Lustre文件系统元数据服务恢复机制的改进

钱迎进 李永刚 汪 毅 周琳琦
(中国卫星海上测控部技术部 江阴 214431)

摘 要 Lustre的重启恢复算法需要集群中所有客户端在指定的恢复时间窗口内与服务器重新建立连接,客户端重传未提交的事务请求,服务器严格按照事务序列号重放所有未提交的事务,要求过于严格。针对Lustre可恢复性不强的缺点,提出了基于版本的恢复和共享时提交算法,它们分别对Lustre现有的元数据更新和恢复机制进行了改进和扩展,根据事务之间的依赖关系,允许客户端在更为宽松的条件下进行恢复并加入到集群而不被驱逐,提高了Lustre文件系统的可用性和可恢复性。最后通过一系列实验对改进后的算法的性能进行了评估。

关键词 Lustre,高性能计算,可恢复性,可用性
中图法分类号 TP391.9 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.9.034

Improvement of Recovery Mechanism for Lustre Metadata Service

QIAN Ying-jin LI Yong-gang WANG Yi ZHOU Lin-qi
(Satellite Marine Tracking & Control Department of China, Jiangyin 214431, China)

Abstract Lustre reboot recovery algorithm needs that all clients reconnect to the server in a special recovery time window, and then clients resend uncommitted transactional requests and the server replays these requests strictly in the transaction number order. The recovery conditions are too strict. To improve Lustre's recoverability and availability, this paper proposed version based recovery and commit on share algorithms. They extend Lustre's metadata update algorithm and recovery algorithm respectively, and allow clients rejoin in the cluster by recovery under a more relaxed condition according to the dependence between transactions. At last, the performance of improved recovery algorithms was evaluated via a series of experiments.

Keywords Lustre, HPC, Recoverability, Availability

1 引言

在过去十年里,随着分布式数据存储和分布式文件系统的广泛应用,如何提供可用的数据服务受到了很多关注。Patterson在技术报告[1]中指出,随着存储技术的发展,人们越来越关注于存储系统的可用性和可维护性,可用性和可维护性的重要性将超过性能和扩展性。对于超大规模HPC集群系统,节点、存储设备、网络等发生故障的几率是不可忽视的。HPC集群文件系统必须具有相应的自恢复技术和容错技术,以满足可靠性、可用性和可服务性(Reliability Availability Serviceability,RAS)的要求。自动恢复的主要目的之一是提供高可用的连续性服务。为了支持连续性服务,一般要求当系统出现故障时,正在运行的应用程序仍能继续得到分布式文件系统提供的文件服务。自动恢复需要处理各种失效情况,本文主要研究最为重要的服务器失效恢复机制(关于网络失效和客户端节点崩溃的常见处理方法见文献[2])。当服务器崩溃后,要有其他备用服务器来接管失效服务器的工作(failover),或者失效服务器能够快速重启,恢复到失效前的状态。客户端具有请求重发机制,即当与服务器重新建立连接时,重发尚未完成的请求。对于无状态的文件系统[3],客户端和服务器状态不相关,服务器失效后,如果能快速重启或被故障接管,则仍可继续提供文件服务;对于有状态的分布式文件系统,服务器的崩溃失效引起的最大问题是服务器状态的丢失,从而导致服务器和客户端状态信息不一致。简单的方法是让所有的客户端重启或者直接清除它们的状态,使其与服务器保持一致,但这样做会导致大量的客户端应用被终止或报错,对用户来说是难以接受的。另一种方法是使用客户端保存的状态信息重构服务器的状态,它需要设计复杂的服务器状态恢复协议。

Lustre是HPC业界处于统治地位的集群文件系统,已经成为构建HPC存储系统的标准范例。有状态的Lustre目前的服务器重启恢复协议(即重放)要求所有的客户端以及客户端所有未提交的事务都参加,并且所有的客户端在检测到服务器失效后,要在指定的期限内与服务器重新建立连接,通过重新发送未提交的事务请求、重新打开已打开的文件、重放锁请求等重放过程重构并恢复那些故障前已经可见但并未提交

到稿日期:2014-05-20 返修日期:2014-08-10 本文受国家973计划资助项目(2009CB723803),国家自然科学基金资助项目(60873120)资助。
钱迎进(1981—),男,博士,工程师,主要研究方向为操作系统、集群文件系统,E-mail:coolqyj@163.com;李永刚 高级工程师,主要研究方向为测控软件系统、航天卫星发射;汪 毅 高级工程师,主要研究方向为测控软件系统、航天卫星发射;周琳琦 高级工程师,主要研究研究方向为测控软件系统、航天卫星发射。

的事务在服务器上的状态信息，而且重放事务序列中不允许有事务间隙，才能进行完整的恢复，要求过于严格[2,4]。对于一个超大的集群文件系统，网络故障、服务器失效以及客户端和服务器的双重失效等不再是不频繁的事件，而且有些慢速客户端可能在恢复窗口内来不及连接服务器，并直接被驱逐，所有这些都会导致重放的终止，所有受影响的客户端被驱逐。被驱逐的客户端要清除缓存的信息，并且可能会向应用程序报错，强制应用退出，从而不能进行透明的恢复。这对于一个大规模 HPC 集群是难以容忍的。为提高 Lustre 事务处理的可恢复性，提出了基于版本的事务恢复（Version Based Recovery，VBR）和共享时提交（Commit On Share，COS）算法，它们分别对 Lustre 现有的元数据更新和恢复机制进行了改进和扩展，根据事务之间的依赖关系，使客户端在更为宽松的条件下进行恢复，重新整合加入集群，减少被驱逐的客户端的数目。

2 相关研究

对于大规模 HPC 分布式存储系统，可恢复性对整个系统的可用性尤为重要。目前针对分布式文件系统文件服务恢复机制的研究较少。HA-NFS[5] 依附在无状态 SUN NFS 协议标准之上，通过双端口磁盘、磁盘镜像和网络复制实现了高可用的文件服务。与传统的复制方法相比，它没有过多的资源开销和性能下降，正在进行的操作在故障接管的过程中不会终止，恢复过程对应用完全透明。由于 NFS 是无状态协议，HA-NFS 只需要恢复相对较少的状态信息（例如文件锁、reply cache 等）。HA-NFS 是在 AIXv3 本地日志文件系统 JFS 之上的实现。它将有限的状态信息保存到 JFS 写前日志中，当备份服务器接管磁盘时，它会重放这些日志，重构 NFS 服务器的状态。有状态的 Calypso 文件系统[6] 使用状态重构技术和设备冗余来保证系统的高可用性。Calypso 的恢复策略保证打开的文件仍然保持打开，客户端修改的数据被保存，正在进行的操作在恢复后能够透明地完成。在其恢复策略中，各个客户端将其维持的状态信息（令牌、锁和打开的文件等）发送到重启或者故障接管的服务器，服务器据此重构系统的分布式状态信息。它保证了恢复期间数据的一致性，并提供拥塞控制。Sprite[7,8] 文件系统在状态重构中解决了与 Calypso 相同的设计问题。Sprite 的设计认为性能和缓存一致性要比状态恢复的简单性重要。Sprite 文件系统在服务器的内存中保存分布式缓存的状态信息，并使用相应的协议保证缓存的一致性。起初，Sprite 使用了与 Calypso 类似的客户端驱动的恢复策略[9]。Baker[10] 对其做了进一步研究，实现了两种新的恢复技术：服务器驱动恢复技术和透明恢复技术。服务器驱动技术中，重启服务器主动发起状态恢复的过程；透明恢复技术中，正常工作时，服务器将分布式文件系统状态信息存储在服务器上的非易失性内存中，当服务器失效重启后，利用非易失性内存中的信息恢复分布式文件系统的状态信息，而不必同客户端交互。

综上所述，分布式文件系统一般通过两种方式进行服务器失效恢复。一种为服务器驱动的恢复技术，它一般将服务器的状态信息保存在写前日志或非易失性内存等永久存储介质中，在服务器失效重启或被备用服务器故障接管后，通过永久存储介质中保存的信息来重构服务器状态，无需与客户端交互。但这种方式存在着缺陷：在服务器恢复后，某个客户端可能由于发生某种原因（永久性客户端硬件故障或网络故障）不能与服务器建立连接，那么服务器可能错误地恢复该客户端创建的状态信息，从而造成状态信息不一致的问题，如果该状态信息为授予失效客户端的锁，还会引发死锁问题。另一种方式为 Lustre、Calypso 和 Sprite 等文件系统所采用的客户端驱动的恢复技术。它的恢复过程一般如下：各个客户端重放它们维护的状态信息给服务器，服务器据此进行状态重构。这种方式不存在客户端与服务器状态信息不一致的问题。与 Calypso 和 Sprite 文件系统不同，Lustre 客户端跟踪保存每个引起服务器状态改变的请求，其中不仅包括锁、打开的文件等操作，还包括所有未提交到磁盘的元数据更新事务请求（如创建/删除文件、改变文件属性等操作），并在服务器失效恢复过程中严格按照未提交的请求的事务序列号进行重放[4]。在 Lustre 文件系统中，请求事务一旦在服务器上执行完毕完成内存提交，其结果对整个集群立即可见。此时，事务更新可能还没有提交到磁盘。由于内存和磁盘速度的差距，它能提供很好的吞吐率，但它允许元数据事务读或写其他未提交事务的数据，因此可恢复性差，分布式恢复协议更加复杂。前期工作[4] 对 Lustre 原有的元数据服务恢复机制进行了研究，本文提出的机制根据事务之间的依赖关系对 Lustre 文件系统的元数据更新和恢复机制进行了扩展和改进，使客户端在更为宽松的条件下进行恢复以提高 Lustre 的可恢复性。

3 基于版本的事务恢复

本节将提出一种基于版本的恢复机制。它对 Lustre 的恢复机制[4] 进行了扩展，允许客户端在更加宽松的条件下进行恢复。它的基本原理如下：在重放事务请求的过程中，基于版本的恢复允许客户端重放当且仅当客户端使用的对象与原来执行时的对象具有相同的版本；重放之后，如果一个客户端能够保证在恢复前它从服务器获得的数据以及创建的状态在恢复后能够呈现，而不是回滚，则恢复可以成功地完成。如果上述两个条件能够满足，则应用可以继续工作而不报错；如果不能满足，一般的结果是客户端被驱逐。基于版本恢复的另一个作用是它能提供对离线操作（Disconnected Operation）[11] 的支持。

3.1 术语与定义

下面首先给出一些基于版本事务恢复的术语和定义。

事务序列号（transno）：元数据服务器（MetaData Server，MDS）处理的每个引起其状态改变的客户端请求（如元数据更新、文件打开等）都被分配一个事务序列号（transno）。它由服务器进行管理，是一个服务器唯一的单调递增的 64 位整数。请求的 transno 被捎带在请求的回复消息中返回给客户端。在恢复时它被用来保证服务器接收到的客户端重放的请求事务按它们原来执行的顺序依次重新执行。

epoch：服务器用它来记录系统成功启动的次数。每次服务器重启且完成恢复后，epoch 的值就会加 1。

版本：以{epoch，transno}的形式来标记对象的版本。其中 transno 是事务执行过程中分配的单调递增的事务序列号。每次修改对象时，版本号发生改变，但并不是任何的修改都产生版本的改变，例如，时间或者大小的改变并不改变对象的版本号。

preop 版本：执行元数据操作之前索引节点对象被锁定后的版本号。

postop 版本：执行操作过程中设置的索引节点对象版本号。

主恢复阶段：第一个恢复阶段，服务器等待所有客户端重新建立连接，试图按照 transno 的顺序进行重放。

延迟恢复阶段：对那些不能及时建立连接的客户端进行恢复，时间比主恢复阶段要稍后。它对于主恢复期间由于网络分区等原因暂时不可用的客户端的恢复尤为有用。

事务序列号间隙：在 Lustre 事务请求的处理过程中，客户端在回复消息中获知事务的 transno，服务器恢复时严格按照事务的 transno 重放。当服务器开始重放过程时，它可能会发现至少有一个需要恢复的客户端没有重放（部分网络失效或客户端故障而未能加入集群）；或者当崩溃时，客户端接收到第 $N+1$ 个事务的回复（事务序列号为 $N+1$），但第 N 个事务的回复丢失，当所有客户端都重连后，没有客户端会重放事务 N。上述两种情况都会在恢复事务序列中产生间隙，可能导致不能进行完整的重放恢复。这种现象称为事务序列号间隙（Transaction Sequence Gap）。

驱逐（eviction）：客户端周期性地发送 ping 消息给服务器，表明它是活跃的，服务器的回复消息表明它也是活跃的。如果服务器长时间没有收到 ping 消息，它会清除客户端创建的状态信息，如撤销客户端获取的锁、关闭它的文件句柄等，这一过程称为驱逐。受影响的客户端只有在下次与服务器交互的时候才知道它被驱逐出集群，此时它必须丢弃缓冲的数据和持有的锁，进行恢复。

3.2 基于版本恢复算法

元数据更新会改变索引节点对象的版本号。索引节点元数据对象卷入的最后的事务的 transno 和服务器的 epoch 被用来表示和唯一标识版本号。同时，对 Lustre 后端文件系统 ldiskfs（改进的 ext3 文件系统）进行了扩展，将版本号内嵌保存在索引节点的磁盘数据结构中。一个元数据操作可能需要保存一个或多个对象的版本号。例如，向一个目录中添加目录项（link 操作）会更新目录索引节点的版本；setattr()操作会更新设置属性的索引节点的版本；create()操作会更新父目录的版本，同时为新创建的文件设置版本；而 rename()操作会涉及到 4 个索引节点对象版本的更新。元数据更新事务会为操作中涉及的各个对象设置相同的版本号。由于每次元数据的事务提交都要更新索引节点在磁盘上的数据结构，位于索引节点内的事务的版本也一并提交，它不需要额外的磁盘更新操作，因此对元数据性能几乎没有影响。

基于版本的重放算法在恢复过程中加入了版本检查，版本匹配的请求可以继续执行。基于版本恢复算法对 Lustre 的元数据更新进行了扩展。服务器在处理客户端发送的事务请求的过程中，会收集元数据操作涉及到的对象的 preop 和 postop 版本号，并将这些版本号捎带在回复消息中返回给客户端。客户端将这些版本号保存在客户端原来的请求中。这样客户端在重放的时候就可以在发送请求中捎带这些版本号给服务器。当重放请求时，客户端发送给服务器的操作请求中包含有涉及对象的 preop 和 postop 版本号。preop 版本用来进行版本检测，而 postop 版本号则存储在索引节点中作为新的版本号。在进行版本检查的过程中，服务器会试图执行客户端发送的每一个请求，即使重新整合（reintegration）发生失败。在对一个重放请求进行版本检查时，服务器会比较操作涉及到的索引节点对象的 preop 版本与重放请求的 preop 版本是否一致。如果版本是一致的，请求的重放可以继续进行，服务器会执行该请求；如果版本号不匹配，请求的重放过程返回一个错误。在重新整合的过程中，即使得到错误返回值，服务器仍然按照同样的方式处理后续的请求。

基于版本的重放恢复算法的工作过程如下：

1. 服务器因故障重启进行重放恢复时，首先进入主恢复阶段。重启的服务器完成初始化过程后，就开始接收来自客户端的连接请求。同时，服务器设置一个恢复时间窗口（如 300s），允许客户端进行连接。在时间窗口内连接到服务器的客户端被认为是正常恢复客户端。

2. 当客户端连接到服务器时，服务器首先告诉客户端它正处于主恢复阶段，同时告诉客户端它最后提交的事务信息。

3. 服务器开始接收来自客户端的重放请求，并严格地按照 transno 的顺序执行。如果请求的 transno 处于正确的顺序，满足正常重放恢复的要求，则执行它。如果整个恢复过程中没有遭遇事务间隙，则整个恢复过程可以顺利地完成。

4. 当发生事务间隙时，主恢复过程会一直等待下一个 transno 的事务。当发生恢复时间窗口超时时，服务器并不中止恢复，它会记录事务间隙信息，然后再等待一定的时间（如 100s），允许其他的客户端加入，企图弥补并关闭事务间隙。同时，在这段时间内，服务器会对已接收到的按 transno 排序的请求进行版本检查，根据重放请求的版本信息判断是否能够处理它。在这期间，服务器会获得所有的事务间隙信息。

5. 当主恢复窗口关闭后，服务器进入延迟恢复阶段。当客户端连接已经完成主恢复阶段的服务器时，服务器会在回复消息中置上延迟恢复的连接标志，并且将客户端最后提交的事务的 transno 和事务间隙信息（如果存在）返回给客户端。

6. 延迟恢复的客户端开始发送它的重放请求。所有的重放请求都基于版本检查进行重放。执行过程中，请求会获得新的事务序列号，但回复中的版本保持不变。在延迟恢复过程中，服务器接收到请求后会直接进行处理，处理过程中会检查是否有请求因版本不匹配而重放失败。如果发生版本不匹配，则客户端必须被驱逐；如果没有，客户端会得到重新整合成功的消息，此客户端完成恢复过程。

VBR 恢复对用户是完全透明的。如果在服务器恢复过程中存在几个客户端未加入集群，则它可能会需要更长的恢复时间。

3.3 事务间隙处理

当发生事务间隙时，服务器会记录第一个事务间隙序列号 first_gap_transno。客户端在恢复期间会用到该值，而且每次重放都会更新该值。当主恢复过程完成后，对于那些未连接的延迟恢复的客户端，服务器会在 last_rcvd[4] 日志文件中写下相应的事务间隙记录。这样，客户端在延迟恢复时就知道事务间隙信息。在此类客户端的恢复中，事务间隙可以随着恢复的进行而被弥补缩短，并进行更新。当发生多次失效时，可能会有来自不同 epoch 的事务间隙，所以只有具有相同 epoch 值的间隙才能被重放弥补更新。每个客户端都保存有最高被使用的事务序列号 highest_used_transno 以及它最后提交的事务序列号 last_committed。客户端每当收到来自服务器的回复消息时，便对上述两值进行更新。如果客户端的 last_committed 和 highest_used_transno 都比 first_gap_transno 要小，即对应客户端所有的事务发生在事务间隙之前，则客户

端能够进行完全的恢复。如果客户端的 last_committed 小于 first_gap_transno 而 highest_used_transno 比 first_gap_transno 要大，那么客户端能够进行部分恢复。客户端会比较它缓冲对象的版本的 transno 值，如果其大于 first_gap_transno，则清除该对象缓冲。当所有版本高于 first_gap_transno 的缓冲对象被清除后，客户端即完成恢复。如果有些缓冲的对象正在被使用，则客户端将会被驱逐。当 last_committed 大于 first_gap_transno 时，如果客户端在重放版本检查中没有遇到版本不匹配，则恢复过程与第二种情况相同；如果发现有重放请求版本不匹配，则客户端最终会被驱逐。

4 共享时提交

前面已经讨论过，当 MDS 失效后快速重启或被备用服务器故障接管时，客户端会重连服务器并重放它们未提交的请求来进行状态恢复。当一个或多个客户端因某种原因没有加入恢复时，就可能会造成恢复事务序列号间隙，导致所有依赖于事务间隙的其他客户端事务被 abort，依赖于被 abort 的事务又被 abort，如此反复，最后甚至会导致大量客户端被驱逐，这就是所谓的叠加 abort 问题。共享时提交(Commit on Share，COS)通过消除特定的依赖事务提供更好的可恢复性，从根源上解决这些问题。

如果客户端未被提交的事务能够被正确地执行，并且它们的缓冲能够重新生效，则客户端的恢复是成功的。如果在重放过程中没有存在依赖关系的未提交的事务，则客户端可以独立地重放它们的事务，而没有被驱逐的危险。出于性能的考虑，Lustre 原来的元数据更新算法只保持内存更新的隔离性，运行的事务可以访问(读取或写入)其他事务写过但未提交的对象的脏数据，这是导致叠加 abort 的主要原因。为了避免叠加 abort，提高事务处理的可恢复性，必须保证每个事务只读或写已提交事务写过对象。也就是说，消除对未提交事务的依赖，在进行元数据操作前，提交依赖事务到磁盘，可以避免叠加失效的发生。为了保证单个客户端的性能，定义了一种更为宽松的依赖关系。

客户端间依赖事务：属于依赖事务的一种。事务 B 客户端间依赖于事务 A，当且仅当：(1)事务 B 读或写已经被事务 A 写过的对象；(2)事务 A 和事务 B 分别来自不同的客户端。

恢复依赖性：客户端 a 恢复依赖于客户端 b 当且仅当客户端 a 和客户端 b 分别有这样的未提交的事务 T_a 和 T_b：T_a 客户端间依赖于 T_b。除非客户端 b 在恢复中能够成功地完成，至少能够成功地恢复事务 T_b，否则客户端 a 必须被驱逐。

COS 就是通过保证客户端间依赖事务不读或写未提交的数据来消除恢复依赖性，达到较强的可恢复性。由于客户端之间没有事务依赖关系，因此来自不同客户端的事务就可以独立地重放恢复。COS 的实现原理是：当服务器检查到客户端间依赖事务时，通过将未提交的客户端间依赖事务写到磁盘来避免读或者写未提交的对象的数据。图 1 给出了一个 COS 实例。客户端 Client1 调用 mkdir 命令创建一个目录 'deirecory'，然后客户端 Client2 调用 touch 'directory/file'命令在该目录下创建一个文件'file'，由于 Client2 的元数据操作会更改目录'directory'加入一个名为'file'的目录项，因此两个元数据操作存在 WRITE→WRITE 客户端间依赖关系，故 MDS 必须先提交元数据操作 mkdir 'directory'的结果，才能开始执行 Client2 的元数据操作。由于检测到依赖冲突时 COS 要进行强制事务提交，在某些情况下对性能有一定的影响，因此，在具体实现中 COS 是可选的，以便用户在性能和可用性之间做出选择。

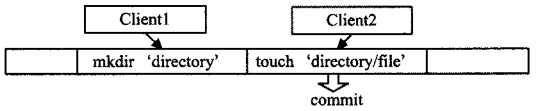

图 1 享时提交(COS)实例

Lustre 使用分布式锁管理器(Distributed Lock Manager，DLM)来实现对元数据访问的并发控制[12,13]。Lustre 支持 6 种锁模式，元数据操作主要用到了其中 3 种：保护读(PR)、保护写(PW)和互斥(EX)。当 MDS 接收到客户端发送的元数据操作请求时，它首先会获取涉及的所有对象上(目录或索引节点)的锁，然后执行元数据操作，完成内存更新，最后发送回复消息给客户端通知请求已经执行完成。此时 MDS 并不立即释放执行过程初期获得的锁，这些锁会被保留，它会在两个地方可能被释放：(1)事务的提交回调中；(2)客户端接收到 MDS 回复时会发送一个轻量级的 REP-ACK 消息，MDS 接收到该确认消息后会释放对应事务请求获得的锁。

利用 Lustre DLM 的锁冲突来检测客户端间依赖事务。首先对 DLM 的锁进行了扩展：在每个对象锁中增加了客户端跟踪信息，用它来识别发起事务请求的客户端；然后定义并引进了一种新的 COS 锁模式，当 COS 锁发生冲突时，必须提交该锁关联的事务。COS 锁仅存于对象资源的授权队列，而且一般是由写锁(如 PW 或 EX 锁)转换得到的。COS 锁不能被请求，也不可能位于资源的等待队列中。COS 锁与 PR/PW/EX 锁的兼容性如表 1 所列。

表 1 COS 锁的兼容性

requested lock	COS lock
PR / same client	compatible
PW / same client	compatible
EX / same client	compatible
PR / another client	no compatible
PW / another client	no compatible
EX / same client	no compatible

COS 算法还需要对元数据服务器的事务处理进行扩展。当元数据更新事务完成时，事务执行初期获得的锁并不马上释放，服务器会将它们保存起来，而且其中的 PW 锁会转化专门的 COS 锁并保留直到事务被提交。在进行锁模式兼容性检查时，会利用前面提到的锁对象的客户端跟踪信息来判断客户端间事务依赖关系。根据 COS 锁的兼容性表 1 可知，如果请求的新的 PR/PW 锁与 COS 锁都来自于同一个客户端，则锁是兼容的；如果来自不同的客户端，则表明检测到锁冲突。当另一个客户端访问对象，检测到客户端请求的锁和已经存在的 COS 锁发生冲突时，在锁冲突阻塞回调中，服务器会强制进行事务提交。在事务的提交回调中，与事务关联的 COS 锁会被释放。这里请注意，当保护对象访问的锁是 EX 模式时，其处理过程与 PW 锁相同。

REP-ACK 锁机制[4]与 COS 机制相类似，都是为了达到事务处理的不同级别可恢复性。REP-ACK 锁机制等待事务的回复消息到达客户端后或在事务的提交回调中释放事务处理中获得的锁，然后才允许后面依赖的事务(如果存在)被处理，它可在一定程度上避免事务间隙的发生；COS 机制提供更高级别的可恢复性，当发生 WRITE→READ 或 WRITE→

WRITE客户端间依赖冲突时，它要等待事务提交到永久存储介质才释放与事务关联的COS锁，避免了客户端间的恢复依赖性，理论上各个客户端可以进行独立并行的恢复。COS机制和REP-ACK机制不能同时使用。当使用COS时，REP-ACK必须关闭；当使用REP-ACK机制时，COS必须关闭。

在COS开启的Lustre集群中进行恢复时，VBR算法用来处理未提交的请求。当两个来自不同客户端的未提交的请求更新了同一个对象，而其中一个客户端由于某种原因不能加入恢复时，对于这样的重放请求，VBR原本是不能进行重放恢复的。而当Lustre文件系统开启COS功能后，它会消除此类客户端间依赖的事务，任何连接的客户端都可以通过重放进行恢复，并使客户端缓冲重新生效，重新整合到集群中。

5 性能评测

当发生客户端间依赖关系时，服务器要强制进行事务提交，将脏数据同步写到磁盘。如果这种同步过于频繁，会对元数据性能有所影响。本节对各种情况下的元数据性能进行了对比模拟评估。

用作测试平台的集群配置如下。

• 客户端：4台，双核Intel Xeon E5410 2.33GHZ CPU，内存4GB，MT23108 SDR Infiniband适配器，安装X86_64 Linux操作系统。

• MDS：1台，双核Intel Xeon E5410 2.33GHZ CPU，内存4GB，MT23108 SDR Infiniband适配器，4×80GB ST380815AS SEAGA ATA磁盘，磁盘配置成RAID1+0阵列，安装X86_64 linux操作系统。

• OSS：与MDS配置相同，但磁盘被配置为2个RAID1阵列。

以下是执行的各种测试用例：

(1)使用Lustre专用的元数据性能测试工具mdsrate进行并行文件创建(create)测试，4个客户端，每个客户端在不同的独立目录下创建文件，其性能如表2所列(速率单位为tps)。从实验结果可以看出，在该测试用例中关闭和开启COS功能，两者的性能相当。主要原因是，客户端是在不同的目录上各自创建文件，元数据操作间没有客户端间依赖关系，使用COS功能并没有导致额外的强制磁盘提交。该实验结果证明了COS能够保证单个客户端的性能。

表2 不同目录并行文件创建性能对比

COS	No.	Creates	Time(s)	Rate (tps)	Avg. rate (tps)
Disable	1	889074	300	2964	2891
	2	857109	300	2857	
	3	855734	299	2852	
Enable	1	856731	300	2856	2846
	2	849962	300	2833	
	3	857846	301	2850	

(2)使用mdsrate进行并行文件创建测试，4个客户端在一个共享目录下创建文件，其性能如表3所列。

表3 共享目录下并行文件创建性能对比

COS	No.	Creates	Time(s)	Rate (tps)	Avg. rate (tps)
Disable	1	799486	30	2655	2710
	2	800607	31	2668	
	3	842740	32	2809	
Enable	1	555338	300	1851	2210
	2	698681	300	2329	
	3	734986	299	2452	

从实验结果中可以看出，开启COS功能后，文件创建性能下降了近20%。这是由于4个客户端在一个共享目录中并行创建文件，会发生大量的客户端间依赖事务，引发过多的磁盘提交而导致性能下降。

(3)跨节点创建删除测试：客户端*A*在安装点mnt1进行文件创建，客户端*B*在安装点mnt2删除同样的文件，测试在两个客户端间并行执行。表4显示了测试结果。

表4 不同安装点创建删除性能对比测试

COS	No.	Files	Time(s)	Disk transcations
Disable	1	10000	30	6
	2	10000	31	6
	3	10000	32	6
Enable	1	10000	39	10894
	2	10000	38	10609
	3	10000	39	10711

从表4可以看出，在并行创建10000个文件并在不同节点将其删除的测试中，关闭COS功能的测试用例只触发了6次磁盘提交，而开启COS功能的测试平均进行10738次磁盘提交。最后结果显示关闭COS测试完成的平均用时为31s，而开启COS测试完成的平均用时为39s，开启COS后性能降低了26%。

以上各个测试用例的结果证明，COS算法能够保证单个客户端的性能不受损失，但是当频繁地发生客户端间依赖元数据事务操作时，它会触发过多的磁盘提交，导致元数据必须频繁地同步到磁盘，破坏了内存缓冲带来的批量磁盘提交的优点，导致过多的磁盘寻道操作，降低了系统的性能。

结束语 在分布式文件系统中，性能和可恢复性是两个相互冲突的需求。Lustre在设计上着重考虑了性能，采用了较为复杂的基于事务的有状态的分布式恢复协议，恢复要求过于严格。Lustre允许服务器完成了元数据事务的内存更新就可以将结果返回客户端，而且还允许事务读或者写其他未提交事务的数据。这种方式能够充分利用内存的高速缓冲提供优异的元数据性能，但它可能会在服务器重启恢复(或者故障切换)时造成事务的叠加abort的问题，导致不能进行无缝透明的恢复。Lustre的重启恢复算法需要集群中所有客户端在指定的恢复时间窗口内与服务器重新建立连接，客户端重传未提交的事务请求，服务器严格按照事务序列号重放所有未提交的事务，其要求过于严格。针对可恢复性不强的缺点，提出了基于版本恢复和共享时提交的算法，它们允许客户端在更为宽松的条件下能够进行恢复加入到集群而不被驱逐。基于版本的恢复算法在恢复的过程中加入了版本检查，如果事务操作对象的版本匹配，则允许事务进行重放恢复；它是一种基于事务对象的恢复算法，而且对性能几乎没有影响。在共享时提交算法中，服务器一旦检测到未提交的客户端间依赖事务，会将它提交写到磁盘来避免读或者写未提交的事务的数据，从而消除客户端间的恢复依赖关系，使得各个客户端可以独立地恢复。实验评估证明，共享时提交算法在发生事务依赖时需要强制进行磁盘提交，对性能有所影响。尽管如此，在超大规模的Lustre集群中，为保证能够提供高可靠、高可用的服务，一般都会选择开启共享时提交功能。

参 考 文 献

[1] Patterson D. Availability and Maintainability >> Performance: New Focus for a New Century [EB/OL]. http://usenix.org/e-

vents/fast02/patterson/sld001.htm

[2] 钱迎进. 大规模 Lustre 集群文件系统关键技术的研究 [D]. 长沙:国防科技大学,2011:97-118
Qian Ying-jin. Research on Key Issues in Large Scale Clustered File System Lustre [D]. Changsha: National University of Defense Technology,2011:97-118

[3] 李晖. 基于日志的集群文件系统高可用关键技术研究[D]. 北京:中国科学院计算技术研究所,2005:9-10
Li Hui. Research on Journal Based High-availability Mechanism of Cluster File Systems [D]. Beijing: Institute of Computing Technology, Chinese Academy of Science,2005:9-10

[4] 钱迎进,伊瑞海,肖侬,等. Lustre 文件系统元数据服务恢复机制研究 [J]. 高性能计算技术(第 19 届全国信息存储技术大会收录),2013,255(6):10-16
Qian Ying-jin, Yi Rui-hai, Xiao Nong, et al. Research on Recovery Mechanism for Lustre Metadata Service [J]. High Performance Computing Technology (19th Annual National Conference on Information and Storage Technology),2013,255(6):10-16

[5] Bhide A, Elnozahy E N, Morgan S P. A Highly Available Network File Server [C]// Proceedings of the Usenix Winter 1991 Conference. Dallas, TX, USA: USENIX Association,1991:199-205

[6] Devarakonda M, Kish B, Mohindra A. Recovery in the Calypso File System [J]. ACM Transaction on Computer Systems,1996, 14(3):287-310

[7] Mogul J C. Recovery in Spritely NFS [J]. Computing Systems, the Journal of the USENIX Association, Spring, 1994, 7(2): 201-262

[8] Baker M, Ousterhout J. Availability in the Sprite Distributed File System [J]. Operating Systems Review,1991,25(2):95-98

[9] Welch B, Baker M, Douglis F, et al. Sprite Position Statement: Use Distributed State for failure Recovery [C]// Proceeding of the Second Workshop on Workstation Operating System. Pacific Grove, CA, USA: IEEE Computer Society,1989:130-133

[10] Baker M. Fast Crash Recovery in Distributed File Systems [D]. California: University of California at Berkeley,1994:34-104

[11] Kistler J, Satyanarayanan M. Disconnected Operation in the Coda File System [J]. ACM Transactions on Computer Systems, 1992,10(1):3-25

[12] 钱迎进,金士尧,肖侬. Lustre 文件系统 I/O 锁的应用与优化 [J]. 计算机工程与应用,2011,47(3):1-5,26
Qian Ying-jin, Jin Shi-yao, Xiao Nong. Application and Optimization for Lustre File I/O Locking [J]. Computer Engineering and Applications,2011,47(3):1-5,26

[13] 钱迎进,肖侬,金士尧. Lustre 分布式锁管理器的分析与改进 [J]. 计算机工程与科学,2009(S1):146-149
Qian Ying-jin, Xiao Nong, Jin Shi-yao. Analysis and Improvement of Lustre Distributed Lock Manager [J]. Computer Engineering & Science,2009(S1):146-149

(上接第 176 页)

[4] 何东,尹青,谢耀宾,等. 反编译中数据类型自动重构技术研究 [J]. 计算机科学,2012,39(5):133-136
He Dong, Yin Qing, Xie Yao-bin, et al. Automatic data type reconstruction in decompilation [J]. Computer Sciense, 2012, 39 (5):133-136

[5] 马金鑫,李舟军,忽朝俭,等. 一种重构二进制代码中类型抽象的方法[J]. 计算机研究与发展,2013,50(11):2418-2428
Ma Jin-xin, Li Zhou-jun, Hu Chao-jian, et al. A reconstruction method of type abstraction in binary code[J]. Journal of Computer Research and Development,2013,50(11):2418-2428

[6] Ding Wei, Gu Zhi-ming, Gao Feng. Reconstruction of data type in obfuscated binary programs[C]// 16th International Conference on Advanced Communication Technology. PyeongChang, South Korea,2014:393-369

[7] Balakrishnan G, Reps T. WYSINWYX: What you see is not what you execute[J]. ACM Transactions on Programming Languages And Systems,2010,32(6):202-213

[8] Balakrishnan G, Reps T. DIVINE: discovering variables in executables[C]// Proceedings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation. Nice, France,2007:1-28

[9] Anand K, Elwazeer K, Kotha A, et al. An accurate stack memory abstraction and symbolic analysis framework for executables [C]// 29th IEEE International Conference on Software Maintenance. Eindhoven, Netherland,2013:90-99

[10] Cousot P, Cousot R. Interpretation: A unified lattice model for static analysis[C]// Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. New York, USA 1977:238-252

[11] 王雅文,宫云战,肖庆,等. 基于抽象解释的变量值范围分析及应用[J]. 电子学报,2011,39(2):296-302
Wang Ya-wen, Gong Yun-zhan, Xiao Qing, et al. A method of variable range analysis based on abstract interpretation and its applications[J]. ACTA Electronica Sinica,2011,39(2):296-302

[12] Brumley D, Jager I, Avgerinos T, et al. BAP: A binary analysis platform[C]// 23rd International Conference on Computer Aided Verification. Snowbird, UT, USA,2011:463-469

[13] Lee J, Avgerinos T, Brumley D. TIE: Principled Reverse Engineering of Types in Binary Programs[C]// Proceedings of the Network and Distributed System Security Symposium. San Diego, USA,2011: session 5

[14] Song D, Brumley D, Yin Heng, et al. BitBlaze: A new approach to computer security via binary analysis[C]// 4th International Conference on Information Systems Security. Hyderabad, India, 2008:1-25

[15] Aho A V, Lam M S, Sethi R, et al. Compilers: Principles, Techniques, and Tools (2nd Edition)[M]. Boston: Addison Wesley, 2007

[16] 刘絮颖. 反编译中控制流重构与控制结构恢复技术研究[D]. 郑州:解放军信息工程大学,2011
Liu Xu-ying. Research on technology of control flow reconstruction and control structure recovery in decompilation[D]. Zhengzhou: PLA Information Engineering University,2011

[17] Durfina L, Kroustek J, Zemek P, et al. Detection and recovery of functions and their arguments in a retargetable decompiler[C]// 19th Working Conference on Reverse Engineering. Kingston, Canada,2012:56-60

[18] 吴滨. 汇编级程序辅助分析中的库函数识别技术研究[D]. 郑州:解放军信息工程大学,2011
Wu Bin. Research on library function identification technology in assemble level program auxiliary analysis[D]. Zhengzhou: PLA Information Engineering University,2011

[19] Jing Jing, Jiang Lie-hui, Liu Tie-ming, et al. A precision-tunable CFG reconstruction algorithm[C]// International Conference on Mechatronic Sciences, Electric Engineering and Computer. Shenyang, China,2013:2095-2099