浙江大学

课程报告

课程名称：大数据存储与计算技术

姓　　名：陈诺 3210102020　詹含蓓 3210106333
夏菲 3210103965　钟睿昕 3210102465

学　　院：计算机科学与技术学院

专　　业：计算机科学与技术

指导教师：陈华钧

报告日期：2024 年 6 月 8 日

目录

大语言模型的存储挑战

0 总体介绍

0.1 存储现状

大语言模型作为人工智能领域的一个突破性进展，对存储技术提出了前所未有的要求。随着模型规模的不断扩大，存储需求从最初的 TB 级别迅速增长到 EB 级别。根据相关规划，到 2025 年，我国存储总量要超过 1800EB。而到 2023 年 6 月底，全国存力总规模为超过 1080EB。这意味着中间还有超 700EB 的巨大存储缺口亟待补充。巨大的缺口意味着巨大的增长潜力，反映出当前存储技术面临着相当多的机遇与挑战。

0.2 发展阶段

0.2.1 初期阶段：Lustre

Lustre 文件系统最初被采用，是因为它在高性能计算（HPC）领域有着深厚的基础，并且提供了符合 POSIX 标准的 UNIX 文件系统接口。这对于需要处理大规模数据集的 AI 训练来说至关重要。Lustre 的设计允许多个客户端并行访问存储集群，这在处理 AI 训练过程中产生的庞大数据量时显得尤为高效。

然而，随着 AI 模型的复杂度和数据集的规模不断增长，Lustre 的一些局限性开始显现。其中包括：

- **升级困难**：Lustre 作为一个内核态的文件系统，升级过程复杂，往往需要停机进行，这影响了集群的可用性。
- **数据孤岛问题**：不同地理位置的训练集群之间进行数据迁移和访问时效率低下。
- **性能瓶颈**：随着单次训练涉及的数据量激增，Lustre 的元数据服务（MDS）开始遇到容量和性能的瓶颈。
- **管理复杂性**：随着集群数量的增加，权限管理、资源分发和数据管理变得越来越复杂。

0.2.2 当下发展：BeeGFS+Ceph

为了克服 Lustre 的局限性，业界开始探索新的存储技术组合，即 BeeGFS 和 Ceph OSS。BeeGFS 是一个应用态实现的并行文件系统，它在很多方面对 Lustre 进行了增强，例如：

- **分布式 MDS 集群**：允许系统轻松扩展到数十组节点，支持数 PB 的存储容量。
- **基于副本的可靠性**：提供了数据副本，增强了数据的安全性。
- **简化的维护和开发**：服务端组件在应用态实现，降低了维护成本和开发难

度。

Ceph OSS 则以其出色的稳定性和社区支持，被选为存储原始数据的解决方案。Ceph 的设计允许它作为一个数据湖来存储大量的原始数据，并提供 HDD 和 SSD 存储池以满足不同的性能和容量需求。

而随着大型语言模型（LLM）的出现，AI 模型参数的急剧增长，BeeGFS+Ceph OSS 方案开始面临新的挑战：

- **吞吐量不足**：大模型训练和多模态训练需要更高的 I/O 吞吐量。
- **性能扩展性不足**：现有方案在处理大规模文件系统时的性能扩展性受限。
- **存储效率偏低**：面向 HDD 设计的存储系统在 NVMe 等新型硬件上的性能未能充分发挥。

0.2.3 展望未来：DAOS

为了应对新技术的挑战，Intel Daos 作为一种新兴的分布式异步对象存储技术开始受到广泛关注。Daos 的特点包括：

- **对新型硬件的支持**：Daos 能够更好地利用现代硬件，比如 NVMe SSD，提供更高的性能。
- **CXL 技术的应用前景**：Daos 预期将与未来的计算扩展（CXL）技术相结合，进一步提升存储性能。
- **Metadata-on-SSD**：Daos 的这一版本预计将显著提升元数据的处理能力，进一步支持大规模 AI 训练。

0.3 过程分析

0.3.1 大语言模型训练生命周期及对存储的要求

		数据采集	预处理	模型训练	推理
		是指通过网络爬虫,数据采购等方式收集模型训练所需要的原始数据并储存起来	对采集的原始数据进行清洗和处理,如:去重,填充缺失值、删除异常值,(人工)标注等	主要是根据预定的目标和要求,选择合适的机器学习算法或构建深度学习网络对预处理后的数据进行训练,建模	利用训练好的模型对新的数据进行预测,并将预测结果应用到实际场景中,例如分类、回归、聚类等
存储需求		具备处理高度多变数据格式的灵活性	减少数据准备时间	缩短训练时间	具备处理高度多变数据格式的灵活性
对存储系统的要求	访问模式	顺序写	随机	随机	随机
	工作负载	写密集型	读写混合(可能达到50/50)	读密集型(某些写入来自checkpoint)	读密集型
	I/O大小	大	大	小	小
	服务质量	高	高	非常高	非常高
	吞吐量	非常高	非常高	非常高	中等

大语言模型对存储系统的要求比起计算机视觉（AI 发展初期的聚焦点）更高。计算机视觉训练更多的是随机小 I/O，对 IOPS 和延迟更敏感，大模型训练则是混合 I/O，对 IOPS，吞吐及延迟都提出了很高的要求。

0.3.2 模型训练挑战

模型训练是大模型生命周期中的核心环节，它直接决定了深度学习模型对数据的理解能力、预测准确性和泛化性能。大模型训练的存储容量需求高，不仅参数量庞大，在训练的每一步中还需要保存前向传播过程中产生的激活量和用于参数更新的优化器。

- **模型参数**（model parameters），大规模深度学习模型的参数量爆炸式增长，且这种趋势仍在持续。

- **激活量**（activation），根据反向传播算法，前向传播阶段产生的激活量需要被先保存，然后在反向传播时用于计算梯度信息，最后在梯度计算结束后被释放. 在大模型训练过程中，激活量所占的存储空间庞大. 根据研究发现，激活量在训练过程中的存储开销占总存储开销的 70%。

- **优化器**（optimizer）。 计算得到的梯度被传输到优化器中以更新得到新版本的模型参数。

- 此外，大模型训练过程中的**容错需求高**。大模型训练会使用大量 GPU，这也导致了故障的可能性提高。

传统支持大规模数据存储系统一方面未充分利用大模型训练中的计算模式、访存模式和数据特征，另一方面不适用于大模型数据更新数据量大、更新频繁的特点，严重影响大模型训练的效率。

1. **传统的分布式存储技术不适用于大模型训练的计算模式：**一方面，大模型训练常使用的 GPU 具有计算资源和存储资源强耦合的特点，需要考虑计算任务与存储之间的依赖关系；另一方面，传统的分布式存储技术未利用大模型训练中各个任务间的数据依赖关系进行优化，可能导致相邻任务间的数据传输方案非最优。

2. **传统的异构存储技术对大模型训练中的访存模式不感知：**未利用这些访存模式设计数据的预取和传输策略，因而无法达到训练的最佳性能。

3. **传统的存储缩减技术不适用于大模型训练中的数据特征：**大模型数据稠密度高，难以通过传统的压缩方法缓解存储压力。

4. **传统的存储容错技术在大模型训练场景下容错开销大：**大模型训练中的数据量庞大，并且数据更新频繁。

1 存储需求分析

1.1 数据存储需求分析

1.1.1 数据存储类型

大语言模型的存储数据可以分为原始训练数据、模型特征数据、模型超参数数据、模型参数数据、实验数据、日志数据等。

1. **原始训练数据**主要由文本组成，这些文本数据是大语言模型训练和推理过程的核心。文本数据用于执行各种自然语言处理任务，如语言翻译、情感分析、文本摘要等。由于文本数据是非结构化的，它们需要存储在能够高效处理此类信息的系统中，例如分布式文件系统（如 Hadoop Distributed File System, HDFS）或对象存储系统（如 Amazon S3、Google Cloud Storage）。随着多模态学习的发展，大语言模型可能还需要结合图像、视频和音频数据进行处理，这要求存储系统不仅要能够处理文本数据，还要能够高效管理其他类型的非结构化数据。

2. **模型特征数据**是经过经过预处理和特征提取的数据，用于模型训练。**模型超参数数据**用于控制模型训练过程，包括学习率、批量大小和网络层数等。**模型参数数据**包括训练好的模型权重和偏置参数，通常是数值型数据。以上数据均属于结构化的数据，可以使用结构化的存储系统存储，如关系型数据库或分布式文件系统。

3. **实验数据**记录了在 AI 模型开发过程中进行的各种实验的详细信息，包括实验设置、参数配置、结果指标等。这些数据对于评估和优化模型至关重要，通常存储在支持版本控制和历史跟踪的系统中。

4. **日志数据**提供了系统运行时的详细记录，包括错误信息、系统状态、用户行为等。日志数据对于监控系统健康、分析用户行为、调试问题和保证系统安全至关重要。日志数据通常需要存储在能够支持高写入吞吐量和灵活查询的系统中。

1.1.2 数据存储数量

1. **原始训练数据：**

如果按照平均一个文本页面（假设 500 字，每字 2 字节）计算，大约需要 1000 字节。若全球有约 500 亿个网页（Google 搜索引擎的索引规模），则需要大约 50TB 的存储空间。

2. **模型特征数据：**

模型特征数据的大小取决于特征提取的复杂性和数据集的大小。假设每个特征向量为 1KB，并且有 10 亿个特征向量，则特征数据可能需要约 100TB 的存储空间。

3. **模型超参数数据：**

模型超参数数据相对较小，但如果考虑到模型需要调整的参数数量众多，假设

有 1000 个超参数，每个参数记录大小为 1KB，则总存储需求可能在 1GB 左右。

4. **模型参数数据：**
大语言模型可能包含数十亿甚至数万亿个参数（GPT-3 拥有 1750 亿个参数）。如果每个参数及其相关信息平均占用 1 字节，则模型参数数据可能需要数十 TB 到数 PB 的存储空间。

5. **实验数据：**
实验数据记录了模型开发过程中的各种实验信息。如果每次实验产生 1MB 的记录，并且进行了 1 万次实验，则实验数据可能需要约 10TB 的存储空间。

6. **日志数据：**
日志数据记录了系统运行时的详细信息。如果每天产生 1GB 的日志数据，一年则需要约 365GB 的存储空间。

综合以上各种数据类型，大语言模型的所有信息总存储量可能在 PB 级别，具体取决于模型的规模、训练数据的多样性。

1.1.3 数据存储性能

1. 高吞吐量：为了应对大语言模型在训练过程中生成和处理的海量数据，存储系统必须具备**高吞吐量**。这意味着系统需要快速处理大量的读写请求，确保数据流的连续性和稳定性。例如，在模型训练期间，可能会有每秒数千到数万次的读写操作，这就要求存储系统能够提供数十 GB/s 的 I/O 带宽。

2. 高并发 I/O 操作：除了高吞吐量，大语言模型的训练通常在 GPU 或 TPU 等并行计算平台上执行，这要求底层存储系统能够支持**高并发 I/O 操作**。存储系统需要能够同时处理来自多个计算节点的大量 I/O 请求，避免出现性能瓶颈。在分布式训练场景中，这一点尤为重要，存储系统需要确保每个 GPU 节点都能以高带宽和低延迟访问数据，以实现高效的并行处理。

3. 低响应时间：**低响应时间**也是存储性能的关键要求之一。为了保障高效的数据处理和优化用户体验，存储系统必须确保数据存取操作的响应时间尽可能短。在模型训练和推理过程中，数据的加载和保存时间需要尽可能短，以提高整体效率。例如，加载一个大型数据集或保存模型参数时，响应时间应控制在 10 秒以内。

4. 数据的一致性和可靠性：**数据的一致性和可靠性**对于大语言模型同样至关重要。训练和推理依赖于数据的准确性和一致性，存储系统需要提供数据一致性保证，确保在并发访问和高负载条件下数据的完整性和准确性。在多节点并行训练环境中，存储系统需要确保所有节点对数据的访问是同步的，避免数据不一致导致训练失败或结果不准确。

5. 可拓展性：随着模型规模的扩大和数据量的增长，存储系统的**可扩展性**变得尤为重要。存储系统应能够无缝扩展容量和性能，以适应不断变化的存储需求，而不影响现有操作。

6. 容错能力：存储系统必须具备**容错能力**，在组件故障或系统异常时能够继续运行，并保护数据不受损失。在发生硬件故障或网络中断的情况下，存储系统应能够自动切换到备用组件，并确保数据的完整性和可用性。

1.1.4 数据更新

- 高频率更新：大语言模型的训练是一个持续演化的过程，伴随着大量的数据读写操作。在这一过程中，**模型参数会不断地被更新和优化**。例如，每次训练迭代都可能导致数亿次的参数更新。这就要求存储系统不仅要有足够的容量来存储这些庞大的数据，还要能够快速响应这些更新操作。

- 同步更新：为了应对这种高频率的更新，存储系统必须设计一套高效的数据更新机制。在实际应用中，这意味着当用户在不同计算节点上并行训练模型时，为了避免数据冲突和保持数据一致性，存储系统需要能够**迅速同步这些更新**，以保证所有节点都能够访问到最新的数据。这种同步机制确保了即使在多个节点同时更新数据的情况下，也能保持数据的完整性和准确性，对于保持模型训练的一致性和高效性至关重要。

- 追踪更新：随着模型参数的迭代，有效的数据版本控制也变得尤为重要。存储系统**需要能够追踪每次更新**，允许研究人员在必要时回溯到特定的训练状态，或者比较不同版本之间的差异。

- 智能缓存：为了提高数据访问速度，存储系统可以采用智能缓存策略，将频繁访问的数据暂存于高速缓存中。这样不仅可以减少对后端存储的访问次数，还能显著提升数据检索的速度。

1.1.5 数据生命周期

1. **数据创建：**
大语言模型需要持续摄入新鲜的数据来优化性能和适应新的语言使用趋势。这要求能够高效采集和存储新数据，如用户反馈，网络文章等。

2. **数据存储：**
大语言模型产生的数据类型多样，包括结构化的元数据和非结构化的文本、音频数据等。需要采用合适的存储解决方案，例如使用分布式文件系统存储非结构化数据，关系数据库存储结构化数据。

3. **数据使用：**
为了支持模型的训练和推理，存储系统必须支持高效的数据访问，并发数据处理，数据的即时处理、数据一致性和完整性等。

4. **数据备份：**
鉴于大语言模型对数据的依赖性，定期进行全量和增量备份至关重要。这确保了在任何数据丢失或损坏的情况下，可以迅速恢复到正常状态。

5. **数据恢复：**
存储系统需要具备快速的数据恢复能力，以应对可能的系统故障或灾难。这包括建立灾难恢复计划和定期进行数据恢复演练。

6. **数据销毁：**
对于不再需要或已过期的数据，需要安全地销毁，以遵守数据保护法规和隐私政策。这要求使用专业工具彻底删除数据，并进行相应的报告和审计。

1.1.6 数据安全管理

1. **数据访问控制：**
实施严格的数据访问控制机制，确保只有授权用户才能访问敏感数据。这可能包括多因素认证、权限分级等安全措施。

2. **数据加密：**
对存储和传输中的数据进行加密，以防止未授权访问和数据泄露。这包括静态数据的加密存储和动态数据的加密传输。

3. **数据完整性：**
采取校验和、哈希等技术确保数据在存储和传输过程中的完整性，防止数据被篡改。

4. **安全监控：**
实施实时监控和审计，以检测和响应可能的安全威胁。这包括异常访问模式的检测、未授权访问的警报等。

5. **合规性：**
确保数据存储和管理流程遵守相关的数据保护法规和行业标准，如 GDPR、HIPAA 等。

1.2 用户行为分析

在大语言模型中进行用户行为分析时，需要记录和分析一系列关键信息，以便更好地理解用户需求、优化模型性能和提升用户体验。以下是一些主要的信息记录点：

1. **交互数据：**
    - 用户与模型交互的详细日志，包括提问、命令和反馈的具体内容。
    - 交互的时间戳、持续时长以及交互发生的环境（如设备类型、操作系统）。
2. **用户反馈：**
    - 用户对模型响应的满意度评价，包括点赞、投诉或其他形式的反馈。
    - 用户对模型错误的纠正建议或额外信息的提供。
3. **使用习惯：**
    - 用户登录和注册的行为模式，包括频率、时间段和使用设备。
    - 用户输入的文本内容、风格和交互的复杂性。
4. **内容生成：**
    - 用户生成的内容主题、风格和情感倾向。
    - 用户对模型生成内容的采纳情况和使用方式。
5. **搜索查询：**
    - 用户在模型中进行的搜索行为，包括搜索词、搜索频率和搜索结果的互动。

- 用户对搜索结果的满意度和进一步的行动。

6. **用户偏好：**
   - 用户对特定类型内容或服务的偏好。
   - 用户对模型功能和服务的使用情况和反馈。
7. **用户行为序列：**
   - 用户在模型中的操作序列，如提问、等待、接收反馈和进一步交互。
   - 用户行为的时间序列分析，以识别行为趋势和模式。
8. **合规性和隐私：**
   - 确保在收集和分析用户数据时遵守相关的数据保护法规和隐私政策。
   - 用户数据的匿名化和去标识化处理，以保护用户隐私。

通过记录和分析这些信息，大语言模型可以实现更精准的用户理解，不断优化模型服务，提升用户满意度，并在遵守合规性和保护隐私的前提下，实现数据的最大化利用。

2 数据模型设计

大语言模型的存储数据主要分为两类：`ObjectData` 和 `MetaData`。`ObjectData` 是用户希望存储的数据，而 `MetaData` 是包括访问权限、文件大小和位置的“关于数据的数据”，`MetaData` 中最重要的是如何从多个文件服务器中找到具体对应的文件，这样才能使客户端获取特定文件或目录的 `MetaData` 后，可以直接与 `ObjectData` 服务器对话以检索信息。以下是我们按训练、推理和生成三个阶段顺序设计的 `ObjectData` 和 `MetaData` 数据模型：

2.1 ObjectData 数据模型

- `TrainingData` 表：主要存储用于训练的文本数据和相关信息，包括自动生成的唯一标识符（id）、训练文本内容（text）、文本标签（label，若有）、数据来源（source，如数据集名称）以及数据记录的时间戳（timestamp）。其中，id 是主键，确保数据的唯一性和便于快速查找

- `ModelParameters` 表：主要存储训练后的模型参数及其元数据，包括自动生成的唯一标识符（id）、模型的唯一标识符（model_id）、参数名称（parameter_name）、参数值（parameter_value，以二进制格式存储）以及参数记录的时间戳（timestamp）。其中，id 是主键，确保数据的唯一性和便于快速查找。

- `Vocabulary` 表：主要存储模型使用的词汇表及其索引，包括自动生成的唯一标识符（id）、词汇或子词（token）、词汇或子词的索引（token_index）以及关联的模型唯一标识符（model_id）。其中，id 是主键，确保数据的唯一性和便于快速查找。

- `TrainingStatistics` 表：主要存储训练过程中的统计信息，包括自动生成的唯一标识符（id）、当前训练的 epoch 编号（epoch）、当前的批次号（batch_number）、当前的损失值（loss_value）、当前的准确度（accuracy）以及记录的时间戳（timestamp）。其中，id 是主键，确保数据的唯一性和便于快速查找。

- `InferenceLog` 表：主要存储推理过程中的日志信息，包括自动生成的唯一标识符（id）、输入的文本（input_text）、模型生成的输出文本（output_text）、使用的模型唯一标识符（model_id）以及推理过程的时间戳（timestamp）。其中，id 是主键，确保数据的唯一性和便于快速查找。

- `GenerationStrategy` 表：主要存储生成过程中的策略参数，包括自动生成的唯一标识符（id）、关联的模型唯一标识符（model_id）、生成温度（temperature）、top-k 采样参数（top_k）、top-p 采样参数（top_p）以及记录的时间戳（timestamp）。其中，id 是主键，确保数据的唯一性和便于快速查找。

2.2 MetaData 数据模型

- `DatasetMetadata` 表：主要存储训练数据集的相关信息，包括数据集唯一标识符（dataset_id）、数据集名称（name）、数据集描述（description）、数据来源（source）、数据集文件大小（file_size）、数据集文件存储位置（file_location）、数据集创建日期（creation_date）、数据集最后修改日期（last_modified_date）以及访问权限信息（access_permissions）。其中，dataset_id 是主键，确保数据的唯一性和便于快速查找。

- `TrainingJobMetadata` 表：主要存储训练作业的相关信息，包括训练作业唯一标识符（job_id）、训练的模型唯一标识符（model_id）、训练作业开始时间（start_time）、训练作业结束时间（end_time）、训练作业状态（status，如进行中、完成、失败）、训练超参数（hyperparameters，以 JSON 格式存储）、关联的训练数据集标识符（training_data_id）以及训练日志文件存储位置（log_location）。其中，job_id 是主键，确保数据的唯一性和便于快速查找。

- `InferenceRequestMetadata` 表：主要存储每次推理请求的相关信息，包括推理请求唯一标识符（request_id）、使用的模型唯一标识符（model_id）、输入数据存储位置（input_data_location）、输出数据存储位置（output_data_location）、推理请求时间（request_time）、推理响应时间（response_time）、推理请求状态（status，如成功、失败）以及访问权限信息（access_permissions）。其中，request_id 是主键，确保数据的唯一性和便于快速查找。

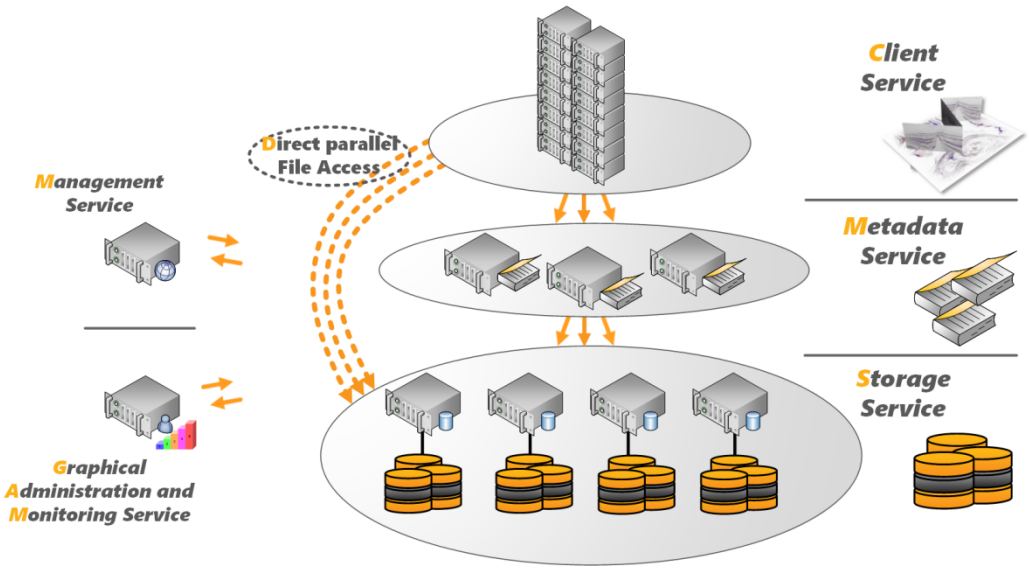
- `GenerationRequestMetadata` 表：主要存储每次文本生成请求的相关信息，包括生成请求唯一标识符（generation_id）、使用的模型唯一标识符（model_id）、生成文本的初始提示（prompt）、生成参数（generation_parameters，如温度、top-k、top-p，以 JSON 格式存储）、初始提示存储位置（input_data_location）、生成结果存储位置（output_data_location）、生成请求时间（request_time）、生成响应时间

（response_time）、生成请求状态（status，如成功、失败）以及访问权限信息（access_permissions）。其中，generation_id 是主键，确保数据的唯一性和便于快速查找。

3 存储架构设计

3.1 架构介绍

在大语言模型的存储架构设计上，我们选取了 BeeGFS 作为我们的基础架构，并在其基础上进行网络拓扑，负载均衡等设计。BeeGFS（Formerly known as Fraunhofer FS）是一个为高性能、易用性和可扩展性而设计的分布式文件系统。



3.1.1 主要组件

对于大语言模型，我们可以充分利用 BeeGFS 文件系统的特点来完成存储架构的构建。BeeGFS 所包含的四个主要组件在这里都将发挥重要作用：

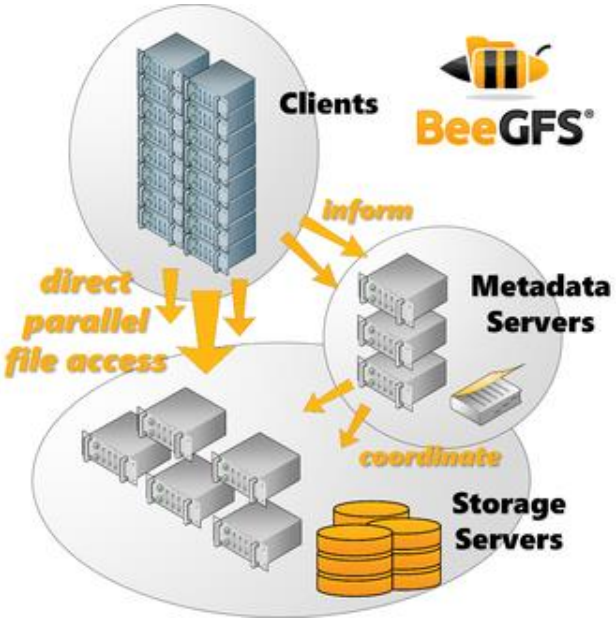
管理服务器(MS)是整个系统的核心，在大语言模型的存储结构当中，它负责跟踪所有其他服务的状态，以及在新服务启动时收集和分发配置。

对象存储服务器(OSS)负责管理文件系统中的对象，这种组件在硬盘驱动器（Object Storage Target，即 OST）上存储文件的实际数据。在大语言模型的训练过程中，我们需要处理大量的数据输入和模型参数的储存。在这个过程中，我们需要将一部分或者全部训练数据和模型参数分散存储在多个 OSS 上的多个 OST 中。这样可以利用 BeeGFS 的分布式特性，加速数据的读取和写入，提高训练效率。

元数据服务器(MDS)管理文件系统的元数 据，比如文件名、目录结构、文件权限等。每个 MDS 都有一个 MetaData Target(MDT，存储 MDS 的 Meta Data)。它保存了所有训练数据和模型参数的位置信息。客户端在需要读取或写入数据时，会先联系 MDS，获取对应数据的元数据。

客户端(Client)，在大语言模型实践当中，客户端指的是进行训练和预测等计算任务的计算机或计算节点。一个客户端可以同时访问多个 BeeGFS 实例，为了保证训练的流畅进行，我们应确保客户端与 OSS 之间的网络带宽足够，以便快速进行数据的读取和写入。

此外，对于 BeeGFS 的两个守护进程，其中 Helper-daemon 守护进程将在客户端上运行，保证其对 BeeGFS 文件系统的顺利访问。而 Admon 守护进程可以帮助我们在大语言模型训练过程中监控存储系统的状态，及时发现并解决可能存在的问题。



3.1.2 数据存储流程

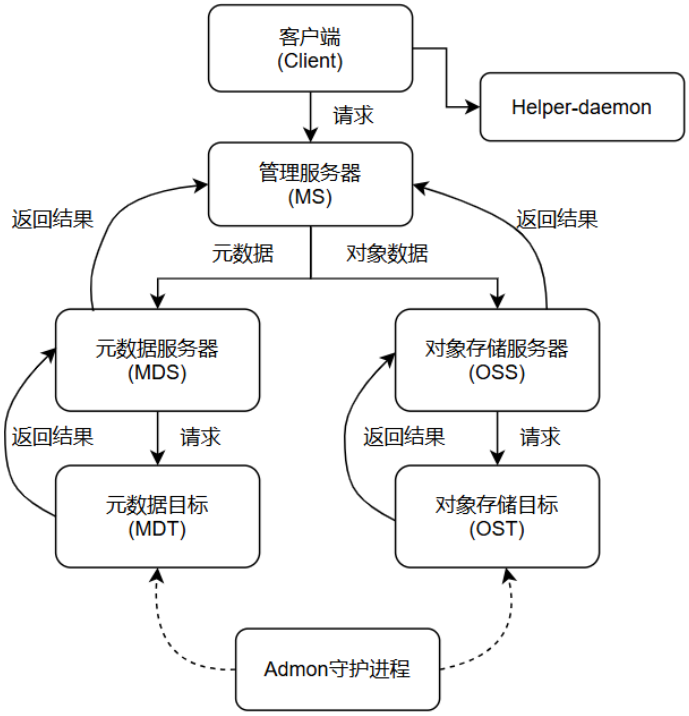
为了方便更直接的理解大语言模型在 BeeGFS 架构下如何完成存储，下面利用数据存储的流程来进行直观的介绍：

1. **客户端发送请求**：大语言模型在客户端发起对存储资源的请求，如读取数据、写入数据或查询元数据等操作。
2. **客户端与 Helper-daemon 交互**：客户端通过内核模块与本地运行的 Helper-daemon 守护进程交互。Helper-daemon 提供辅助功能，如 DNS 解析和日志记录。
3. **请求路由至管理服务器（MS）**：客户端的请求首先被发送到管理服务器（MS）。MS 负责维护文件系统组件的列表，包括客户端、元数据服务器（MDS）、对象存储服务器（OSS）等。
4. **数据操作**：分为两种
    a. **元数据操作**：如果请求涉及元数据（如文件属性、权限等），MS 将请求转发到相应的元数据服务器（MDS）。MDS 负责处理元数据，并存储在元数据目标（MDT）上。
    b. **数据存取操作**：对于数据存取请求，MS 将请求转发到 OSS。OSS 作为文件内容的主要存储服务，它将请求进一步转发到相应的对象存储目标（OST）。

5. **数据读写和其他存储操作**：在 OSS 的控制下，OST 执行实际的数据读写操作。OST 可以是本地文件系统或 LUN，并且通常配置为 RAID 以提供数据保护。同时在存储过程中，BeeGFS 支持数据复制和条带化，以提高性能和可靠性。条带化是指将文件数据分割成多个数据块，分布在不同的 OST 上，以实现更高的并行性和吞吐量。

6. **响应客户端**：一旦数据操作完成，OST 将结果返回给 OSS，OSS 再将结果返回给 MS，最终 MS 将响应发送回客户端。

7. **监控和日志记录**：Admon 守护进程在存储集群中运行，帮助系统管理员监控系统状态，记录日志，并在出现问题时提供警报。



3.1.3 优势

选取 BeeGFS 架构作为大语言模型的存储架构，是考虑到其具有以下优势：

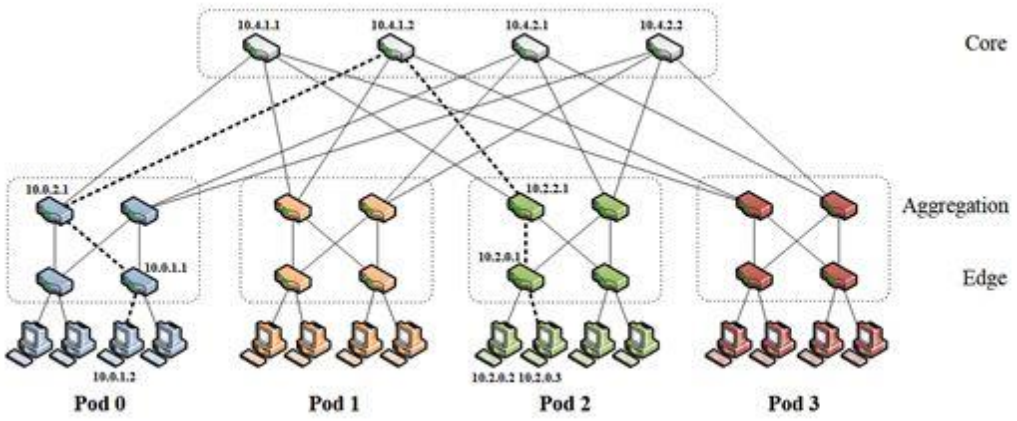
- 高性能：BeeGFS 设计用于提供高吞吐量和低延迟的 I/O 操作，这对于大模型的训练和推理至关重要。
- 可扩展性：BeeGFS 支持水平扩展，可以轻松添加更多的存储节点和元数据服务器，以满足不断增长的数据和计算需求。
- 并行访问：BeeGFS 支持并行文件访问，这意味着多个计算节点可以同时读写文件，这对于分布式训练非常有用。
- 数据管理：BeeGFS 的自动数据分片和负载均衡机制可以优化数据的存储和访问。
- 容错能力：BeeGFS 具有故障检测和恢复机制，可以自动处理节点故障，确保数据的持续可用性。

- 灵活的部署：BeeGFS 可以部署在各种硬件平台上，包括传统的服务器、云环境和虚拟化环境。
- 监控和管理：BeeGFS 提供了监控工具和 API，方便管理员监控和管理系统状态。
- 开源：作为开源软件，BeeGFS 可以节省许可费用，并且可以根据特定需求进行定制。
- 社区支持：BeeGFS 拥有一个活跃的社区，提供技术支持和持续的软件更新。

3.2 网络配置和拓扑

对于基于 BeeGFS 的大语言模型存储结构，我们选择使用胖树（Fat-Tree）的 CLOS 网络架构，这主要是基于以下考虑：

- Fat-Tree 的基本理念是：使用大量低性能的交换机，构建出大规模的无阻塞网络，能够保证对于任意的通信模式，总有路径让他们的通信带宽达到网卡带宽。
- Fat-Tree 用到的所有交换机都是相同的，这让我们能够在整个数据中心网络架构中采用廉价的交换机。
- Fat-Tree 是没有带宽收敛的，即树根处的网络带宽等于各个叶子处所有带宽的总和。这使得其能够保证无阻塞网络。



在 Fat-Tree 网络架构中，每个 BeeGFS 服务都直接连接到一个交换节点，即叶结点。这些叶结点进一步连接到高级别的中间结点。所有的叶结点都通过中间结点相互连接，构成了一个完全互联的网络。同时，由于 Fat-Tree 是没有带宽收敛的，这保证了这种网络架构为大语言模型提供了大量的带宽，几乎无法出现网络瓶颈。这对于处理大数据量、高并发性以及具有高 IO 需求的大语言模型训练非常有利。

Fat-Tree 网络架构具有良好的扩展性。每个网络节点（也就是 BeeGFS 的服务，如 MS、OSS、MDS 和客户端）连到中心交换节点，这种结构允许我们根据需要增加更多的服务。如果我们需要添加更多的存储容量或计算能力，只需要添加 OSS 或客户端即可，不需要重新设计整个网络。

其次，该架构提供了高度的鲁棒性。每个节点都可以通过多个路径访问网络的其他部分，因此，如果一个路径出现问题，数据可以通过其他路径进行传输。这大大提高了网络的可靠性，保证了大语言模型训练中数据的稳定传输。

通过以上的设计，我们能够构建一个稳定、高性能、易于扩展的大语言模型存储结构。训练大语言模型需要处理大量的数据并需要稳定、及时的数据交换，这是由 Fat-Tree 网络拓扑架构及 BeeGFS 为我们提供可能。

3.3 负载均衡

BeeGFS 提供了条带化（Striping）的功能，可以将大文件分成多个块，不同的块存储在不同的存储节点上。当大型文件被分解存储在多个节点时，单个查询可以被多个节点同时处理。不同部分的数据可以并行读取，从而提高系统的数据读取速度。这有效地克服了单节点读取限制，从而显著增强了系统性能。这还为负载均衡创建了路径，允许平均分配资源，防止任何特定节点过载。

同时考虑使用 RDMA 网络可以有效地提高网络通信的性能，减少 CPU 间的数据交换延迟，从而实现计算节点间的负载均衡。RDMA 网络允许从一个网络中的计算机直接访问另一个网络中的计算机的内存，无需在远程系统上执行 CPU 和操作系统的技术。使用 RDMA，BeeGFS 可以通过 InfiniBand、iWARP 或 RoCE 网络实现高带宽和低延迟的数据传输，这对于并行读/写大型文件，如语言模型训练中的数据和模型文件，尤为重要。在启用了 RDMA 的网络中，数据路径的延迟大大降低，因为数据包不需要在每个网络交换点上被处理和复制。它直接从发送缓冲区复制到接收缓冲区，大大提高了数据交换的速度，降低了 CPU 的负载。

3.4 数据安全

在网络环境下进行数据分布式存储和传输，数据安全问题显得尤为重要。在下面的部分，我们将探讨一些保障 BeeGFS 系统数据安全的主要方法，以确保我们的数据在移动和静态状态下都能得到足够的保护。

- **数据冗余与故障恢复**：BeeGFS 支持对文件创建冗余副本（Replica），从而保证存储数据不会因为单个节点的故障而丢失。
- **数据保护和备份**：对于训练过程中生成的语言模型结果数据，需要按一定策略进行定期备份，以便进行模型的版本控制或故障恢复。
- **网络安全**：因为 BeeGFS 的数据在网络中进行分布式的存储和传输，所以需要考虑网络安全。可以采取合适的加密算法进行数据传输的加密以防止数据被截获或篡改。
- **权限管理**：可以通过设置权限管理来限制对文件的访问和操作，比如可以设置读权限、写权限和执行权限。这样可以防止未经授权的用户获取敏感的数据或者进行恶意的操作。
- **故障检测与恢复**：对于存储设备的故障，BeeGFS 可以通过监视和报告工具来进行实时的检测和及时的报告。在出现故障时，通过预先设定的故障恢复策略，可以快速将数据迁移到正常的设备上，防止数据丢失。

4 磁盘存储策略

4.1 存储媒质选取原因

从成本效益而言，磁盘存储（如 HDD 和 SSD）的硬件成本通常较低，特别是在大规模数据存储时具有明显的优势。此外，磁盘存储的功耗相对较低，尤其是 HDD，能够降低数据中心的能源消耗和散热成本，从而降低总体运营成本。

在存储容量方面，磁盘存储，特别是 HDD，提供了较高的存储容量，可以以较低的成本存储大规模数据集，这对于需要庞大数据集的 LLM 训练非常重要。同时，磁盘存储系统易于扩展，可以随着数据量的增加增加更多存储设备，而不需要大幅修改现有架构。

在数据持久性方面，磁盘存储适合长期数据存储，相较于 RAM 和其他易失性存储器，磁盘存储器的数据保留时间更长，这对于存储训练数据、模型参数和历史记录等重要数据至关重要。现代磁盘存储系统具备良好的数据保护机制，如 RAID 技术，能够在硬件故障时提供数据恢复能力，提高数据存储的可靠性。尽管磁盘存储的读写速度不如内存和 NVMe 快，但其性能足以应对许多 LLM 训练中的数据存取需求，尤其是对于不需要实时数据处理的批量处理任务。此外，结合内存和高速缓存，磁盘存储系统能够在一定程度上提高数据访问速度，满足训练过程中的频繁读写需求。

4.2 存储方案

本系统采用磁盘存储数据。磁盘存储的特点是不易丢失数据，可以长期保存，写入和访问能力主要取决于磁盘的读写性能，可以有效控制成本。磁盘主要分为 HDD（机械盘）和 SSD（固态盘），两者的 QPS 都在千级别，可以满足大规模语言模型训练和推理的数据存取需求。

4.2.1 多级存储架构

SSD 盘相比于 HDD 盘拥有更快的读写速度和更高的 QPS，能够更快地响应读写请求，大幅提高存储性能和系统效率，但 SSD 盘的造价远高于 HDD 盘，容量和寿命也较受限制，需要特别关注其容量适用范围和维护成本。

使用多级存储架构即使用快速、昂贵的存储（如 SSD/Flash）和慢速、便宜的存储（如磁盘 / HDD）的结合。为了在中等预算下实现高存储性能，可以使用 BeeGFS 的内置存储分层功能来自动将经常访问的数据块移到更快的存储上，而将不常访问的数据块移动到较慢但更便宜的存储上。

接下来我们考虑将数据分为热数据、温数据和冷数据，分别选择适合的存储媒质进行存储。

- **热数据：**
  - 训练数据：LLM 训练过程中的当前批次训练数据和频繁访问的数据，需

快速访问和高频读写。这些数据放在 SSD 盘中，以保证训练过程的高效性和快速响应。

- 模型参数：训练过程中生成的模型参数和需要快速访问的模型权重文件，放在 SSD 盘中以提高训练和推理的效率。

温数据：

- 历史训练数据：过去几轮训练使用的数据，虽然访问频率低于当前训练数据，但仍需较快的访问速度。将这些数据存储在 HDD 盘中，平衡性能和成本。

- 模型检查点：定期保存的模型检查点文件，用于模型恢复和调试，存储在 HDD 盘中以保持较好的读取性能，同时控制成本。

- **冷数据：**

  - 旧模型参数：长期保存的过往模型参数文件和权重文件，这些数据访问频率较低，但需要长期保存，以备后续分析和调试。将这些数据存储在 HDD 盘的冷存储区域中。

  - 历史日志：训练和推理过程中产生的历史日志文件，这些文件在生成后访问频率较低，但需要保留以供后续审计和分析。将这些日志文件存储在 HDD 盘的冷存储区域中。

4.2.2 条带化存储设计

条带化技术就是一种自动的将 I/O 的负载均衡到多个物理磁盘上的技术，条带化技术就是将一块连续的数据分成很多小部分并把他们分别存储到不同磁盘上去。这就能使多个进程同时访问数据的多个不同部分而不会造成磁盘冲突，而且在需要对这种数据进行顺序访问的时候可以获得最大程度上的 I/O 并行能力，从而获得非常好的性能。

大语言模型产生的语料库、词汇表以及训练过程中的模型数据等，都可能是大文件，直接存储可能会导致数据读取瓶颈。在这种情况下，适合条带化存储设计。将大文件分割为多个小块（也称之为条带），并将这些块分布在 BeeGFS 的多个存储节点（OSS）上。这样的设计使得对大文件的读写操作都可以并行进行，极大提高了 I/O 性能。

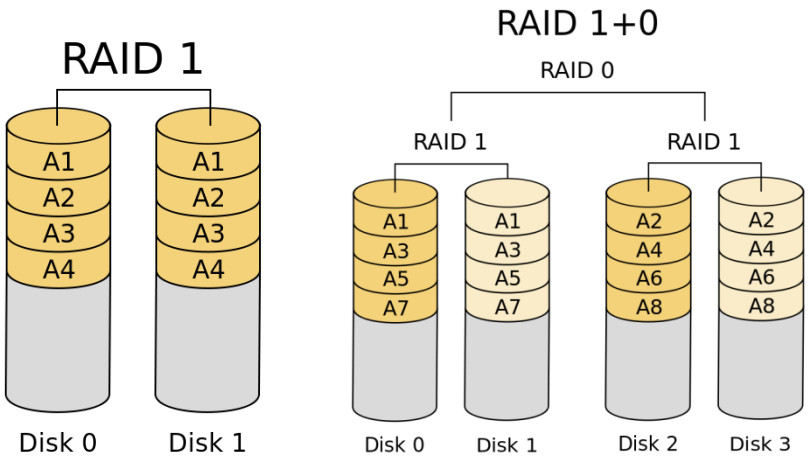
例如，当读取一个大规模语料库进行语言模型训练时，我们不需要在一个节点上顺序地读取整个语料库。而是可以将读取操作分布到多个节点上，每个节点读取文件的一个部分。这样，不仅可以提高数据读取速度，同时还可以利用多节点的计算能力并行处理语料库。

在设计存储架构时，我们可以根据文件的大小和访问模式来调整条带的大小，并选择合适的存储节点来存储这些条带。 BeeGFS 允许我们为每个文件或目录设置不同的条带化策略，这为优化大语言模型的数据访问提供了很大的灵活性。

4.3 RAID 级别的选择及解释

4.3.1 MetaData Target (MDT)

MDT 负责存储系统中的元数据信息，采用 RAID1 或 RAID10 进行 RAID 保护。RAID1 提供镜像备份，保证数据的高可用性和数据安全性，而 RAID10 结合了镜像和条带化，既提供高数据安全性，又能提升读写性能。相反，RAID5 和 RAID6 虽然可以提供容错能力并节省存储空间，但由于其在处理随机小 IO 操作时性能较差，不适用于存储元数据，会导致元数据性能严重下降。

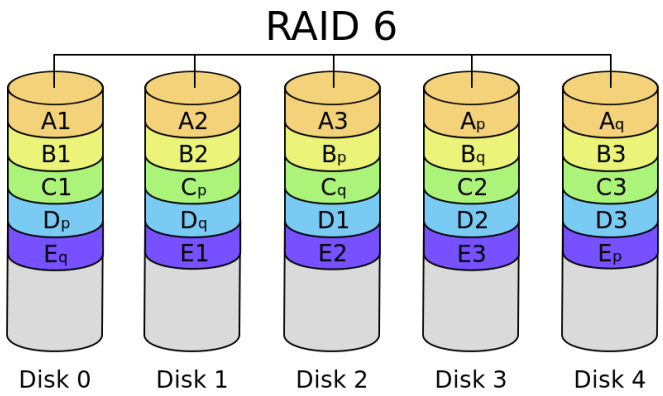


4.3.2 Object Storage Target (OST)

OSS 是存储文件内容的主要服务，每个 OSS 可以有一个或多个 Object Storage Targets (OST)。典型的 OST 配置包括 6 到 12 个硬盘，我们采用 RAID6 级别保护。

RAID6 在提供冗余保护的同时允许两个硬盘故障，因此非常适合大规模存储环境。与 MDT 不同，OST 上的 IO 通常是大顺序 IO，RAID6 的优势在于能够提供较好的读取性能和数据保护，尽管写入性能相对较低，但对于顺序 IO 的影响较小。此外，RAID6 的冗余特性在存储大量数据时能够显著提高数据的可靠性和系统的容错能力。

总之，MDS 采用 RAID1 或 RAID10 主要是为了提高小文件随机 IO 的性能和数据安全性，而 OSS 采用 RAID6 则是为了在大规模存储中提供有效的冗余保护和处理大顺序 IO 的能力。两者的 RAID 级别选择均根据其特定的工作负载需求和性能特点进行了优化。



5 数据清洗和处理

5.1 数据去重

对训练数据集进行彻底的审查，通过算法识别重复的文本片段、相同的用户输入或冗余的标记数据，并将其从数据集中移除。这有助于降低存储成本并提升数据处理速度。

5.2 数据格式化

实施标准化流程，将所有输入数据转换为统一格式。这包括文本的编码标准化、标点和特殊字符的统一处理，以及语言和方言的一致性。

5.3 数据清洗

- **噪声数据处理：** 采用自动化工具识别和清除数据中的噪声，如无关字符、乱码等。
- **异常值处理：** 通过统计分析识别异常值，并根据上下文决定是修正还是删除这些数据点。
- **缺失数据处理：** 对缺失值进行分析，采用插值、均值替换或模型预测等方法填补缺失数据。
- **停用词过滤：** 自动识别并过滤掉常见的停用词，减少对模型训练的干扰。
- **敏感信息处理：** 对个人身份信息等敏感数据进行脱敏处理，确保用户隐私。

5.4 数据验证和修复

- **数据完整性验证：** 定期检查数据集的完整性，确保所有必要的字段都已正确填写。
- **数据纠正和修复：** 对识别出的错误进行纠正，如拼写错误、格式错误等，并填补或修正缺失数据。

6 备份策略设计

6.1 数据分级

按照 80/20 原则，数据的重要性是有差异的。如果不加区分的恢复 100PB 级别的数据，无论是从成本上还是效率上都是无法承受的。大语言模型从垂直与水平两个方向对数据的优先级进行拆分：

6.1.1 垂直层面：数据重要性分级

核心数据与非核心数据的划分：

在大语言模型中，可以根据模型数据的关键程度将业务划分为核心和非核心。核心数据可能包括模型参数、关键训练数据等，这些数据对模型性能至关重要，通常需要更频繁的备份和更快速的恢复。

API 的分级管理：

对于对外提供的 API，可以进行分级管理，区分核心 API 和非核心 API。核心 API 可能涉及关键功能，需要更严格的监控和更频繁的备份。

6.1.2 水平层面：数据访问时效性分级

数据访问的时效性：

在大语言模型中，数据访问同样具有时效性。例如，最新的训练数据或实时的用户交互数据可能在短期内具有高访问频率。

访问热力值的评估：

可以采用类似访问热力值的概念，根据数据被访问的频率对数据进行分级。例如，最近一周内被频繁访问的数据可以被视为高优先级数据。

数据访问模式的分析：

通过对数据访问模式的分析，可以识别出访问频率低的旧数据，这些数据可能不需要频繁备份，或者可以迁移到成本更低的存储介质中。

6.2 备份频率与类别选择

- **完全备份：**

完全备份策略适用于大语言模型中最为关键的数据，如模型参数、核心训练集等。这些数据需要最完整的保护，以便在任何灾难情况下能够实现 100%的数据恢复。

- **增量备份：**

增量备份适用于频繁变更的数据，如日常的用户交互数据、日志文件等。由于增量备份只记录自上次备份以来发生变化的数据，因此可以节省存储空间并缩短备份时间。然而，恢复过程可能较为复杂，因为它需要依赖之前的备份记录。

- **差异备份：**

差异备份适用于定期更新的数据，比如模型的微调参数。差异备份策略备份自上次完全备份之后有变化的所有数据。它提供了一个恢复时间上的平衡点，因为它比完全备份更快，同时比增量备份更简单，因为它只需要两份数据：最后一次完全备份和最后一次差异备份。

结合备份策略：

鉴于大语言模型的数据规模和重要性，可以采取**完全备份与差异备份**相结合的方式。对于核心数据，进行定期的完全备份，并辅以差异备份以应对日常的数据变更。

备份频率：

- **核心数据：** 以天为单位进行完全备份，以小时为单位进行差异备份，确保在紧急情况下可以快速恢复。
- **非核心数据：** 随着数据热度和业务重要性的降低，可以减少备份频率，例如以周为单位进行完全备份，以天为单位进行差异备份。

事件驱动的备份调整：

在模型训练的关键阶段或数据更新频率显著增加的重大事件发生时，应提高备份频率，以小时为单位进行数据备份，以降低数据丢失的风险。

6.3 备份标准化与自动化

开发和实施一套标准化的备份流程，并集成自动化工具，确保备份过程的一致性和准确性。自动化工具应能与数据源无缝集成，实现实时或近实时备份。

自动化备份流程，并实施实时监控，以确保备份操作按计划执行，并在出现问题时及时警报。

通过这种综合备份策略，大语言模型可以确保关键数据的安全性和可恢复性，同时优化存储资源的使用，提高备份和恢复的效率。

6.4 321 备份策略

对核心数据实施 321 备份策略，即至少 3 个数据副本，存储在 2 种不同的物理位置和媒介上，以及 1 个离线备份。

大语言模型底层在数据备份上遵循 321 策略：

- 所有的数据备份都至少包含 2 个热备，2 个冷备；
- 在线热备数据存储在 SSD 设备，冷备数据存储在独立的 OSS 集群；
- 数据会在异地离线存储，通过专线进行数据同步与传输。

6.5 灾难恢复计划

制定详细的灾难恢复计划，包括：

- **恢复优先级：** 确定哪些数据和系统需要首先恢复。

- **RTO 和 RPO：** 设定恢复时间目标（RTO）和恢复点目标（RPO），以保证关键数据的快速恢复。
- **演练和测试：** 定期进行灾难恢复演练和备份数据的恢复测试，确保计划的有效性。

通过这些详细策略，大语言模型可以确保数据的高质量、高可用性和高安全性，同时在面对数据丢失或系统故障时，能够迅速恢复到正常运行状态。