文章编号:1006-2467(2024)10-1629-08 DOI: 10.16183/j.cnki.jsjtu.2024.099

创新设计

医疗 PACS 影像系统的数据存储性能优化

尤丽珏[1], 焦圣品[2], 李小勇[3]
(1. 复旦大学附属华东医院 计算机中心,上海 200040; 2. 上海霄云信息科技有限公司,上海 200240; 3. 上海交通大学 电子信息与电气工程学院,上海 200240)

摘 要: 医疗影像归档与通信系统(PACS)影像数据是典型的海量小文件场景,其面临的挑战主要为高效的海量元数据管理和有效解决碎片化导致的性能下降.通过分析医疗 PACS 影像存储系统路径的全 IO(input/output)各个关键环节,从 PACS 软件调阅算法、存储协议网关高并发设计、小文件聚合以及数据存储服务并发模型 4 个维度进行优化设计,实现对 PACS 调阅性能的大幅提升.实际测试表明,优化设计后 PACS 影像系统的调阅性能可达到每秒 300 幅图像,为传统存储调阅速度的 3 倍以上,有效解决了 PACS 影像数据调阅卡顿的问题.

关键词: 海量小文件;PACS 影像;分布式存储;性能优化
中图分类号: TP391 **文献标志码**: A

Optimization of Data Storage Performance in Medical PACS Imaging System

YOU Lijue[1], *JIAO Shengpin*[2], *LI Xiaoyong*[3]
(1. Huadong Hospital Affiliated to Fudan University, Shanghai 200040, China; 2. Shanghai Xiaoyun Info Tech Co., Ltd., Shanghai 200240, China; 3. School of Electronic and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: Medical picture archiving and communication system (PACS) is a typical application scenario with massive small files, which faces two challenges in data storage, i.e., efficient metadata management and effective performance reduction caused by fragmentization. By analyzing various key components of the full IO (input/output) path in the medical PACS imaging system, this paper optimizes the design to achieve a significant improvement in the retrieval performance of the PACS from four dimensions, PACS software retrieval algorithm, storage protocol gateway high-concurrency design, small file aggregation, and data storage service concurrency model. The results of test show that the retrieval performance after optimizion can reach 300 images per second, which is more than three times that of traditional storage, and resolves effectively the performance problem of PACS image data retrieval.

Keywords: massive small files; picture archiving and communication system (PACS) image; distributed storage; performance optimization

收稿日期:2024-03-21 **修回日期**:2024-06-07 **录用日期**:2024-06-28
作者简介:尤丽珏(1975—),高级工程师,从事医院数字化研究.
通信作者:李小勇,副教授;E-mail:xiaoyongli@sjtu.edu.cn.

数字化医疗影像设备已经成为医院不可或缺的基础设施，是医生获取诊断依据的重要来源. 随着医疗数字化成像技术的发展，医疗影像设备的成像分辨率从 64 排提升到 256 排，设备扫描切层分辨率从 5 mm 提升至 1 mm，一次检查可产生 300～10 000 张图片；同时各大医院的数字化医疗影像设备的数量也在逐年增加，医疗影像数据量显著增长. 以千字节计医疗影像数据大小从几十到几百不等，医疗影像归档与通信系统（picture archiving and communication system，PACS）属于典型的海量小文件场景，一家大型三甲医院 PACS 数据量可达 300～500 TB，文件数量高达十亿级别，这对存储系统的性能和可扩展性提出了更高要求[1-2].

针对海量小文件的存储优化问题，目前已有大量的研究工作. 基于 HDFS（Hadoop distributed file system）[3]，曾梦等[4]提出一种基于双层哈希编码和 HBase[5] 的海量小文件存储优化方法，通过构建哈希索引的方式提高文件读取效率；基于合并和预取的思想，文献[6-8]根据不同应用场景提出了相应的海量小文件性能优化方法；Aggarwal 等[9]从内存占用、节点间通信、数据处理开销以及跨节点数据传输等方面综合分析影响小文件访问效率的原因，并给出了相应的处理策略. Tao 等[10]提出的 LHF（linear hashing file）可用于加速确定小文件在聚合大文件中的位置；基于 LFN（logic file name）的概念[11]，并提出 SMSBL（small file merge strategy based LFN），可有效提高相同块中小文件的关联性，配合预取机制显著提升了小文件的读写性能. 基于分布式存储系统 Ceph，陈法河等[12]通过提取小文件的关联性实施对小文件的合并，可以减少合并过程中内存损耗，提供数据预取的命中率；Shi 等[13]扩展了 Ceph 的小文件处理架构，通过数据去重合并的方式降低数据处理开销，同时利用预取和文件索引的手段提高小文件访问效率. Beaver 等[14]针对海量图片存储的场景通过优化元数据以提高缓存利用率，从而提升系统整体吞吐率. 同样地，iFlatLFS（flat lightweight file system）基于优化元数据的思路，同时采用扁平化存储架构以实现数据访问流程的简化，从而提升海量小文件的访问性能[15]. 综上所述，目前已有工作主要从去重合并、缓存预取和读写流程优化三方面切入提升海量小文件进行存取效率. 但是，在小文件合并后，频繁的数据修改和删除会导致存储空间的碎片化问题凸显，从而影响数据的读写速率，本文在设计小文件合并算法时增加了聚合对象的管理操作，有效解决了上述问题.

为有效解决数据压缩、删除和重写等操作引起的空洞和存储空间碎片化问题，通过聚焦医疗 PACS 的业务流程，从 PACS 调阅软件、协议网关转发、数据存储服务和海量小文件存储 4 个方面进行调阅流程全链路优化，特别在海量小文件存储方面，结合连续与链式分配相结合的方式，实现对聚合对象的动态空间规划与管理. 实际测试证实优化后的医疗 PACS 影像系统调阅性能有效提升，可解决 PACS 调阅卡顿的问题.

1 整体优化设计

医疗 PACS 典型架构包括多种医疗影像检查设备、PACS 服务器及其数据库、PACS 调阅软件以及存储系统，如图 1 所示. 患者检查产生的影像数据由医疗影像检查设备通过 DICOM（digital imaging and communications in medicine）协议发送给 PACS 服务器，然后 PACS 服务器将影像数据写入数据存储设备，并将数据存储路径记录到 PACS 数据库中. 最后，医生调阅终端软件时根据 PACS 数据库中记录的数据存储路径向数据存储设备请求调阅所需数据.

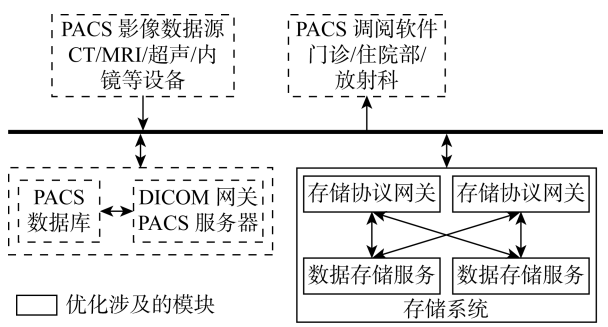

图 1 医疗 PACS 系统典型架构

Fig. 1 Typical architecture of medical PACS system

在传统 PACS 系统架构中，为了解决性能和海量数据之间的矛盾，存储系统采用在线、近线、离线三级架构进行构建，在线存储中只保存少量数据，以实现较高的性能. 随着医疗影像数字化技术的发展，医学影像的切片密度和分辨率提升，患者每次检查生成的数据量也随之增加，使得医疗 PACS 影像数据的增长速度进一步提高；另一方面，随着人工智能（AI）技术在医疗 PACS 领域的应用和普及，AI 模型的训练需要对历史数据进行高频次访问，并且患者的长期随诊也会导致较高频次的读取历史数据，因此数据全部在线更符合医疗 PACS 应用场景的存储需求. PACS 应用需求的变化使得传统三级存储架构在灵活扩展和海量小文件访问性能方面的发

展瓶颈愈发明显.

考虑传统 PACS 影像存储系统在高并发处理能力、扩展性以及海量小文件存储性能的稳定性等方面难以满足要求，通过对医院 PACS 中数据调阅流程的关键环节进行分析和多层面优化设计. 在 PACS 调阅软件层面设计实现用户行为感知的文件调阅算法，使数据读取实时响应用户行为变化，保证关键影像数据被优先调取；在存储系统内部的存储协议网关层，根据协议命令交互特点设置不同优先等级的处理队列，提高协议交互的流畅性和并发度；在存储系统的并发模型方面，数据存储端服务采用多进程与事件驱动相结合的方式，完成数据的异步 IO (input/output) 提交，不仅充分利用了多核 CPU 的并行处理能力，并且实现了存储系统中多个硬盘的 IO 并发. 同时，针对 PACS 海量小文件的特点在存储系统端设计实现小文件聚合功能，在存储文件数达到十亿级以上时仍然能够保证性能的稳定性. 在 PACS 影像数据的压缩、删除和重写等操作中，聚合对象将带来空洞和存储空间碎片化的问题，因此，通过连续与链式分配相结合的方式实现对聚合对象的空间规划与管理，保证存储空间的动态回收和再利用.

2　用户行为感知的文件调度算法

一次医学影像检查可能会产生几百到几千张 DICOM 格式的影像文件，根据图像位置和角度的差异分为不同的序列，以序列实例号区分，其中包含的关键序列和关键帧是医生在调阅诊断时优先关注的内容. 基于该特点，设计用户行为感知的文件调阅算法，实现对医生关注的序列文件进行优先调取. 以下从数据结构和执行逻辑两个方面对该算法进行介绍.

图 2 为该算法中 PACS 影像数据调阅任务的数据结构. 整体而言，该数据结构是一个二维链表，其中每个序列中多个影像数据的调阅任务构成一个一维链表，并以影像数据的 SOP (service-object pair) 实例号为索引构建每个调阅任务的随机访问结构，然后将每个序列的调阅任务链表作为节点，链接构成第二个维度的链表，称为序列链表，并以每个序列实例号为索引实现链表的随机访问.

医生每次调阅某位患者的医疗影像数据时，PACS 软件会在内存中构建一个调阅任务的二维链表结构，并以患者信息作为索引进行查询. 该二维链表需要根据医生的调阅行为进行相应调整，以保证所需序列及相应序列下所需图像数据的调阅任务得

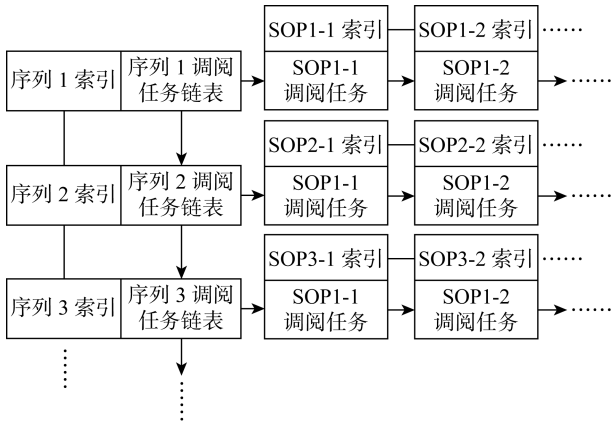

图 2　PACS 影像文件调阅任务的数据结构

Fig. 2　Task data structure of retrieving PACS image file

到优先处理. 具体地，当医生选择优先调阅某个序列的指定影像数据时，其指定的序列实例号和影像数据的 SOP 实例号会被 PACS 软件获取. 首先，PACS 软件以该序列实例号为索引查找到相应序列的调阅任务链表，将该调阅任务链表整个移动至序列链表的头部；然后，以获取的影像数据 SOP 实例号为索引，在相应序列的调阅任务链表中查找到相应影像数据的调阅任务，并将该调阅任务移动至该序列调阅任务链表的头部. 至此，根据医生调阅需求，PACS 软件完成了对调阅任务内存结构的调整.

对于每个二维调阅任务链表，PACS 软件为其构建一组对等的线程执行影像数据调取任务. 所述线程在序列链表的维度检查获取非空的调阅任务链表，从中取出头部的调阅任务完成数据的调取，关于数据调取时采用的存储协议可以根据实际业务需求而定.

3　存储协议网关优化设计

存储协议网关也是 PACS 影像数据调阅过程中的关键 IO 环节，其并发处理能力和转发效率关系到数据调阅性能的高低，因此这一环节的优化设计从并发模型、优先队列和数据端口复用 3 个方面进行，以实现单节点协议网关支持 2 000 以上的 TCP (transmission control protocol) 并发连接. 在案例的实际应用场景中，PACS 软件采用 FTP (file transfer protocol) 协议与后端存储系统进行数据交互，因此以 FTP 协议为例对协议网关优化设计进行介绍.

目前，对于多数 FTP 协议网关的开源实现，为了支持多并发访问，一般采用一个进程服务一个客户端的方式来实现. 当 FTP 协议网关的主进程收到一个客户端的控制连接建立请求时，主进程创建一

个子进程与该客户端完成后续的协议交互过程，在一个进程空间内只会存在一个控制连接和一个数据连接，二者对应关系简单清晰. 但是，这种方式使得服务进程与控制连接成为一对一的关系，在 PACS 影像数据调阅这类高并发访问的应用场景下，服务端进程数不断膨胀，而进程的创建和调度切换对操作系统而言都是开销较大的操作，在消耗服务端操作系统资源的情况下也难以提供稳定可靠的高并发访问需求.

3.1 多线程并发模型设计

针对已有 FTP 网关在支持高并发访问时的问题，采用多进程与多线程结合的方式重新设计实现了 FTP 协议网关，多个进程同时侦听 FTP 协议端口，接收客户端的 FTP 协议请求. 在每个进程内部，采用多线程模型实现协议请求的分发和处理，如图 3 所示. 不同线程处理来自同一个控制连接的协议请求会导致请求执行时序的错乱，从而影响协议交互语义的正确性. 因此在进行协议请求分发时，以控制连接的文件描述符（file descriptor，fd）为标识信息，对工作线程数取余确定控制连接与工作线程的对应关系，保证同一个控制连接的请求只会交给同一个线程处理.

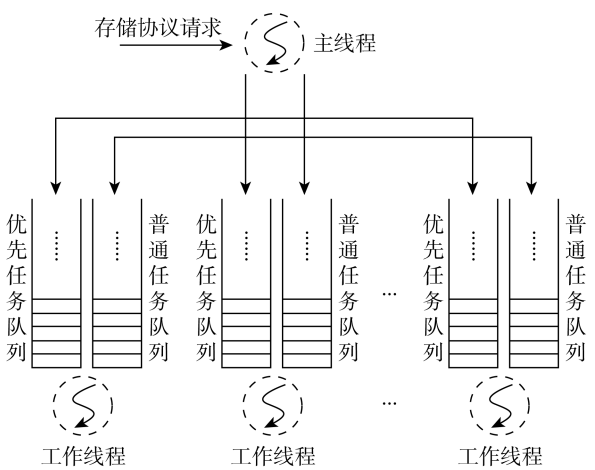

图 3 FTP 协议网关的线程模型

Fig. 3 Threading model of FTP protocol gateway

3.2 优先队列设计

在 FTP 协议交互的逻辑中，控制连接用来发送协议命令如 PASV、PORT、STOR、RETR、MKD、LIST 等，其中 STOR、RETR 和 LIST 等命令需要建立数据连接，用来传输文件数据和目录列举的条目数据，因此在客户端发送这些命令之前，协议规定必须要先通过 PASV 或 PORT 命令建立数据连接，所需数据通过数据连接完成传输. 该流程在一个进程服务一个客户端的情况下没有问题，但采用多线程模型的情况下，一个工作线程同时服务多个控制连接，因此，多个控制连接的协议请求需要在任务队列中等待工作线程依次执行. 当队列中出现一个大文件读写的请求时，PASV 和 PORT 这类建立数据连接的请求会阻塞较长时间，在高并发场景下会导致数据连接建立失败. 针对该问题，为每个工作线程定义一个优先任务队列，接收 PASV 和 PORT 协议请求，工作线程将优先处理该队列中的请求，避免耗时较长的任务阻塞 PASV 和 PORT 请求的执行而导致数据连接建立失败.

3.3 TCP 端口复用

在数据连接端口的分配上，协议网关服务可以将同一个 TCP 端口分配给不同的客户端进行数据连接的建立. 基于该理论，可在处理数据连接端口的分配上进行精细化控制，在每个服务进程内部维护客户端 IP 地址与已分配给该客户端的数据端口的映射关系，在执行 PASV 请求分配数据端口时，以客户端 IP 地址作为端口资源的分配依据，保证同一客户端不会被分配到已经使用的数据端口，同时针对同一个数据端口可以分配给不同的客户端，从而达到充分利用端口资源的目的.

4 小文件聚合优化

海量小文件的读写性能是存储领域具有挑战性的长期技术问题. 传统文件系统因其复杂的元数据管理结构导致在应对海量小文件的元数据访问时效率低下，并且庞大的元数据条目缓存在提升访问效率的同时难以保证性能稳定性；另外，海量小文件的修改删除操作会导致小段数据随机散落在硬盘的不同位置，从而造成存储空间的碎片化，不仅降低数据访问性能，而且造成存储空间的浪费. 针对以上问题，通过综合分析目前小文件合并方案的优劣性，重新设计实现了小文件聚合方案，以进一步提升存储系统的可管理文件数，减少文件系统碎片化程度，保证在存储小文件数达到百亿级时写入性能的稳定.

根据聚合对象分配时机的不同可以将小文件聚合分为两种实现方式. 第一种如图 4(a)所示，针对所有小文件写入，首先以原始数据信息写入内存或固态盘（solid state drive，SSD）等高速存储介质，不进行聚合，然后由系统后台服务遍历所有小文件数据，分配聚合对象，将热数据之外的小文件合并为聚合对象，写入机械盘（hard disk drive，HDD）等低速存储介质，并更新小文件元数据信息，记录小文件与所属聚合对象的对应关系. 第二种如图 4(b)所示，在小文件写入的流程中，增加聚合对象的分配操

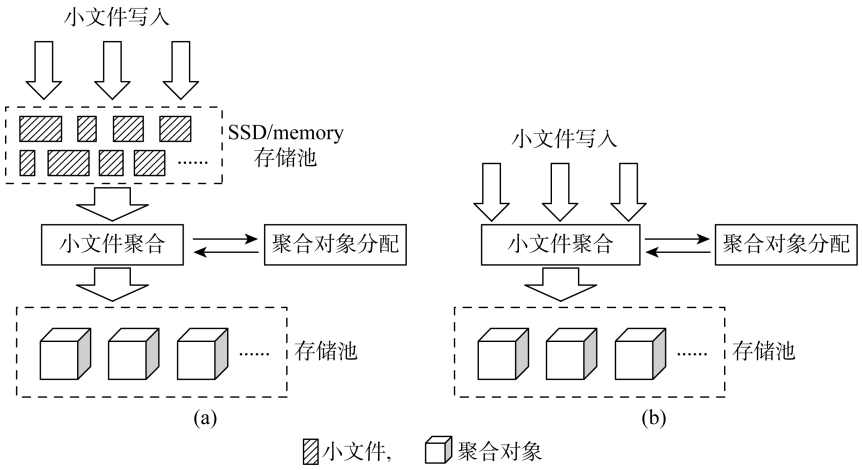

图 4　小文件聚合的两种实现方式

Fig. 4　Two implement methods for small file aggregation

作，确定小文件与聚合对象的对应关系后，直接将小文件写入聚合对象中相应的数据范围，并将该对应关系记录在小文件的元数据中.

第一种实现方式与冷数据打包归档的思路较为相似，所有小文件在聚合之前都独立存储于 SSD 缓存池，可以较好地支持文件的删除和调整大小的操作，但是该方式将小文件元数据的 IO 操作次数翻倍，并且需要对所有小文件进行全盘扫描，再次读写以完成小文件的聚合任务. 第二种实现方式则避免了该问题，但是在文件删除和调整大小等操作较多的应用场景，会导致聚合对象产生大量空洞，从而导致文件系统出现较为严重的碎片化问题，第一种实现方式可以缓解该问题，但无法根除. 对于聚合对象空洞和碎片化的问题，目前大多数存储厂商基本采用二次合并的方式加以解决，即分配新的聚合对象，将位于空洞较多的聚合对象上的小文件再次读写到新的聚合对象，并更新小文件与聚合对象的对应关系.

根据以上论述，不论何种小文件聚合的实现方式，在存在频繁删除和文件大小调整的应用场景下，都将面临聚合对象空洞和数据碎片化的问题. 针对该问题，基于上述第二种思路中的实现方式，将其中聚合对象分配模块重新设计为聚合空间管理（aggregation space management，ASM）模块，完成聚合对象的空间管理，如图 5 所示，ASM 模块在存储集群中以独立进程为文件系统接口提供服务，包括聚合对象空间的申请和归还.

在 ASM 模块内部，维护一定数量的聚合对象与其空闲空间链表的映射关系，其中聚合对象数量越多，读写并发数越高，同时也将增加聚合对象空闲空间维护的空间和时间复杂度. 每个聚合对象的空闲空间链表记录空闲的数据范围，数据范围以块（block）为单位进行管理. 以聚合对象大小为 64 MB、block 大小为 4 KB 为例，聚合对象初始化的空闲空间链表中只有 1 个空闲范围节点，其中起始块（start block）为 0，块数（block count）为 16 384.

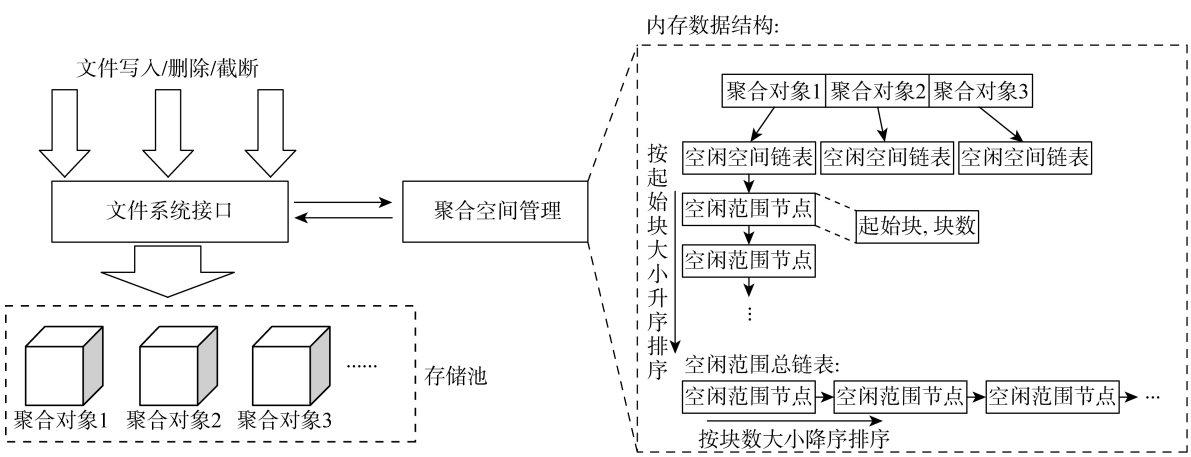

图 5　聚合空间管理模块设计

Fig. 5　Design of ASM

聚合空间申请请求中包含需要的块数,应答消息中包含给申请者的聚合对象标识及其起始块和块数,如果单个聚合对象中没有连续的空闲范围可以满足请求需要的块个数,则应答中将包含多个聚合对象的空闲范围.聚合空间归还请求中则包含一个或多个聚合对象标识及其起始块和块数,ASM 模块收到聚合空间归还请求后,将请求中聚合对象及其归还的空间范围加入 ASM 模块的内存数据结构.该内存数据结构作为 ASM 内部的核心子模块,相应地提供聚合空间申请和归还操作.

在 ASM 模块的核心内存数据结构中,存在两类排序链表.一类是每个聚合对象的空闲空间链表,其中空闲范围节点按起始块升序排列;另一类是空闲范围总链表,按空闲范围节点的块数降序排列.执行聚合空间申请操作时,结合首次合适和最优合适的思想,首先从空闲范围总链表头部查找到首个满足请求需要块数的空闲范围节点,从中取出相应块数给申请者,同时调整该空闲范围节点在其所属聚合对象的空闲空间链表和空闲范围总链表中的位置.如果总链表头部节点已经无法满足请求需要的块数,则从总链表尾部逆向查找多个空闲范围节点,凑出请求需要的块数返回给申请者;同样地,需要调整涉及空闲范围节点在两类链表中的位置,对于没有空闲范围的节点则从链表中移除.执行聚合空间归还操作时,首先将请求中归还的空闲范围节点按起始块升序插入到相应聚合对象的空闲空间链表,合并存在重叠的空闲范围节点,然后按块数降序调整涉及的空闲范围节点在总链表中的位置.聚合空间的补给操作属于 ASM 模块的内部操作,由独立的线程负责.在执行聚合空间申请操作时,发现剩余的总块数小于一定阈值,则通知该线程执行创建新的聚合对象,并向内存数据结构中添加聚合对象及其空闲空间信息.

5 数据存储服务设计

业务数据经过存储协议网关转发到达集群中的各个存储节点后,由存储节点的数据服务进程完成业务数据的落盘.数据服务进程作为存储协议网关的服务端,其设计时需要考虑以下两个方面:面向数据请求高并发处理能力;持续数据写入性能的稳定性.本节将从这两个方面介绍数据存储服务的设计.

5.1 单线程异步并发模型设计

作为存储节点的核心服务,数据服务进程的并发模型设计必须充分发挥存储节点多核 CPU 的并行处理能力,以及并发处理网络和硬盘的数据 IO 操作.因此,存储端数据服务的设计将采用进程池与事件驱动的方式完成数据 IO 的异步并发处理.单个数据服务进程内部的事件驱动主循环执行数据异步 IO 的并发模型如图 6 所示,进程池的模式用以发挥多核 CPU 的并行处理能力,其中的每个数据服务进程是对等关系,内部采用相同的并发模型,网络与硬盘的 IO 操作通过相应的操作系统接口完成异步提交和回调结果处理,最大限度地让数据 IO 填满存储节点网络和硬盘的带宽及 IO 资源.

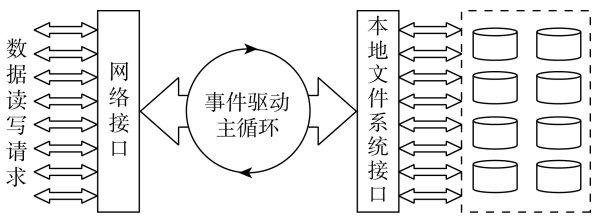

图 6 数据服务进程的并发模型

Fig. 6 Concurrency model of data service processes

事件驱动是进程内部实现异步并发的关键.操作系统中所有数据 IO 操作都可以通过文件的形式提供统一的处理接口.由图 6 可见,数据服务进程中驱动主循环执行逻辑的事件主要有两类:网络和文件系统的数据 IO 事件.两者执行数据 IO 操作的句柄对象分别是 socket 和 fd,前者通过 epoll 实现对每个 socket 上的数据 IO 事件的监测,后者通过 io_submit 和 io event 机制处理每个 fd 的异步 IO 响应.每个 IO 事件的发生都将触发主循环执行相应的回调处理逻辑,完成网络与文件系统之间的数据交换.

5.2 性能稳定性保障

多个任务由单个进程并发异步执行的过程中,每个任务执行时间的均衡可控对性能的持续稳定性也起到了关键作用.对于时间较长的网络和硬盘 IO 操作,采用异步 IO 的方式释放进程处理其他任务,而针对那些没有异步 IO 方式、耗时较长且低频的 IO 操作,启动一个独立的服务进程进行处理,防止这类 IO 操作对业务数据 IO 性能产生影响.

无锁机制是高并发请求处理中保证持续稳定高性能的有力支撑.为了实现进程池中多个数据存储服务进程之间对文件数据读写的无锁设计,每个数据服务进程侦听不同的 TCP 端口,存储协议网关转发数据读写请求时,对文件标记进行哈希计算映射到固定的 TCP 端口,实现多个数据存储服务之间的无锁设计.同时,每个数据存储服务进程侦听的端口数保持相同,在高并发请求处理的过程中,进一步确保每个服务进程的任务分配均衡性.

高并发处理场景下，存储协议网关与数据存储服务进程之间会建立数以千计的连接数，这些连接在数据存储服务进程内部被公平处理，数据请求通过不同的连接到达数据存储服务进程后，每个服务进程以轮询的方式接收并执行每个连接上的请求. 而数据服务进程之间用于数据请求转发的连接被设为高优先级，在主循环中被优先处理，保证每个来自存储协议网关的数据请求能够被尽快应答.

6　实验验证

在进行以上优化后，通过实际测试来验证上述设计在实际生产系统中性能方面的表现特征，包括对接实际 PACS 业务系统的测试，以及与传统 SAN (storage area network)存储性能的对比测试.

首先将优化后的存储系统与上海某三甲医院实际 PACS 进行对接测试，以验证医生调阅终端的 PACS 软件调阅速度提升效果. 测试客户端为 9 台医生调阅终端，运行优化后的 PACS 调阅软件，开启 8 个线程执行数据调阅任务. 表 1 和表 2 列出了此次测试中影像(CT 和 MR 检查)数据规模大小和调阅时间的统计数据，其中调阅时间为 9 台终端测试数据的平均值. 测试结果显示，经过对 PACS 医疗影像数据调阅的全路径中各个关键环节进行优化升级后，医生端 PACS 软件的调阅速度提升至每秒约 300 幅图像，达到了原来调阅速度的 3 倍以上.

表 1　CT 数据调阅时间

Tab. 1　Time consumed for CT data retrieval

序号	CT 图像信息			平均调阅时间/s	
	数量/张	大小/MB	分辨率/(像素×像素)	优化前	优化后
1	311	63.9	512×512	3.83	1.11
2	629	141	512×512	7.96	2.29
3	909	191	512×512	11.35	3.28
4	1 539	328	512×512	18.76	5.29
5	2 132	491	512×512	25.68	7.55
6	4 597	939	512×512	53.45	15.47

表 2　MR 数据调阅时间

Tab. 2　Time consumed for MR data retrieval

序号	MR 图像信息		平均调阅时间/s	
	数量/张	大小/MB	优化前	优化后
1	1 010	161	12.46	3.38
2	2 106	222	25.37	5.57
3	3 064	347	35.62	8.19

为进一步验证本文设计实现的 PACS 分布式存储系统在性能方面提升的普适性，与传统 SAN 存储进行对比测试. 图 7 所示为优化设计的分布式存储系统与传统 SAN 存储之间的性能对比，可见：传统 SAN 存储在业务客户端线程数为 8 时性能最优，之后随着线程数增加性能呈下降趋势；优化后的分布式存储系统性能随着客户端线程数的增加呈线性增长趋势，线程数越多性能优势越明显，在 64 线程时，下载速率达到传统 SAN 存储的 3 倍以上，能够更好地满足医疗 PACS 业务系统对海量小文件性能、高并发访问以及扩展性等方面的存储需求.

最后，为了验证 ASM 对存储空间碎片化问题的改善效果，进行对比实验. 准备两组硬件配置相同的存储集群，软件配置为使用 ASM 组和未使用 ASM 的对照组，业务端对接医院的 PACS 服务，在 PACS 服务中增加数据的删除和复写操作，持续运行 1 周，每天同一时刻统计存储集群中各个节点上硬盘的碎片化程度. 测试记录的结果如图 8 所示，可见：未使用 ASM 的情况下，存储空间的碎片化程度随着业务端对数据的修改、压缩和删除操作的增加

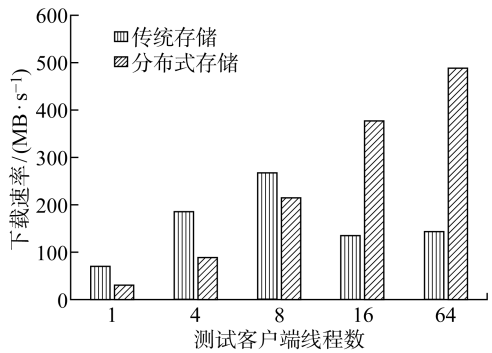

图 7　分布式存储与传统 SAN 存储性能对比

Fig. 7　Comparison of performance between distributed storage and traditional SAN storage

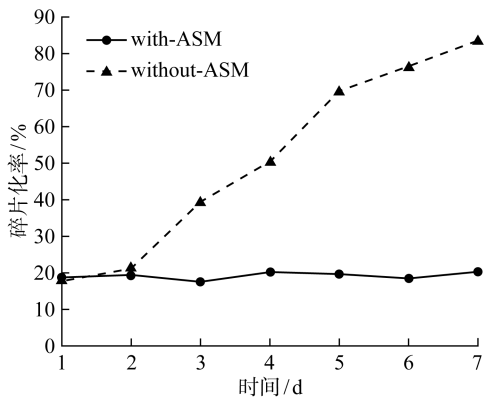

图 8　使用 ASM 前后存储空间碎片化率对比

Fig. 8　Comparison of fragmentation rate with ASM and without ASM

而不断增加;使用ASM后,存储空间碎片化程度基本维持在19%上下,不会随着业务端长时间对海量小文件的复写和删除而持续增加,避免了周期性的碎片整理工作,减少运维成本的同时保证了海量小文件读写的持续稳定.

7 结语

本文针对医疗PACS影像系统调阅速度慢的问题,从医生端PACS软件、存储协议网关、小文件聚合以及数据存储端服务4个环节入手,对PACS影像数据调阅的全IO路径进行优化升级,解决了传统架构下医疗PACS调阅卡顿的问题.测试结果显示优化升级可使PACS软件调阅速度达到原速度的3倍以上,约为每秒300幅图像;随着并发线程数的增加,本文优化设计的PACS分布式存储系统性能呈线性增长趋势,能够更好地应对医院PACS应用场景在高并发访问性能、可扩展性以及海量小文件等方面的存储需求.

参考文献:

[1] ALDOSARI H, SADDIK B, KADI A K. Impact of picture archiving and communication system (PACS) on radiology staff[J]. **Informatics in Medicine Unlocked**, 2018, 10: 1-16.

[2] HAAK D, PAGE C E, REINARTZ S, *et al*. DICOM for clinical research: PACS-integrated electronic data capture in multi-center trials[J]. **Journal of Digital Imaging**, 2015, 28(5): 558-566.

[3] SHVACHKO K, KUANG H R, RADIA S, *et al*. The hadoop distributed file system[C]// **IEEE 26th Symposium on Mass Storage Systems and Technologies**. Incline Village, USA: IEEE, 2010: 1-10.

[4] 曾梦, 邹北骥, 张文生, 等. 多模态医疗数据中海量小文件存储优化方法[J]. **软件学报**, 2023, 34(3): 1451-1469.
ZENG Meng, ZOU Beiji, ZHANG Wensheng, *et al*. Optimization method for storing massive small files in multi-modal medical data[J]. **Journal of Software**, 2023, 34(3): 1451-1469.

[5] VORA M N. Hadoop-HBase for large-scale data[C]// **Proceedings of 2011 International Conference on Computer Science and Network Technology**. Harbin, China: IEEE, 2011: 601-605.

[6] 郑通, 郭卫斌, 范贵生. HDFS中海量小文件合并与预取优化方法的研究[J]. **计算机科学**, 2017, 44(11A): 516-519.
ZHENG Tong, GUO Weibin, FAN Guisheng. Research on optimization method of merging and prefetching for massive small files in HDFS[J]. **Computer Science**, 2017, 44(11A): 516-519.

[7] BENDE S, SHEDGE R. Dealing with small files problem in Hadoop distributed file system[J]. **Procedia Computer Science**, 2016, 79: 1001-1012.

[8] PATEL A, MEHTA M A. A novel approach for efficient handling of small files in HDFS[C]// **IEEE International Advance Computing Conference**. Banglore, India: IEEE, 2015: 1258-1262.

[9] AGGARWAL R, VERMA J, SIWACH M. Small files' problem in Hadoop: A systematic literature review[J]. **Journal of King Saud University-Computer and Information Sciences**, 2022, 34(10): 8658-8674.

[10] TAO W J, ZHAI Y L, TCHAYE-KONDI J. LHF: A new archive based approach to accelerate massive small files access performance in HDFS[C]// **IEEE Fifth International Conference on Big Data Computing Service and Applications**. Newark, USA: IEEE, 2019: 40-48.

[11] GAO Z P, QIN Y H, NIU K. An effective merge strategy based hierarchy for improving small file problem on HDFS[C]// **4th International Conference on Cloud Computing and Intelligence Systems**. Beijing, China: IEEE, 2016: 327-331.

[12] 陈法河, 柴小丽. 基于Ceph存储系统的小文件存储优化方案[J]. **计算机系统应用**, 2022, 31(2): 108-113.
CHEN Fahe, CHAI Xiaoli. Optimization of small file storage algorithm based on Ceph storage system[J]. **Computer Systems & Applications**, 2022, 31(2): 108-113.

[13] SHI A W, TIAN Z C, CHEN G, *et al*. Research and optimization of massive small file processing performance based on Ceph[C]// **International Conference on Electronic Information Engineering and Data Processing (EIEDP 2023)**. Nanchang, China: SPIE, 2023: 311-315.

[14] BEAVER D, KUMAR S, LI H C, *et al*. Finding a needle in Haystack: Facebook's photo storage[C]// **Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation**. Vancouver, BC, Canada: USENIX Association, 2010: 47-60.

[15] FU S L, HE L G, HUANG C L, *et al*. Performance optimization for managing massive numbers of small files in distributed file systems[J]. **IEEE Transactions on Parallel and Distributed Systems**, 2015, 26(12): 3433-3448.

(本文编辑:孙伟)