计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2014.20121094
51(8):1663-1670, 2014

MDDS:一种面向高性能计算的并行文件系统元数据性能提升方法

陈 起 陈左宁 蒋金虎
(江南计算技术研究所 江苏无锡 214083)
(chenqi.jn@gmail.com)

MDDS: A Method to Improve the Metadata Performance of Parallel File System for HPC

Chen Qi, Chen Zuoning, and Jiang Jinhu
(*Jiangnan Institute of Computing Technology*, *Wuxi*, *Jiangsu* 214083)

Abstract With the increasing of the computational ability of supercomputers, problem size and complexity targeted by applications, higher performance of I/O subsystems is required. While the throughput of single metadata server limited the performance of the parallel file system in high concurrent access and high-frequency file creating/deleting scenarios. Focused on the typical parallel I/O scenario in high performance computing, MDDS (meta data delegation service) is implemented in Lustre file system, which uses loose coupling to keep the high availability of the cluster, organizes the MDDS namespace by directory subtree to avoid the complexity and inefficiency of distributed atomic operations introduced by cross-node operations, and uses metadata migration mechanism to avoid objects data moving between data servers. Oriented to I/O-forwarding framework, two job-scheduling strategies, one job scheduled on single MDDS and jobs sharing multiple MDDS, are addressed to achieve load balancing of the requests for metadata inside or between jobs. The performance of MDDS is evaluated on 116 storage servers. The initial experimental results show that quasi linear scalable metadata performance is achieved by MDDS, and even show better scalability than Lustre CMD (cluster metadata Design) in larger-scale cluster. The two job-scheduling strategies distribute the applications' metadata access load effectively, and overcome performance bottlenecks in accessing file metadata in HPC.

Key words high performance computing; parallel file system; metadata delegation service; I/O-forwarding framework; load balance

摘 要 随着计算能力的增强、应用课题规模和复杂度的增加,高性能计算机对并行文件系统性能要求越来越高.在海量小文件和大规模并发I/O操作的应用场景中,文件系统元数据的吞吐率成为限制其性能的关键因素.设计并实现了元数据代理(meta data delegation service, MDDS),通过降低元数据服务间的耦合度,保证元数据集群的高可用性;使用目录子树方式管理元数据代理空间,避免跨节点目录引入的分布式原子操作的复杂性和低效性.并针对高性能计算中I/O转发架构,提出基于元数据代理的两种作业调度策略——单作业独占单元数据代理调度和多作业共享多元数据代理调度——实现作业间和作业内的负载均衡.在116台存储服务器上对MDDS进行评估,实验结果表明,元数据代理提供了拟线性

收稿日期:2012-11-29;**修回日期**:2013-06-07
基金项目:国家"八六三"高技术研究发展计划基金项目(2011BAH04B03,2013AA013203)

的元数据性能,在大规模的环境中较 Lustre CMD 方案有较好的扩展性;两种调度方式有效分散了作业元数据的负载,改善了高性能计算中的元数据瓶颈问题.

关键词 高性能计算;并行文件系统;元数据代理;I/O 转发;负载均衡

中图法分类号 TP333

高性能计算系统被广泛应用在科学研究、工业生产、国防军事等各种领域,在能源、航天、气象、生命科学、金融、动漫等各个行业发挥着重大作用,其按功能可分为存储子系统和计算子系统两部分. 存储子系统为整机提供全局一致的共享存储空间,计算子系统以全局存储空间为纽带相互协作完成课题计算. 计算规模的扩充对存储子系统在支持客户端规模、存储性能上提出了更高的要求.

高性能计算机常采用集中式元数据管理,很难应对海量、高并发的元数据访问. 文献[1]指出由于单个元数据服务器和分布式锁的限制,在面临大规模并发访问时文件系统元数据吞吐率会下降,并建议采用多目录分层的方法减少锁冲突,使用 I/O 聚合和 I/O 中间层降低存储系统元数据访问并发度. 文献[2-3]等利用 MPI 的聚合 I/O 方式减少存储系统访问的并发进程数,缓解元数据压力,但需要专用文件格式和访问接口[4-6],加大了编程难度,且很多成熟高性能大型计算软件由于基于传统 POSIX 语义,无法利用这些技术. 京速系统[7]采用分级存储技术为不同的计算任务构建独立的全局文件系统,使不同作业运行在不同的全局文件系统上,避免了作业间存储访问冲突. 但由于存储空间隔离,多个作业不能共享数据存储空间,会造成数据服务器负载不均衡,且不同存储系统间数据迁移会产生大量的数据移动. I/O 转发架构[8]是一种缓解海量客户端对存储系统压力的一种可扩展的架构. 它将计算节点的 I/O 请求聚合到规模相对较小的 I/O 代理上,由 I/O 代理完成对全局文件系统的访问,降低元数据服务器处理的并发度,避免了由大规模并发访问引起的元数据服务性能下降的问题,但由于所有的元数据请求仍需由单个元数据服务器处理,元数据性能瓶颈依然存在.

针对 I/O 转发架构,采用分层的元数据管理方法,基于 Lustre CMD[9]设计并实现了 Lustre 元数据代理 MDDS,并提出了两种调度策略解决作业内和作业间的元数据负载均衡问题.

1 元数据代理 MDDS 的设计

1.1 系统整体架构

元数据代理架构是针对高性能计算的 I/O 转发架构设计的,其架构如图 1(a)所示,在 Lustre 并行文件系统和计算节点间加入了 I/O 代理节点,Lustre 的客户端仅部署在 I/O 代理节点上,计算节点通过轻量级文件系统 LWFS 访问元数据代理节点的 Lustre 客户端完成对全局文件系统的访问. 该架构可通过增加 I/O 代理节点的数量提高支持的计算节点规模和文件系统读写性能,具有很好的可扩展性.

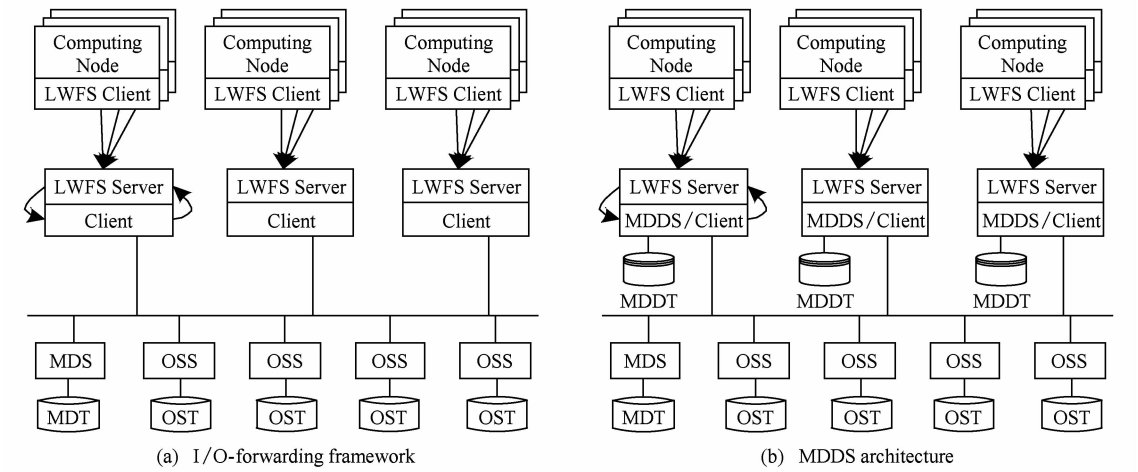


Fig. 1 I/O-forwarding framework and lustre mdds architecture.

图 1 I/O 转发架构和 Lustre 元数据代理架构

由于元数据请求仍只能被单一元数据服务器处理,I/O 转发架构的元数据性能瓶颈依然存在. 在 Lustre 中设计分层的元数据集群架构,实现了元数据代理机制 MDDS,将元数据集群分为元数据服务器 MDS 和元数据代理集群 MDDS,并将 MDDS 部署在 I/O 代理节点上,如图 1(b)所示. 元数据服务器 MDS 用于维护整个文件系统的全局一致性视图和元数据代理 MDDS 的信息,MDS 和 MDDS 共享所有的数据对象服务器空间. MDDS 具有临时的本地存储,计算节点的元数据请求可由元数据代理直接处理和暂存;MDDS 暂存的元数据可迁移至元数据服务器进行可靠性存储. 元数据集群的这种分层管理方式,使其可针对 MDS 和 MDDS 进行专门优化. MDS 管理全局的信息,可采用大容量、高可靠性的存储介质进行元数据存储,同时使用 failover 机制保证元数据服务的高可用性;元数据代理上的数据仅作暂存使用,其可采用高性能、低容量的 SSD 或内存作为后端存储以提高存储访问性能.

1.2　元数据代理的组织

Lustre CMD 使用目录 Hash 进行元数据在元数据集群内的布局,它能有效进行元数据负载均衡. 但目录 Hash 机制会致使文件系统中所有目录的 Entry 和 Inode 信息分布在不同的元数据服务器上,需采用分布式原子操作保证文件系统的目录创建、删除和修改等操作的原子性,而分布式原子操作语义复杂,效率低下,在大规模并发元数据操作的环境中增加了服务器间的交互,限制元数据集群的性能. 此外,目录 Hash 方式会使元数据集群中的元数据服务器处于同等的地位,单个元数据服务器故障会致使所有的元数据服务不可用. 随着元数据集群规模的扩展,使用目录 Hash 构建元数据集群的元数据服务的可用性下降. 在高性能计算系统中,全局文件系统是整个计算系统的纽带,文件系统的可用性限制了整机的可用性. 使用目录 Hash 方式很难应对高性能计算对存储系统可扩展性和可用性的要求.

子树分割[10-11]是元数据集群构建的一种常见方式,它以目录子树为单位将文件系统命名空间分配到不同的元数据服务器上,最大限度地维护了文件系统访问的局部性,同时单个元数据服务器故障仅会影响其所代理的目录子树,而不会影响其他目录子树的元数据服务. MDDS 采用该方式进行元数据代理命名空间的组织,以避免由元数据代理规模增大造成的元数据服务可用性的降低问题,简化元数据代理交互逻辑.

系统的实现基于 Lustre CMD 软件栈,简化了 CMD 中元数据服务器间的连接,仅使 MDDS 与 MDS 保持连接,而 MDDS 之间相对独立,如图 2 所示. 这样单个 MDDS 出现故障不会影响其他的元数据服务.

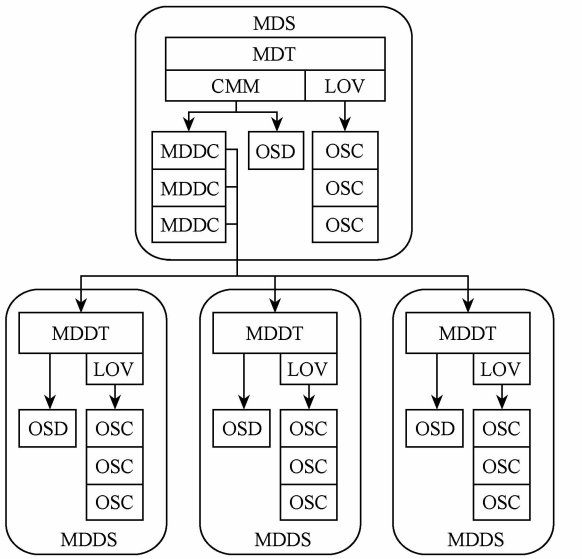


Fig. 2　The logical inter-connection between MDS and MDDS.

图 2　MDS 和 MDDS 逻辑连接关系图

在客户端添加 MDDC Pool 的逻辑用于管理和元数据代理的连接,如图 3 所示. MDDC Pool 管理所有与 MDDS 的连接关系,同时也可用于元数据代理添加、删除等功能的实现.

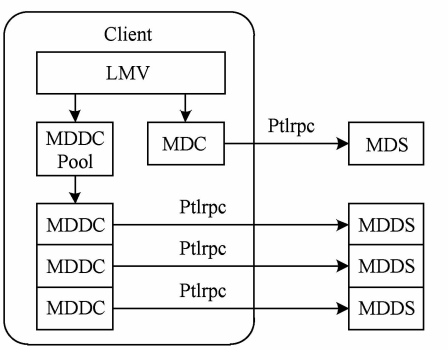


Fig. 3　The logical inter-connection between Lustre client and MDDS/MDS.

图 3　Lustre 客户端和 MDS/MDDS 交互关系

子树分割方式虽然简化了元数据代理命名空间组织的逻辑,但较难做到元数据负载均衡,为此根据高性能计算中典型的 I/O 模式[12-13]设计了两种作业调度方式来解决元数据负载均衡的问题,详见第 3 节.

1.3 元数据迁移机制

目录元数据代理主要用于高性能计算中并行课题的加速. 随着计算环境和课题的变化,原元数据的位置可能已不是最优的,且目录元数据代理的空间有限、数据可靠性比文件系统元数据服务器差,需要实现元数据在 MDS 和 MDDS 命名空间的移动.

Lustre 采用的类 ext3/ext4 的磁盘格式进行元数据的存储,很难在块级别进行存储数据语义的识别. 为此,借鉴 Linux 中的 mv 操作实现文件级的元数据迁移.

文件系统中的 mv 操作是通过 rename 系统调用实现的,该操作仅进行了文件 Entry 信息的转化,不会进行数据的移动,如图 4(a);跨文件系统的 mv 是通过在目标文件系统上创建对应的文件,并将数据复制到新文件中实现的,它涉及到文件的创建和文件数据的移动,如图 4(b)所示. Lustre CMD 的跨 MDS 的 mv 操作中,采用的 4(a)方案,不能改变 Inode 的位置,不能达到元数据迁移的目的,为此利用 MDDS 和 MDDS 共享数据空间的特点,采用图 4(c)的方案进行元数据在 MDDS 和 MDS 间的迁移.

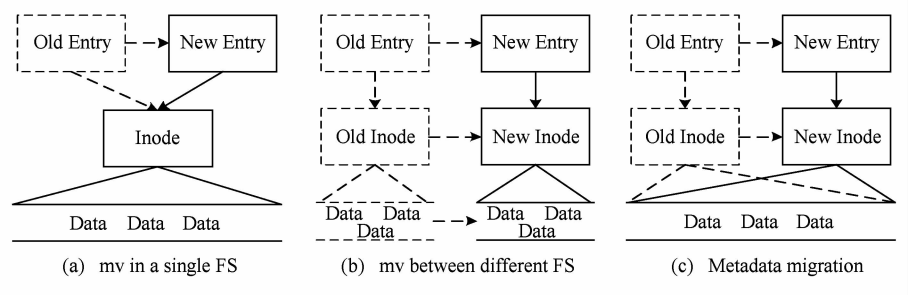


Fig. 4 Three different methods of moving file metadata.

图 4 3 种移动元数据的方法

Lustre CMD 采用图 5(a)中的元数据存储格式,使用 FID 进行元数据在全局一致的标示,同时通过它进行文件或目录的 Inode 定位,使用 OOI (OSD object index)实现 FID 到本地元数据存储的定位. 针对该存储格式,使用如图 5(b)中的元数据迁移过程,在 MDDS 上使用 MDS 上文件的元数据信息重建 Inode 和 Entry,并将其更新到 MDDS 上的 OOI 中,同时在 MDS 上清理到已迁移的元数据信息.

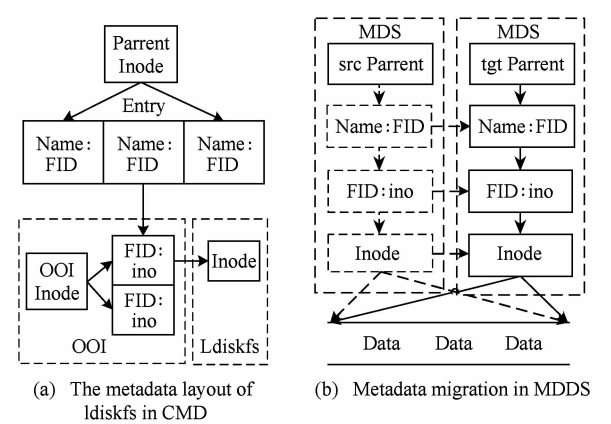


Fig. 5 The principle of metadata migration between MDS and MDDS.

图 5 文件元数据迁移原理

元数据迁移流程如图 6 所示,为了保留 FID 的位置信息,如果目标文件存在,则重用其 FID,否则会在客户端申请新的 FID.

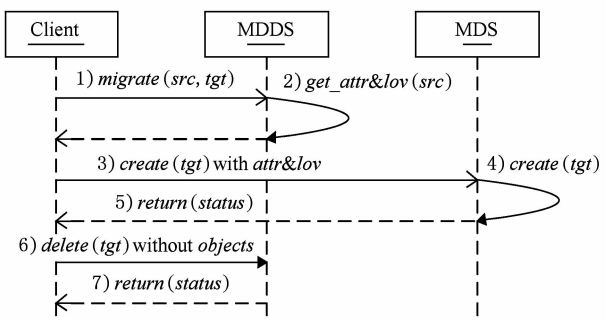


Fig. 6 The process of file-level metadata migration.

图 6 文件级元数据迁移流程

在 Lustre 中,LOV 数据维护着文件或目录在对象服务器上的分布,通过 LOV 信息保证迁移后文件内容不改变. 在迁移的环境下,为了区分不同元数据服务节点上的文件分条数据信息,采用 4 元组 ($f\_MDDS, f\_seq, f\_oid, f\_ver$)来标示客户端缓存中的对象,其中 f_MDDS 为对象所属的元数据服务器编号,(f_seq, f_oid, f_ver)用来进行对象在数据服务器上寻址.

2 基于 MDDS 的作业管理

元数据代理只用来并行运行课题的加速,一些交互任务(如文件的编辑、修改、编译)和课题初始化

数据存储 MDS 上.在课题运行时,其对应元数据需从 MDS 迁移到 MDDS 中,使用 MDDS 提供课题的元数据服务;当课题运行完成后,元数据从 MDDS 迁回到 MDS 中,实现可靠存储.元数据代理 MDDS 使用子树方式管理元数据代理空间,无法实现作业的负载均衡[14].高性能计算中的作业具有典型的并行 I/O 模式,利用该特点,提出了两种作业调度方法以满足典型课题作业内和作业间的文件系统元数据负载均衡.

2.1　单作业独占单元数据代理调度

成熟的高性能计算软件多使用传统 I/O 访问接口进行存储系统的访问,这些软件由于成熟度高、甚至软件代码不开源,很难修改其 I/O 访问方式.单作业独占单元数据代理调度方式就是针对该应用场景,实现对传统作业的透明地兼容,原理如图 7 所示:

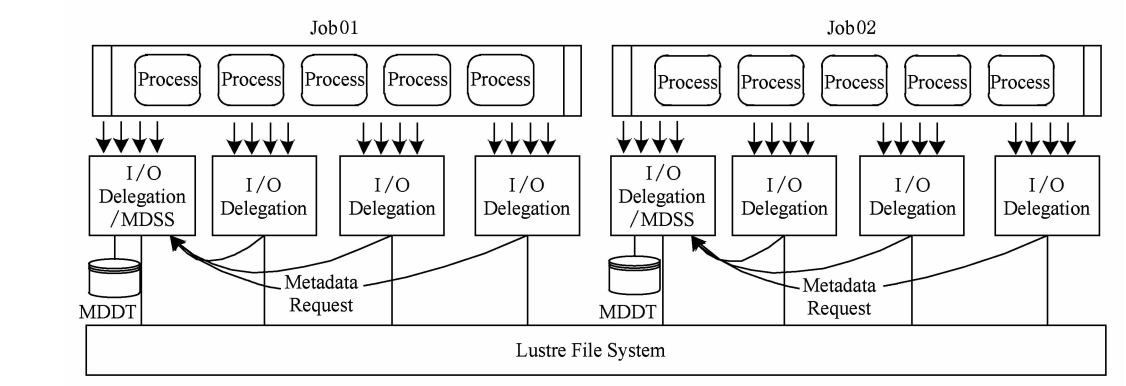


Fig. 7　Per job scheduled on single MDDS.

图 7　单作业独占单元数据代理调度

其将作业调度分为 3 个阶段:第 1 个阶段是由作业调度程序根据作业使用的计算节点组选出支持计算节点最多的 I/O 代理节点,并将该作业相关的元数据信息迁移到该代理节点上,使用该节点提供文件系统元数据服务;第 2 个阶段是作业与选中的元数据代理交互完成作业的执行;第 3 个阶段是作业元数据从元数据代理迁移到元数据服务器上实现可靠、永久保存,并释放出元数据代理空间供其他课题使用.

当作业的计算节点规模由单个 I/O 代理就能应对时,作业规模较小,文件系统元数据性能不易成为作业的瓶颈.当作业需由多个 I/O 代理处理时,元数据代理选择方法可以保证在多作业并发的环境中,不同作业元数据服务分散到不同的元数据代理上,避免了作业间元数据访问干扰,提高了整机的元数据服务性能.

2.2　多作业共享多元数据代理调度

单作业独占单元数据代理调度虽能提供对传统作业透明的支持,实现作业间的负载均衡,但由于单个作业的元数据请求局限在单个元数据代理上,无法实现作业内的元数据负载均衡.针对该不足,将 Lustre CMD 中的目录 Hash 的思想应用到作业调度中,提出了多作业共享多元数据代理的调度方式,实现作业内文件系统元数据负载均衡,其原理如图 8 所示:

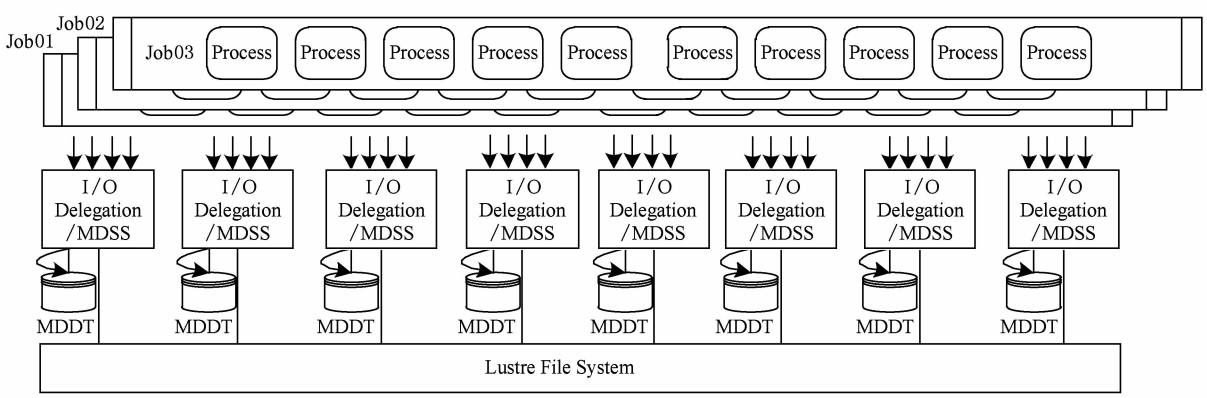


Fig. 8　Jobs sharing multiple MDDSes.

图 8　多作业共享多元数据代理调度

在该调度方式中,计算节点和 I/O 代理节点间使用 Hash 方式连接,使单个作业内的 I/O 请求尽可能地分散到多个 I/O 代理上去处理. 作业需指定元数据代理目录来引导作业的元数据请求分散到不同元数据代理节点上. 作业调度完成后,在元数据代理上相关的元数据信息通过元数据迁移方式收集到元数据服务器上,供后续课题的处理. 该调度方式可将单个作业的元数据访问分散到多个元数据代理上,获得并行的元数据性能,对高频文件创建删除的课题会有较大提高,实现了课题内元数据访问负载均衡.

I/O 转发架构中往往有成百上千的 I/O 代理节点. 若课题的元数据分散到所有的 I/O 代理节点上,单个元数据代理故障会影响所有课题的执行,系统可用性会降低. 在实际工程中一般通过先对 I/O 代理节点进行分组管理,再在组内采用 Hash 互连来保证系统的可用性.

3 实验及分析

元数据代理 MDDS 开发基于 Lustre-2.1.3,实验在 116 台 HP Gen8 服务器上进行,操作系统使用 RHEL5.8,元数据和对象存储使用 loop 设备构建,网络使用千兆以太网. 在测试通过不分配数据对象屏蔽 Lustre 对象申请对文件创建元数据性能的影响,测试工具使用 mdtest[15].

首先测试了 MDDS 和 Lustre CMD 在元数据集群上的可扩展性,结果如图 9 所示:

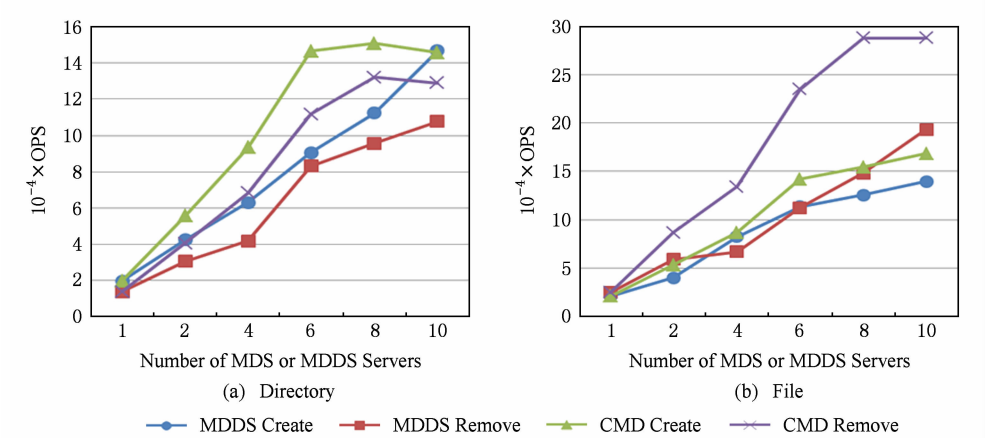


Fig. 9 The comparison of metadata performance between MDDS and CMD architecture.

图 9 MDDS 和 CMD 元数据性能对比

由图 9 可知,Lustre CMD 在元数据负载均衡上更具优势,获得较好的元数据性能. 但随着元数据集群规模的扩大,元数据代理 MDDS 由于使用子树方式避免了元数据服务器间的交互,获得更好的可扩展性,即使在元数据规模较大时,也能表现出拟线性的元数据性能可扩展性.

元数据迁移是元数据代理和元数据服务器进行元数据移动的主要方式,在此对其性能进行了测试,同时与 cp 操作和文件创建操作进行了对比,结果如图 10 所示.

由图 10 可知,对空文件拷贝时,由于需要创建目标文件和打开源文件,其所花费的时间约为文件创建的 2 倍;当文件中有数据存在时,文件的拷贝操作性能就受限于数据的大小. 而元数据迁移由于不需要移动数据对象,且不需要进行目标数据对象的创建,其性能不随文件大小的改变而变化,可高效地实现 MDS 和 MDDS 间元数据移动.

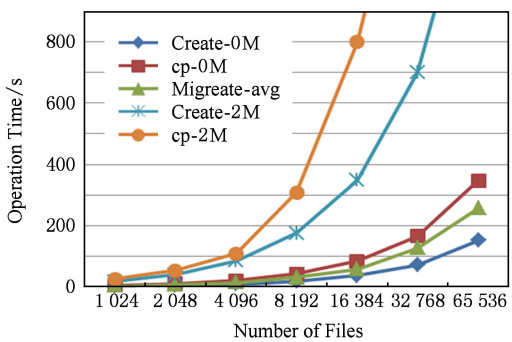


Fig. 10 The performance of metadata migration and file copy.

图 10 元数据迁移性能对比图

通过模拟 LWFS 服务端的行为产生高并发的元数据负载,通过单个作业获得元数据吞吐率来衡量整机元数据性能的优劣. 在实验中,并行模拟了 10 个作业,每个作业独占 8 个 I/O 代理节点,并将该结果与单元数据服务器模式进行了对比,实验原理

如图 7 所示,结果如图 11 所示:

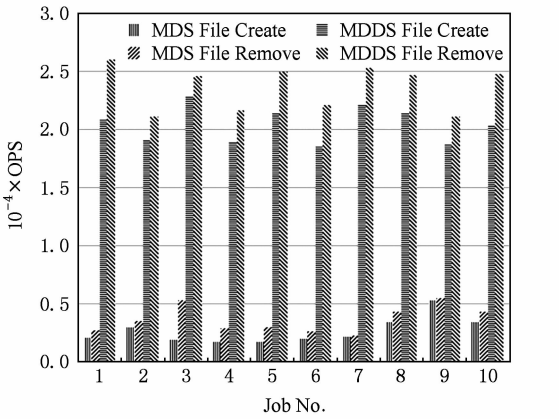


Fig. 11　The metadata performance comparison of MDDS and MDS in per job scheduled on single MDDS mode.

图 11　单作业独占单元数据代理作业文件元数据性能

由图 11 可知,在作业并发的模式下,单作业独占单元数据代理中每个作业其可获得单元数据代理的最大性能;而在单元数据服务器模式中,所有的作业共享同一个元数据服务器,多个元数据服务器只能获得部分的元数据服务器性能. 单作业独占单元数据代理模式较传统单元数据服务器在元数据性能有较大提升,且有效地避免了作业间元数据的干扰.

模拟使用作业共享多元数据代理的调度方式,通过整机获得元数据性能评估元数据代理的性能,实验原理如图 8 所示,共使用了 30 个元数据代理节点,模拟了在不同作业数量下的元数据负载. 结果如图 12 所示.

同时在该模式下,对 MDDS 和 CMD 的性能进行了对比,结果如图 13 所示.

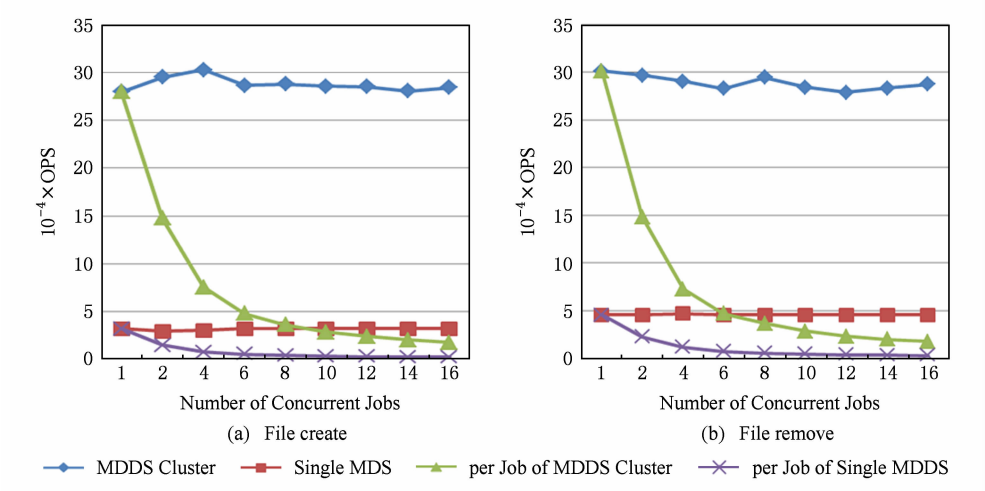


Fig. 12　The metadata performance comparison of MDDS and MDS in jobs sharing multiple MDDSes mode.

图 12　多作业独占多元数据代理作业元数据性能对比

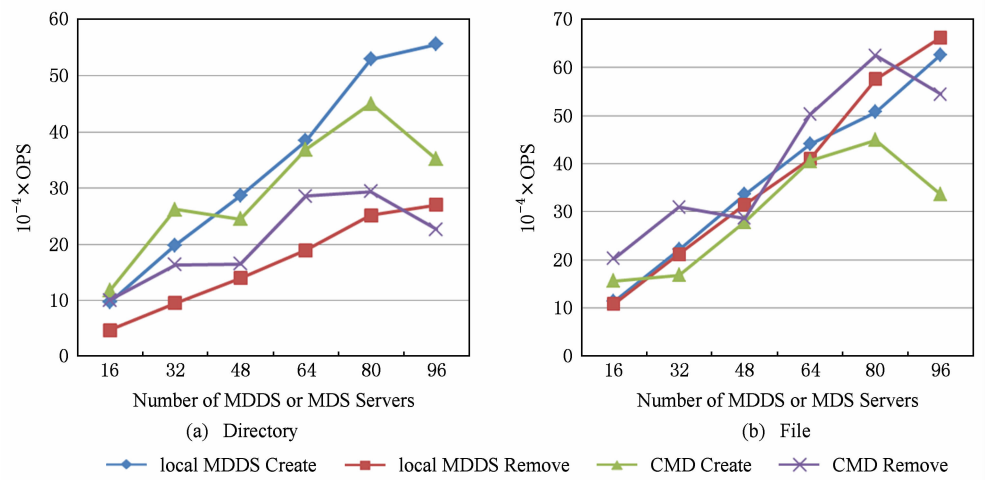


Fig. 13　The metadata performance comparison of MDDS and CMD in jobs sharing multiple MDDSes mode.

图 13　多作业共享多元数据代理中 MDDS 和 CMD 性能对比

由图 13 可知,在多作业共享多元数据代理调度中,作业所获得的元数据性能可达与 Lustre CMD 相似的性能.随着 I/O 转发集群规模的扩大,由于跨节点目录访问的开销,Lustre CMD 的元数据性能增长变缓、甚至出现下降的情况;而 Lustre MDDS 不因规模扩展而带来大量的网络开销,仍然表现了出色的可扩展性.

4 结束语

针对高性能计算中 I/O 转发架构,采用分层的元数据管理方法,基于 Lustre CMD 设计并实现了元数据代理 MDDS,简化了元数据集群交互逻辑,保持了元数据集群规模扩展时元数据服务的可用性.同时提出了两种调度方法实现了高性能计算作业内和作业间的元数据负载均衡,有效缓解高性能计算中的文件系统元数据访问瓶颈.

参 考 文 献

[1] Shipman G M, Dillow D A, Oral S, et al. Lessons learned in deploying the world's largest scale lustre file system[C/OL] //Proc of the 52nd Cray User Group Conf (CUG2010), 2010: 1-10. [2012-05-20]. http://www4.ncsu.edu/~zzhang3/pubs/lessons-spider-cug10.pdf

[2] Fu Jing, Min M, Latham R, et al. Parallel I/O performance for application-level checkpointing on the blue gene/p system [C] //Proc of 2011 IEEE Int Conf on Cluster Computing (CLUSTER'11). Los Alamitos, CA: IEEE Computer Society, 2011: 465-473

[3] Fu Jing, Liu Ning, Sahni O, et al. Scalable parallel I/O alternatives for massively parallel partitioned solver systems [C] //Proc of 2010 IEEE Int Symp on Parallel & Distributed Processing (IPDPSW 2010). Piscataway, NJ: IEEE, 2010: 1-8

[4] Folk M, Heber G, Koziol Q, et al. An overview of the HDF5 technology suite and its applications [C] //Proc of EDBT/ICDT 2011 Workshop on Array Databases (AD'11). New York: ACM, 2011: 36-47

[5] Min M, Fischer P F, Chae Y C. Spectral element discontinuous Galerkin simulations for wake potential calculations: NEKCEM [C] //Proc of 2007 IEEE Particle Accelerator Conf (PAC'07). Piscataway, NJ: IEEE, 2007: 3435-3437

[6] Shen Weichao, Cao Liqiang, Xia Fang, et al. HDF5 performance optimization for numerical simulation data [J]. Journal of Computer Research and Development, 2012, 49 (Supp 1): 314-318 (in Chinese)
(沈卫超, 曹立强, 夏芳, 等. 面向数值模拟数据的 HDF5 性能优化[J]. 计算机研究与发展, 2012, 49(增刊 1): 314-318)

[7] Sakai Kenichiro, Sumimoto Shinji, Kurokawa Motoyoshi. High-performance and highly reliable file system for the *K* computer [J]. Fujitsu Scientific and Technical Journal, 2012, 48(3): 302-309

[8] Ali N, Carns P, Iskra K, et al. Scalable I/O forwarding framework for high-performance computing systems [C] //Proc of 2009 IEEE Int Conf on Cluster Computing and Workshops (CLUSTER'09). Piscataway, NJ: IEEE, 2009: 1-10

[9] Wang Di. Clustered metadata design [OL]. 2009: 1-22. [2012-05-20]. http://wiki.lustre.org/images/d/db/HPCS_CMD_06_15_09.pdf

[10] Menon J, Pease D A, Rees R, et al. IBM storage tank—A heterogeneous scalable SAN file system [J]. IBM Systems Journal, 2003, 42(2): 250-267

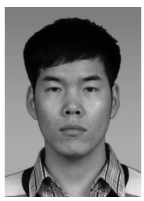
[11] Welch B, Unangst M, Abbasi Z, et al. Scalable performance of the panasas parallel file system [C] //Proc of the 6th USENIX Conf on File and Storage Technologies (FAST'08). Berkeley: USENIX Association, 2008: 17-33

[12] Newman H. HPCS mission partner file I/O scenarios [OL]. [2012-05-20]. http://ftp.jaist.ac.jp/pub/sourceforge/h/hp/hpcs-io/DARPA.HPCS.IO.Scenarios.2011.pdf

[13] Layton J. I/O pattern characterization of HPC applications [C] //Proc of the 23rd Int Symp on High Performance Computing Systems and Applications (HPCS 2009). Berlin: Springer, 2010: 292-303

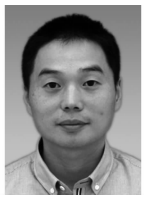
[14] Weil S A, Pollack T P, Brandt S A, et al. Dynamic metadata management for petabyte-scale file systems [C] //Proc of the 2004 ACM/IEEE Conf on Supercomputing (SC'04). Piscataway, NJ: IEEE, 2004: 523-534

[15] mdtest. [2012-05-20] http://sourceforge.net/projects/mdtest



Chen Qi, born in 1986. Received his master degree in computer architecture from Huazhong University of Science and Technology, Wuhan, China, in 2013. His main research interests include distributed storage system.

Chen Zuoning, born in 1957. Member of Chinese Academy of Engineering. Her main research interests include parallel computer system architecture and distributed systems.



Jiang Jinhu, born in 1974. Senior engineer. His main research interests include high performance computing storage architecture and storage software.