基于云原生架构的异构存储目录更新机制设计

姬贵阳[1] 段国栋[1] 陈 培[1] 郑玉会[1]
JI Guiyang DUAN Guodong CHEN Pei ZHENG Yuhui

摘 要 随着人工智能技术的快速发展，文件存储面临数据量的爆发式增长，传统异构存储系统在管理海量文件时，暴露出目录更新缓慢、资源浪费严重等问题，难以满足集群高效稳定运行的需求。针对现有方法在海量目录信息更新方面存在的更新时间漫长等性能瓶颈，文章提出了一种基于云原生架构进行异构文件存储目录信息更新的方法。通过设计高效的目录数据结构、引入增量更新机制、识别热点目录并结合云原生技术构建分布式服务，实现目录信息的高效更新与实时管理。实验结果表明，该方案在提升存储系统响应速度、降低延迟方面显著优于现有技术，同时具备良好的可扩展性和易用性，其显著的高性能表现和较强的创新特性，使其在人工智能领域异构文件存储管理中极具借鉴与推广价值。

关键词 人工智能；异构文件存储；目录更新；增量方法；热点目录；云原生

doi：10.3969/j.issn.1672-9528.2025.11.046

0 引言

随着人工智能技术飞速发展，科研机构、高校与企业在云原生集群上存储了海量数据集和模型等文件，在异构文件存储环境下，海量目录的统计更新是一项重大挑战。具体表现为，云原生集群中的存储动态扩缩和频繁的文件操作导致目录信息迅速膨胀，在人工智能集群，一个月内可生成 TB 及以上规模数据。面对如此海量的目录数据，现有的更新方案存在以下不足，一方面，更新过程极为缓慢，需要耗费超长的时间才能完成；另一方面，在更新期间会持续占用大量的集群 CPU 和内存资源，严重影响了集群的正常运行效率。如何高效管理海量目录使用量与容量控制，已成为保障人工智能训练开发及推理高效稳定运行的基石。通过快速精准地获取用户、模型及数据集等目录的存储资源占用情况，不仅能确保集群系统任务稳健运行，更能显著降低企业存储运维成本，优化多个不同类型文件存储中海量目录管理的设计，具有极为重要的现实意义。

当前，存储目录管理方法存在一定的局限性，首先，依赖文件存储（GPFS、CephFS、BeeGFS 等）本身功能管理目录信息的方法 [1-4]，由于技术设计的局限性、异构环境的复杂性、性能和扩展性等问题，仅能获取单一存储、单一目录的信息，难以满足异构文件存储环境下多存储、跨层级目录信息整合的需求。其次，采用简单方法对存储进行遍历并更新目录信息 [5-6]，不仅效率低下，难以快速更新海量存储目录信息，还会过度占用集群环境中 CPU 和内存等资源，尤其在人工智能对资源要求苛刻的场景中，难以平衡目录管理需求和性能要求。

为解决上述问题，本文提出一种新的异构文件存储目录更新管理方案，该方案核心在于引入增量更新机制，仅对发生变化的目录进行更新，避免目前全量更新模式，通过实验验证，其效率提升显著。本文章节安排如下：首章为整体系统设计，构建方案整体框架，次章聚焦技术实现，深入剖析目录数据结构、增量更新方法、热点目录管理策略及分布式服务架构，末章着重于实验验证与结果分析，总结研究成果。

1 系统设计

在异构文件存储目录管理中，目录信息的高效存储与快速获取是实现集群存储业务控制的关键 [7]。本文围绕此关键问题，开展四项研究：目录数据存储结构设计、“自下而上”遍历方法、热点目录技术、目录管理服务，从而保证了存储系统的性能和可靠性，如图 1 所示。

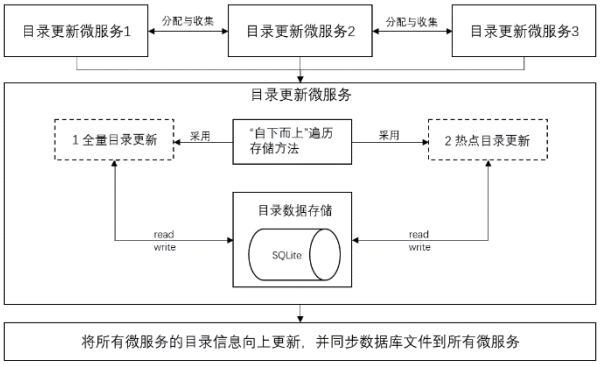

图 1 基于云原生中异构文件存储目录更新整体架构

1. 浪潮（北京）电子信息产业有限公司 北京 100095

设计围绕异构文件存储系统的目录管理方案，从以下四个方面进行深入优化。

一是为全面存储异构文件存储中所有目录的详细信息，详细设计了目录数据存储的数据表结构，并根据异构存储海量特性，进行分表动态管理，提升数据管理效率。

二是基于文件系统特性，当目录下一层文件或子目录发生增删改操作时，会触发该目录最近修改时间更新为当前时间的机制，使用栈和队列两种数据结构方法提出了一种高效的“自下而上”遍历算法，据此仅需针对变化的目录，进行增量更新，能够快速准确的获取目录大小信息。

三是为了实现常用目录信息的实时更新，识别热点目录，并对其进行单独更新管理，从而减少了等待目录更新时间，达到热点目录信息快速更新并实时获取目录信息的效果。

最后，设计基于云原生架构的分布式目录更新微服务，充分利用 Kubernetes 容器编排技术，将目录更新服务分解为多个可独立扩展的微服务实例，进行并发管理存储目录，从而实现高效更新目录信息的目的，显著提高系统的整体性能和稳定性。

2 技术实现

2.1 目录数据结构

为实现高效的存储目录管理，需将存储系统中所有目录信息（含目录大小、最近修改时间、拥有者等关键属性）存储于 SQLite 数据库 [8] 中。SQLite 作为一种文件型数据库，以其轻量级、体积小、便于备份与迁移的特性更加适用于人工智能领域。在集群平台中，采用多数据源接入方式，能够有效避免对业务数据库性能造成潜在的干扰。表 1 定义了存储所有目录信息的数据结构。

表 1 目录信息存储数据结构

字段名称	含义用途	示例说明
id	id 编号	1
path	目录（索引）	/dir1/dir2/dir3
last_modify_time	最近修改时间	1667288616000
size	目录大小	124 720
last_size	上次统计目录大小	124 720
owner	目录拥有者	root
level	目录层级	4
parent_path	目录的父目录（索引）	/dir1/dir2
update_time	记录更新时间	1667288616000

其中设计父目录字段可以进行批量查询子目录，目录拥有者字段可获取不同用户的目录信息，目录上次大小字段可以判断目录大小是否发生变化。为适配异构文件存储系统，支持多个不同类型文件存储，存储在平台不同的使用位置，对不同表的数据进行差异化动态管理。表 2 展示了针对异构文件存储环境中设计的分表结构，有助于分散数据库压力，提高数据库操作效率，以实现对各类文件存储资源高效管理。

表 2 异构文件存储分表存储数据结构

名称	含义用途	示例说明
table_name	异构文件存储表名	storageFile_*
storage_path	异构文件存储根路径	/dir1/dir2

2.2 “自下而上”方法

在云原生集群存储的目录管理中，采用了一种结合栈与队列数据结构的遍历策略 [11]，以实现高效的信息更新。具体而言，首先将根目录压入栈中，随后依次执行出栈并入队列的操作。通过不断地出栈和出队列操作，能够确保目录遍历方式按照“自下而上”[12] 的思路进行，从而高效完成对云原生集群存储中所有目录的遍历与信息更新。

在出栈过程中，若在出栈过程中发现目录在底层存储不存在，则会触发数据库中目录删除操作，递归地删除该目录及其所有的子目录，以确保存储与数据库的信息一致性。若数据库目录表中不存在当前处理的目录，则将存储于该目录下的一层子目录依次压入栈中。随后出栈过程中判断底层存储和数据库中目录的最近修改时间是否一致，若检查到发生变化，则表明该目录可能删除子目录或包含新的子目录，需将存储中该目录的一层子目录依次压入栈中。相反，若检查到未发生变化，则表明该目录下结构相对稳定，此时可直接将数据库中记录的该目录的一层子目录信息压入栈中，以减少不必要的存储访问操作，提高处理效率，加速实现“向下”遍历过程。

在入队列过程中，若出栈的目录无子目录，即为叶子目录，则检查其最近修改时间是否发生变化，若发生变化，需将其加入队列，若未发生变化，则无需入队列。同时，若目录的所有子目录（包括孙子目录）均已入队列且均未发生变化，则该目录无需入队列；反之，若其所有子目录中存在某个目录变化，则该目录需入队列。在整个过程中，持续进行出队列操作，直至根目录出队列且队列为空，加速实现“向上”更新过程。

在遍历过程中，出栈和出队列操作并行执行。若栈中存在数据时，进行出栈操作，若队列中有数据，则执行出队列操作。整个过程持续进行，直至栈和队列均为空时结束。整个“自下而上”增量方法如图 2 所示。

在判断目录是否发生变化时，若文件持续被特殊命令操作，例如通过 echo 命令操作文件内容，可能导致基于最近修改时间的变化更新目录信息产生误差。为解决这一特殊场景，

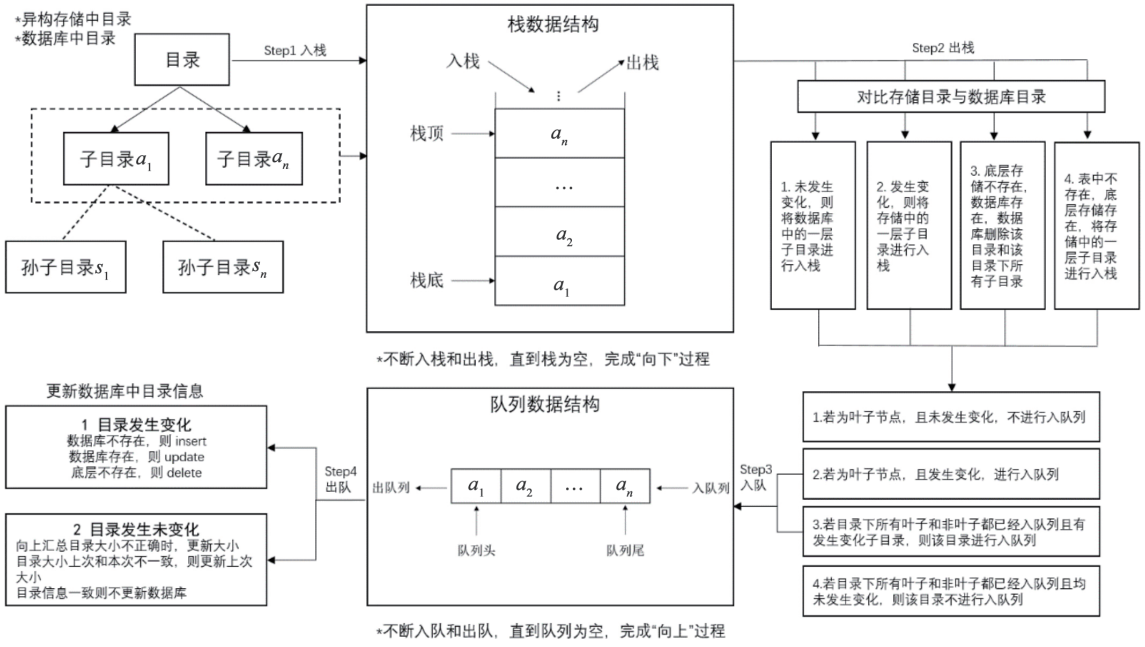

图 2 “自下而上”方法流程

对此方法进行改进，遍历目录时，依据以下规则判定文件是否持续发生变化：

（1）文件的最近修改时间等于其父目录的最近修改时间，且当前时间与文件的最近修改时间的差在 2 s 内。

（2）文件的最近修改时间大于其父目录的最近修改时间，且当前时间与文件的最近修改时间的差在 2 min 内。以上满足一条，则判定该文件持续发生变化。

$$f=\begin{cases}t_{\text{lastmodifytime}}(f)=t_{\text{lastmodifytime}}(d)\\ t_{\text{now}}-t_{\text{lastmodifytime}}(f)\le 2s\end{cases} \tag{1}$$

$$f=\begin{cases}t_{\text{lastmodifytime}}(f)>t_{\text{lastmodifytime}}(d)\\ t_{\text{now}}-t_{\text{lastmodifytime}}(f)\le 2\text{min}\end{cases} \tag{2}$$

式中：f 为持续变化的文件；t_{now} 为当前遍历的系统时间；$t_{\text{lastmodifytime}}(f)$ 为文件的最近修改时间；$t_{\text{lastmodifytime}}(d)$ 为其父目录的最近修改时间。

在认定文件持续发生变化后，需要更新数据库中其父目录的最近修改时间为当前时间。通过这种方式，确保在下次任务更新该目录时，对比存储目录信息会认定该目录发生了变化，目录大小会再次被统计更新，保障目录信息的准确性。

2.3 热点目录更新

在异构文件存储管理中，采用“自下而上”方法将所有目录信息存入 SQLite 数据库。当存储规模达千万级以上目录时，全量定时遍历更新耗时较长。为此，提出一种热点目录识别策略，具体来说，在设定的时间间隔范围内，若目录下文件或子目录发生增删改操作，导致该目录的最近修改时间处于这个时间间隔内，则将该目录识别为热点目录，通过监测目录最近修改时间的变化范围来识别热点目录。例如，若设置阈值变更范围为 1 天，则获取在最近一天内发生变化的目录，则将其判定为热点目录，并对此类目录进行单独的遍历更新，避免等待全量存储目录更新的时间。在云原生人工智能集群存储的实际应用场景中，1 天的热点目录变化量通常仅占总目录数的 0.05%~0.1%，即 100 万个目录中约有 500~1 000 个热点目录。依据数据库中存储的目录最近修改时间（last_modify_time）来识别热点目录，这些目录能够从数据表中被迅速获取。热点目录的获取公式为：

$$H=\left\{d\in D\mid t_{\text{now}}-t_{\text{lastmodifytime}}(d)\le \Delta t\right\} \tag{3}$$

式中：H 为热点目录集合；D 为存储系统中所有目录的集合；d 代表集合 D 中的任意一个目录；$t_{\text{lastmodifytime}}(d)$ 为目录 d 的最近修改时间；Δt 为系统设定的时间阈值。

在热点目录识别过程中，系统设置的时间阈值起到关键作用，该阈值以天为单位，例如，若将一天内发生变化的目录判定为热点目录，则阈值设为 1，阈值大小直接影响热点目录的识别精度，阈值越大，识别出的热点目录数量越多，识别结果越准确。在后续存储遍历时，仅需针对这些热点目录进行遍历，就能够更快速精准地更新存储目录信息。

对于识别出的热点目录，仅需根据其下一层子文件与子目录的信息进行更新，无需深入递归遍历子目录。若发现热点目录及其子目录在底层存储系统中不存在，则应从数据库中删除不存在的目录及其全部子目录。当热点目录的子目录在数据库中缺失对应记录时，需采用“自下而上”方法进行数据同步。若热点目录本身发生变化，其大小应重新计算，具体计算方式：一层子文件的大小总和，加上数据库中记录

的一层子目录的大小总和。更新完成后，还需向上更新热点目录的所有父目录的信息，从而确保整个异构文件存储系统的目录信息准确无误，通过这种方法，可以有效减少更新操作的总体耗时，显著提升存储管理效率。

2.4 分布式目录更新

在云原生 Kubernetes 集群中，采用多节点协作的目录管理策略 [9]。根据异构文件存储的目录结构对其进行划分，并将目录更新任务交由多个目录管理微服务（Docker 容器）共同完成遍历更新 [10]。具体而言，若统计的目录树相互独立，则将其交予其他空闲的目录管理微服务进行遍历更新。若统计的目录树为一棵独立目录树，则把其子目录树分配给空闲的目录管理微服务进行更新。更新任务分配时，应优先选择独立子目录树，再依据分布式架构的需求分配子目录树。其中独立树和子目录树如图 3 所示。

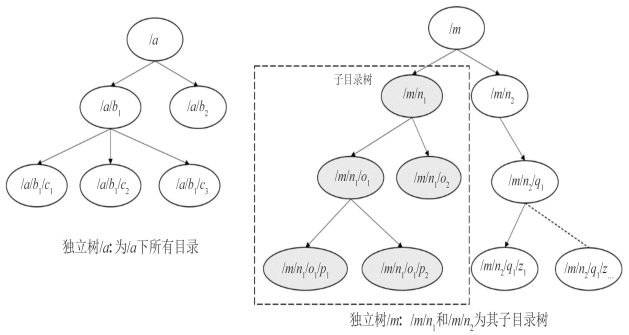

图 3 独立树与子目录树

各目录管理微服务基于异构文件存储的目录结构进行动态任务分配，将微服务状态划分为空闲、繁忙和故障，仅当微服务处于空闲状态时，才会被分配到新的目录更新任务，同时将状态切换为繁忙。本文的目录划分方案通常分配次数控制在十次以内即可完成所有目录的遍历与更新，微服务会记录被分配任务的目录信息、分配节点、运行节点和任务状态。所有微服务均采用“自下而上”的增量方法，高效更新目录的大小、拥有者、时间等详细信息，并将更新结果暂存于各自服务的数据库中。

在任务执行过程中，云原生监控组件实时监测微服务及节点状态，若检测到服务或服务所在节点异常，则将相应服务的状态更新为故障。对于出现故障的微服务，系统会依据集群中微服务的实时数量和状态信息，自动将未完成的任务重新分配给其他空闲状态的微服务，实现动态负载均衡，确保分配任务顺利完成。所有微服务目录更新任务完成后，由专门的分配服务汇总各微服务的更新结果，统一“向上”更新目录大小等信息，确保整体数据准确一致。最后，同步所有微服务数据库，保证所有数据库信息一致，为下一次定时更新任务做好数据准备，图 4 为详细流程图。

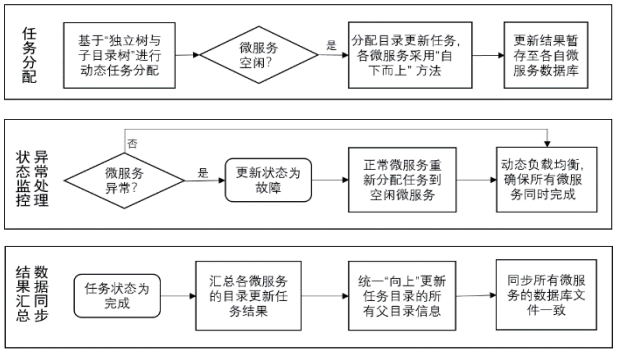

图 4 分布式目录管理流程图

3 实验验证

本文的实验环境硬件配置：由两台服务器构成，每台服务器的操作系统为 Ubuntu22.04，服务器搭载双路 CPU，配备内存 128 GB，其中一台服务器采用 SSD 磁盘存储，与系统所在磁盘相互独立，总容量为 101.2 TB，该磁盘通过千兆以太网，采用 NFS 协议挂载至另一台服务器，构建跨网络集群共享文件存储，以实现跨网络环境下的目录更新验证。在该实验环境中部署了 Kubernetes 和 Docker，用于容器编排与管理，以支持分布式应用的部署和运行，在每个目录更新服务中配置 jdk1.8，采用 Java 语言进行运行测试。该环境主要用于验证本文提出的存储目录更新设计方案，主要针对单一存储场景，对比普通全量遍历与“自下而上”增量遍历和分布式方案的效果。鉴于复杂异构文件存储中部分测试逻辑与单一存储类似，本次实验未涵盖复杂异构文件存储场景。

本次实验重点对比分析了两种核心方法：一是“自下而上”增量方法，二是分布式微服务方案。通过模拟人工智能集群训练与推理的数据场景，构建了使用量包含约 0.5、1、5、15、30、50 及 100 TB 的多种数据规模测试集，以验证本设计在海量异构目录管理中的性能。需指出，热点目录方法因其具有实时更新目录信息的特性，在一个更新周期内可实时完成，在本次实验中不具备可比性，故未对其进行罗列对比。实验过程中，每种场景重复测试 3 次，取平均值作为性能评估指标，重点对比了本文两种方案的时间消耗情况。

实验结果表明，随着存储使用量的增加，本文提出的优化设计展现出卓越的性能优势，详细实验数据参考图 5 和图 6。通过对实验结果的分析可知，相较于传统遍历方法，优化方案在处理大规模存储目录时，遍历更新目录速度实现 20 倍以上提升。以 100 TB 使用量的存储目录更新为例，传统方法需耗费约 2 天时间，而本优化方法仅耗时 2 h，分布式方案更是将耗时压缩至不足 1 h，所需耗时则大幅缩短，这一效率的提升直接导致在相同时间周期内，目录信息的更新频率显著增多，进而提升了存储目录的管控能力。实验数据清晰地反映出，优化设计在极大地提高了目录更新的效率的同时，优化了集群资源的整体利用率，这种改进不仅有助于减轻集群的资源消耗负担，还提升了存储的高效性和稳定性。

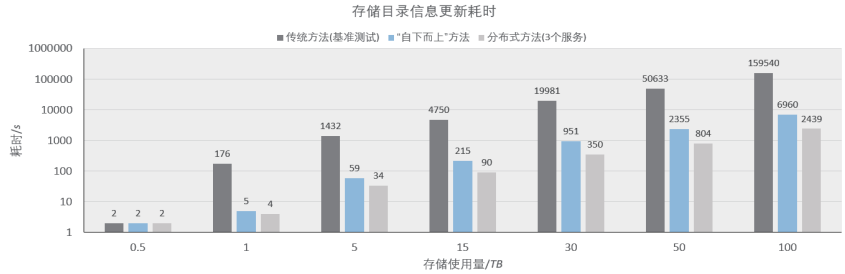

图 5 遍历更新目录方案耗时对比

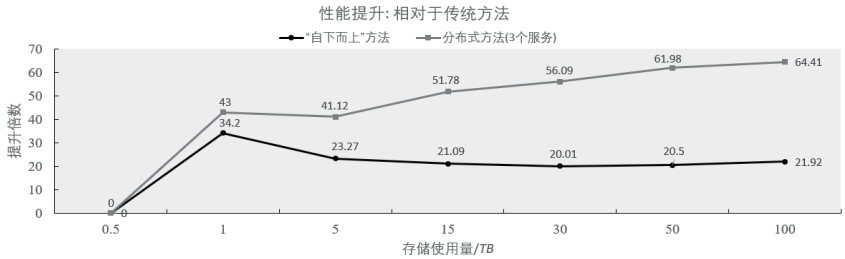

图 6 遍历更新目录方案性能对比

4 结论与展望

基于 Kubernetes 环境下的异构文件存储目录管理难题研究中，创新性地提出了一套综合解决方案，包括目录信息存储结构的设计、“自下而上”增量遍历方法、热点目录管理策略以及分布式目录管理服务的构建，实现了存储目录信息的高效更新。实验结果表明，该方案相较于传统方法实现了质的飞跃，以亿级目录数的测试为例，与现有存储目录更新方案对比，目录信息更新速度实现倍数级提升，CPU 占用时长也是倍数级减少。这一技术创新不仅大幅提高了存储目录更新效率，还有效降低了系统资源消耗，展现出卓越的性能表现，可移植应用在人工智能集群和大数据平台，提高研发工作效率，降低企业运维成本，满足未来存储数据高速膨胀的需求，未来研究重点将聚焦于异构存储系统监控，旨在减少目录状态比对过程。综上所述，本文提出的优化设计方案在提升目录更新效率具有显著优势，对存储目录精细化管理的发展具有重要的实践意义，为未来大规模数据存储环境中的目录管理提供了新的技术思路和解决方案，有望推动相关领域的技术进步。

参考文献：

[1]WEIL S A, BRANDT S A, MILLER E L,et al.Ceph: a scalable, high-performance distributed file system[C]//Symposium on Operating Systems Design & Implementation.USA: USENIX,2006:307-320.

[2]SCHMUCK F, HASKIN R. GPFS: a shared-disk file system for large computing clusters[C]//Proceedings of the FAST'02: Conference on File and Storage Technologies. USA: USENIX, 2002: 231-244.

[3]BRZENSKI J, PAOLINI C, CASTILLO J E.Improving the I/O of large geophysical models using PnetCDF and BeeGFS[J]. Parallel computing, 2021,104-105: 102786.

[4] ROCH L M, ALEKSIEV T, MURRI R, et al.Performance analysis of open-source distributed file systems for practical large-scale molecular ab initio, density functional theory, and GW + BSE calculations[EB/OL].(2018-01-05)[2025-10-23]. https://doi.org/10.1002/qua.25392.

[5]CALDERBANK A R,COFFMAN E G, FLATTO L. Optimal directory placement on disk storage devices[J].Journal of the ACM (JACM), 1988, 35(2): 433-446.

[6]WANG Y Z, YANG F K, ZHOU K, et al. An optimized learning-based directory placement policy with two-rounds selection in distributed file systems[J]. Future generation computer systems, 2024, 154:235-250.

[7]BRANDT S A, XUE L, MILLER E L, et al. Efficient metadata management in large distributed file systems[C]//20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003. (MSST 2003).Piscataway: IEEE, 2003: 290-298.

[8]BI C Y. Research and application of SQLite embedded database technology[J]. WSEAS transactions on computers, 2009, 8(1): 83-92.

[9]HEIDARI S M, PAZNIKOV A A.Multipurpose cloud-based compiler based on microservice architecture and container orchestration[J].Symmetry,2022,14(9):1818.

[10]BERNSTEIN D. Containers and cloud: from LXC to Docker to Kubernetes[J].IEEE cloud computing, 2014, 1(3): 81-84.

[11] 殷人昆 . 数据结构 : 用面向对象方法与 C++ 语言描述 [M]. 北京 : 清华大学出版社 , 2007.

[12] 苏州元脑智能科技有限公司 . 一种存储目录统计方法、装置及电子设备和存储介质 : CN202311483331.0[P]. 2024-02-13.

【作者简介】

姬贵阳（1991—），男，河南鹤壁人，硕士，工程师，研究方向：人工智能平台。

段国栋（1982—），男，内蒙古呼和浩特人，本科，高级工程师，研究方向：人工智能平台。

陈培（1982—），男，河南郑州人，博士，高级工程师，研究方向：深度学习算法、人工智能平台。

郑玉会（1993—），女，河南安阳人，硕士，工程师，研究方向：人工智能平台。

（收稿日期：2025-05-26 修回日期：2025-11-13）