一种基于授权机制的分布式文件系统小文件访问优化策略①

曹风华

(内蒙古财经大学 计算机信息管理学院, 呼和浩特 010070)

摘 要: 针对分布式文件系统应用于海量小文件访问模式时, 元数据请求过多导致系统性能下降的问题, 提出了客户端元数据缓存授权机制的解决方案. 客户端从服务器读取元数据时, 申请相应类型的授权, 服务器分析请求并决定是否授予此客户端所访问的元数据的授权. 若客户端成功获取授权, 则将其与本地缓存的元数据相关联, 作为缓冲有效性的凭证. 当再次访问本地缓存的元数据时, 若有相关授权, 则可以直接从本地获取元数据, 无需向服务器发送缓存数据的有效性验证 RPC. 仿真实验表明, 文中的方法有效的降低了客户端发送元数据请求 RPC 的数量, 节省了宝贵的网络带宽资源, 降低了元数据服务器的负载.

关键词: 分布式文件系统; 海量小文件; 元数据缓存; 授权机制; pNFS;

Optimization Strategy Based on Delegation Mechanism for Distributed File System in Massive Small File Access Patterns

CAO Feng-Hua

(Computer Information Manage College, Inner Mongolia University of Finance and Economics, Hohhot 010070, China)

Abstract: Aiming at the system performance degradation problems of distributed file system in massive small file access patterns, designed and implemented a delegation mechanism for meta data cache in client .The client apply some kinds of delegations when getting meta data from the server. The meta data server checked and decided ecide whether to grant the delegation or not. Once the client getting the delegation which is stored in the local memory, the meta data getting operation can be unnecessary to sent RPC request. The simulation result shows that our method in this paper can reduce the number of the RPC request effectively, and aving valuable network bandwidth resources, reducing the metadata server load.

Key words: distributed file system; massive small files; metadata cache; authorization mechanism; pNFS;

web2.0、社交网络等新兴应用的兴起, 海量小文件的访问性能问题日益严重. 分布式文件系统作为存储管理数据的关键技术, 面临着海量小文件的严峻的挑战. 海量小文件访问性能问题的本质是元数据操作比例升高, 数据局部性缺失, 元数据热点瓶颈问题进一步加重, 导致单个小文件操作延迟过大. 如何降低元数据的访问次数成为解决问题的关键. 目前对海量小文件的访问优化主要从访问方式和存储组织方式两方面展开研究. 前者包括 I/O 通路分离技术[1]、元数据预取技术[2]、服务器文件预创建技术[2,3]、I/O 聚集[2,3]、Cache 技术[4,5]; 后者的典型代表是元数据分割方法[6]、命名空间扁平化[7]、大目录高效组织[8]、文件聚集元数据内嵌数据[9]. 由于基于存储方式的优化技术需要对存储布局做出修改, 基于此优化技术难以控制复杂度, 且通用性以及扩展性较差. 为达到系统设计实现简单效果稳定并有较好的通用性的目的, 本文提出一种新的基于访问方式的分布式文件系统小文件访问优化策略: 基于授权的元数据客户端缓存管理策略. 对于客

① 收稿时间:2012-12-25;收到修改稿时间:2013-03-01

户端缓存的每一份元数据, 服务器都授予与之相关的一个授权, 以保证其有效性. 当客户端访问本地缓存的元数据, 若其拥有授权, 则不必再向元数据服务器再次获取元数据的请求. 实验数据表明, 授权机制有效地减少了网络开销并降低了作为系统热点的元数据服务器的负载, 对海量小文件的访问优化有显著效果.

1 典型分布式文件系统及其架构

分布式文件系统给用户提供了大容量, 高性能, 高可靠, 高可用, 高可扩展, 统一命名空间, 便捷数据管理的基础设施. 针对非结构化数据的共享管理, 分布式文件系统被视为最优选择.

1.1 典型分布式文件系统介绍

从分布式文件系统的发展来看, 伴随着信息数据的迅猛增长, 传统单服务器文件系统如 NFS、CIFS, 已经无法满足应用对文件系统可扩展性和 I/O 性能方面的需求. 一些新型的系统如 IBM GPFS、Panasas PanFS、Lustre 使得客户端能够直接访问存储设备, 较好的解决了 I/O 单点瓶颈问题, 可以提供较高的数据访问带宽, 但是仍没有很好的解决海量小文件访问的低延时、高并发问题. 此外, 由于这些系统具有一定的专用性, 依赖特定的软硬件平台不具通用性, 增加了用户实施和培训成本. pNFS 作为 NFSv4 协议扩展, 通过带外直接访问存储设备, 较好的解决了 NFS 性能瓶颈问题, 同时兼有 NFS 平台的广泛兼容性, 这使得 pNFS 很有可能成为一种并行文件系统的标准而被广泛使用.

1.2 pNFS 介绍

虽然 pNFS 还没有到大规模商用阶段, 但是已经有比较稳定的 beta 开源实现, 并已经进入 Linux 内核源码目录. 在 pNFS 框架中主要有三种角色: 客户端(client), 元数据服务器(Metadata Sever), 存储设备(Storage Device), 如图 1 所示:

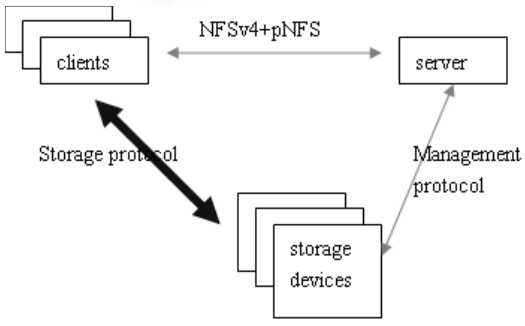

图 1 pNFS 总体结构图

存储设备可以是块设备, 文件设备, 或者是对象设备, 该设备必须通过某种方式(例如在 Linux 系统中, 使用 mount 命令)被被客户端和服务器所共享, 才能实现带外访问.

元数据服务器负责管理元数据的查询和更新, 并需要处理所有客户端的元数据操作的请求. 当客户端和存储设备上文件数量增多时, 元数据服务器极易成为系统的热点瓶颈.

客户端访问文件系统时, 需要首先和元数据服务器交互取得相关文件在存储设备上的映射信息(即元数据), 然后根据映射信息通过存储协议来直接访问存储设备上的数据.

存储协议为客户端提供直接访问存储设备协议, 当前有文件协议(NFSv4 以及 NFSv3), OSD, 块协议(iscsi, NBD, FC, IB).

管理协议是元数据服务器和存储设备之间的协议, 主要用于存储资源的分配与回收, 参与一些与客户端状态相关的管理.

每次访问文件客户端首先要从服务器端获取大量的元数据信息, 然后再根据元数据信息读写后端存储设备. 在小文件应用环境, 每一次读写的数据量很少, 但由于目录解析等操作元数据的访问反而增加. 这种结构下的读写流程适用于大文件, 不适用于小文件. 文献[1]提出了一种针对 pNFS 小粒度读写优化的带内外混合读写方式, 其在尽量减少对 pNFS 结构的修改之下, 针对小规模环境下的小粒度 I/O 应用模式取得了较好优化效果. 这种结构的实质在读写粒度上设定一个阈值, 当读写粒度少于某个值, 它就通过带内的方式进行读写. 这种方式可以减少进行小粒度 I/O 与 Layout 相关的 RPC 操作. 但是这种方式有以下几个缺点:

(1) 小文件这种应用模式下, 由于元数据操作占用大部分时间, 这种结构只是针对小文件数据操作, 没有考虑元数据操作优化的问题, 因此该技术对大文件的小粒度数据 I/O 访问优化效果较好.

(2) 它使得原先 pNFS 为减轻元数据服务器单点瓶颈而设计的带外访问机制失去效果, 加重 MDS 的负载, 设计上不够优雅, 并且固定粒度的阈值设计也不够灵活.

(3) 带内外混合访问, 带来额外的一致性语义维护的难度.

2 元数据缓存授权机制描述

2.1 元数据缓存与一致性问题

分布式文件系统中, 客户端为成功访问文件数据通常需要从服务器端获取必要的元数据信息, 例如索引节点(inode)文件属性以及存储设备上的布局信息. 过多的元数据流正是影响海量小文件访问性能的关键因素. 通常为减少网络开销和访问延迟, 元数据信息会在客户端本地缓存起来, 以供下次客户端可以直接从本地内存中快速读取. 而客户端所缓存的元数据存在超时失效的问题, 失效后需要再次和服务器通信, 对所缓存的元数的有效性进行验证. 例如在 NFSv3 采用的就是超时更新机制维护缓存数据的一致性. 缓存超时更新机制就是说客户端缓存的数据在被读取过来的某一段时间是有效的, 这个时间在 mount 时可以指定. 当超时之后, 通过发送元数据请求, 这仍然是不可忽略的网络开销. 在保证元数据一致性的前提下, 为尽量减少缓存失效, 本文提出了缓存授权机制.

2.2 授权机制描述

元数据授权机制是一种元数据一致性访问保证机制, 当客户端获得某个元数据授权的时候, 可以看成发出授权的服务端给了客户端一种承诺, 能够保证其他客户端对该目录的操作不会造成对文件系统一致性语义的冲突. 具体来说, 获得授权的客户端在访问本地缓存的相关元数据时候, 不必担心有其他客户端对该目录的数据做出了修改, 使得操作在本地命中缓存直接返回. 例如, 在 pNFS 中, 当客户端授权关联的操作主要有 READDIR, ACCESS, LOOKUP, GET_ATTR. 这些操作都会去进行缓存的有效性验证(revalidate)操作, 如图 2 所示, 其目的查看与该文件相关的元数据是否有效. 增加元数据的授权后, 就不必再发送网络请求, 可以直接从本地中读取信息.

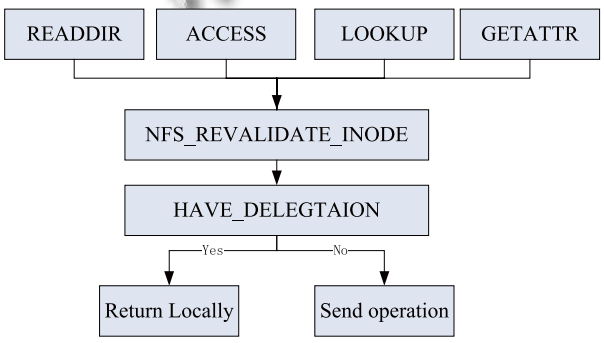

图 2 持有授权操作访问流程图

为了保证文件系统的可用性, 如果其他客户端有冲突访问, 文件系统的授权必须是可以召回的, 因此目录授权适用读写访问共享小的环境, 文献[10]表明在一些常用 NFS 的 trace 当中目录读写访问共享在 3.5%-4%之间, 因此召回的成本是可以接受的.

客户端元数据缓存授权机制总体流程描述如下:

(1) 客户端首次和服务器通信获取元数据时, 说明要申请的授权类型, 发出请求.

(2) 服务器决定是否授予这个客户端目录授权, 如果可以, 返回这个目录的 cookiverfier 和这目录授权的 stateid, 并且在服务器记录这个目录授权.

(3) 客户端收到服务器返回的结果后, 判断是否成功获得目录授权, 成功获得则在本地记录目录授权.

(4) 客户端访问本地缓存的元数据时, 如果该数据在缓存命中, 并且持有授权, 则认为是有效的, 直接返回; 如果没有命中, 该操作发送到服务器端. 在没有授权的情况, 即使是命中缓存, 首先也得检查缓存数据是否超时, 如果超时, 该操作需要发往服务端进行有效性验证.

(5) 如果服务端察觉到目录冲突, 则同步召回和某个元数据相关的所有授权.

(6) 如果客户端收到召回通知, 则同步返回目录授权.

(7) 召回超时, 则服务器强制使客户端的目录授权失效.

(8) 客户端卸载(Umount)时, 客户端返回并清除授权.

3 实验结果与分析

本文在开源的 pNFS 分布式文件系统之上设计和实现了元数据缓存授权机制. 由于本文要优化的问题是海量小文件访问, 因此测试采用的标准工具和测试场景都针对小文件访问模式. 为了突出本文设计实现方法的有效性, 我们将以和原系统(无授权机制的 pNFS 文件系统)对比测试为主. 实验环境和相关数据如下表所示:

表 1 客户端配置表

客户端(2 台)	
CPU	Intel(R) Xeon E5606 2.13GH 16 核 64 位
物理内存	DDR3 2GB*6
网卡	千兆网卡
HBA 卡	Qlogic 4Gb FC 卡
操作系统	Suse11u1 2.6.32.12-0.7-default

表 2 元数据服务器列表

元数据服务器(1 台)	
CPU	Intel(R) Xeon E5606 2.13GH 16 核 64 位
物理内存	DDR3 2GB*6
网卡	千兆网卡
HBA 卡	Qlogic 4Gb FC 卡
操作系统	Suse11u1 2.6.32.12-0.7-default
元数据卷	本地单盘希捷 SAS146GB

表 3 共享存储列表

共享存储列表	
CPU	Intel(R) Xeon E5606 2.13GH 16 核 64 位
CPU	DDR3 2GB*6
物理内存	Qlogic 4Gb FC 卡
HBA 卡	Suse11u1 2.6.32.12-0.7-default，scst target 2.2.0
操作系统	FC 导出 2 个卷：单盘希捷 SAS146GB，双盘希捷 SAS146GB*2 RAID0
导出卷	Intel(R) Xeon E5606 2.13GH 16 核 64 位

本文采用 filebench 测试了元数据缓存授权的效果, Filebench 是一款标准的文件系统性能的自动化测试工具, 它通过快速模拟真实应用服务器的负载来测试文件系统的性能. 它不仅可以仿真文件系统微操作(如 copyfiles, createfiles, openfiles, readfile, writefile), 而且可以仿真复杂的应用程序(如 varmail, fileserver, oltp, dss, webserver, webproxy). 我们选取了两种小文件较多的应用场景进行测试: Fileserver 和 Webserver 场景, 对比测试了只读目录授权优化前后客户端发送 RPC 操作的次数.

测试方式采用一台客户端和一台服务器, 客户端挂载服务器导出的目录, 用 filebench 测试程序测试挂载目录, 利用该工具自带的两种场景(Fileserver 和 Webserver), 选 filebench 工具默认配置的参数, 测试结果如下:

(1) 针对 Fileserver 场景, 测试时间为 60 秒(提高幅度=减少数目/优化前的操作数):

表 4 Fileserver 测试结果

	Getattr	Readdir	Lookup	Access	Rpc
授权机制	50	42	2163	9099	59660
原有方式	1124	2054	10809	11149	82591
减少数目	1074	2002	8646	2050	22931
提高幅度	95.3%	97.4%	80.0%	18.4%	27.8%

(2) 针对 Webserver 场景, 测试时间为 60 秒(提高幅度=减少数目/优化前的操作数):

表 5 Web server 测试结果

	Getattr	Readdir	Access	Lookup	Rpc
授权机制	49	43	59	1020	96114
原有方式	1102	2046	1090	8612	116053
减少数目	1053	2003	1031	7592	19939
提高幅度	95.5%	97.8%	92.6%	88.1%	17.1%

从测试结果看出 Getattr 和 Readdir, Lookup, Access 这样的操作可以大幅度减少, 除了这些操作之外其他相关类型操作也有减少只不过幅度不大, 为了看出整体效果, 本文除了统计减少幅度比较大的操作和总的 RPC 操作次数, 针对 Fileserver 这种场景, 测试结果显示总的 RPC 操作次数可以减少到 27.8%, Webserver 这种场景, 减少幅度虽然没有 Fileserver 这种场景大, 但是也可以减少总的 RPC 操作 17.1%, 上述测试结果反映出授权机制对元数据操作的减少.

4 结论

海量小文件访问性能瓶颈是当前分布式文件系统所要面临的难题, 随小文件应用模式逐渐覆盖了许多分布式文件系统应用领域, 解决小文件访问的性能瓶颈的问题越来越迫切. 本文研究分析了小文件访问性能瓶颈问题, 提出了一种授权机制减少元数据缓存的有效性, 降低网络开销和元数据服务器的负载, 从而提高整个系统的访问性能. 实验数据表明, 授权机制简单有效的减少了客户端获取元数据的 RPC 数量. 在本文现有的工作基础之上, 下一阶段的工作可以从以下两个方面来展开: 1)进一步扩展现有的只读和可写两种授权类型, 减少授权召回的开销, 使得授权机制更加灵活高效; 2)本文是基于 pNFS 分布式文件系统实现的授权机制, 需要进一步扩展扩展性和可迁移性, 使其适用于其他分布式文件系统.

参考文献

1 Hilderbrand D, Ward L, Honeyman P. Large files, small writes,and pNFS. Proc. of the 20th Annual International Conference on Supercomputing. 2006: 116−124.

2 Carns P, Lang S, Ross R, Yannur MV, Kunkel J, Ludwig T. Small-file access in parallel file systems. IPDPS, 2009.

3 Cluster File Systems Inc. Lustre: A Scalable, High-Performance File System. www.lustre.org, 2002.

4 Tomkins A, Patterson RH, Gibson G. Informedmulti-process

(下转第 176 页)

表 1 不同参数格攻击实验结果

n	|m|	|w|	|e|	|f|	背包密度 n/|m|	实验次数	成功率
32	33	32	21	21	0.9696969697	10	100%
64	65	65	37	38	0.9846153846	10	100%
128	129	128	70	71	0.992248062	10	50%
256	257	257	135	136	0.9961089494	10	30%
512	513	511	265	265	0.9980596823	10	10%
1024	1025	1024	521	522	0.9990243902	10	10%

5 结语

本文利用LLL算法对一种基于二元一次不定方程的改进背包密码算法进行了格攻击. 以计算实验的方式证明即使背包密度在安全区间内也并不能保证背包方案的安全性, 虽然实验并不是百分之百的成功, 但也说明了格攻击算法的有效性以及文献[3]中改进算法的不安全性. 至于理论方面的证明, 将在下一步的研究中进行.

参考文献

1 Merkle RC, Hellman MH. Hiding information and signatures in trapdoor knapsacks.IEEE Trans.on Info.Theory,1978,IT-24 (5):525−530.

2 Shamir A. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. IEEE Trans. on Information Theory, 1984,30(5):699−704.

3 费向东,潘郁.安全背包公钥密码的要点和设计.信息网络安全,2012,(9):81−84.

4 古春生,于志敏,景征俊.基于随机背包公钥密码的格攻击.计算机应用研究,2012,(9):3486−3488.

5 Shoup V. NTL.a library for doing number theory. [2009-08-14].http://shoup.net/ntl/

6 Lenstra AK, Lenstra HW, Lovaz L. Factoring Polynomials with Rational Coefficients. Math Ann, 1982(261): 515−534.

7 Coster MJ, Joux A, LaM acchia BA, et al. Improved low-density subset sum algorithms. Computational Complexity, 1992,2(2):111−128.

8 Ajtai M. Generating hard instances of lattice problems. Proc. of the 28th Annual ACM Symposium on Theory of Computing. 1996: 99−108.

(上接第 186 页)

prefetching and caching. Proc. of the 1997 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems. Seattle, Washington, 1997: 100−114.

5 Patterson RH, Gibson GA, Ginting E. Informed prefetching and caching. 15th ACM Symposium on Operating System Principles. 1995.

6 Anderson TE, Dahlin MD, Neefe JM, Patterson DA, Roselli DS, Wang RY. Server-less network file systems. ACM Trans. on Computer Systems, ACM, February1996,14(1):41−79.

7 Hendricks J, Sambasivan RR, Sinnamohideen S, Ganger GR. Improving small file performance in object-based storage. Technical Report 06-104, Parallel Data Laboratory, Carnegie Mellon University, 2006.

8 Patil S, Gibson G. Scale and Concurrency of GIGA+: File System Directories with Millions of Files. Proc. of the 9th USENIX Conference on File and Storage Technologies (FAST'11). San Jose CA, February 2011.

9 https://btrfs.wiki.kernel.org/.April 24,2012.

10 Radkov P, Yin L, Goyal P, Sarkar P, Shenoy P. A Performance Comparison of NFS and iSCSI for IP-Networked Storage. Proc. of the 3rd USENIX Conference on File and Storage Technologies. March 31, 2004, San Francisco, CA.