T, Krishnan S, Kumar R, et al. Just-In-Time Checkpointing: Low Cost Error Recovery from Deep Learning Training Failures[C]//19th European Conference on Computer Systems (EuroSys 24). ACM, 2024: 1110-1125.

参考资料

1. Meta研究论文页面：https://research.facebook.com/publications/
2. Microsoft Research：https://www.microsoft.com/en-us/research/
3. OpenAI：https://platform.openai.com/docs/overview
4. 字节-豆包大模型团队：https://team.doubao.com/zh/publication
5. 阿里：https://www.alibabacloud.com/help/zh/pai/user-guide/easyckpt



Meta

ByteDance

字节跳动

Microsoft

谢谢！

之江实验室 ZHEJIANG LAB