EDFUSE:一个基于异步事件驱动的FUSE用户级文件系统框架

段翰聪 王勇涛 李 林

(电子科技大学计算机科学与工程学院 成都 611731)

摘 要 开源FUSE文件系统用户模块实现方式采用多线程并发模型,在高并发条件下,线程间的同步将降低系统的吞吐率,增加响应时间。基于流水线分段数据通信思想和异步事件网络驱动模型,消除线程间的同步,通过优化文件和元数据缓存来提高缓存命中率等方式,实现了异步事件驱动的FUSE用户级文件系统的用户态框架。实验结果表明,在大量请求环境下系统的吞吐率得到提高。

关键词 用户级文件系统,多线程模型,异步事件驱动,流水线

中图法分类号 TP316 **文献标识码** A

EDFUSE: FUSE Framework Based on Asynchronous Event-driven

DUAN Han-cong WANG Yong-tao LI Lin

(School of Computer Science & Engineering, UESTC, Chengdu 611731, China)

Abstract FUSE is a classical framework of developing filesystem in userspace conveniently, it is composed of kernel module and userspace module. However, userspace module is developed in multithreading context; the number of threads is in proportion to the number of requests, this means that it is explosive growth when plenty of requests are reached, it is harmful for system performance and responsive time. A novel asynchronous event-driven FUSE(EDFUSE) framework using pipeline is proposed to optimize the cache of data and metadata to increase the hit ratio of cache. Finally, the experiment studies show that the EDFUSE framework gains better throughput and efficiency.

Keywords Filesystem in userspace, Multithreading model, Asynchronous event-driven, Dataflow-pipeline

1 引言

文件系统是Linux操作系统内核组成部分之一,它向Linux虚拟文件系统(VFS)注册一系列操作函数。当VFS接收来自应用层文件操作请求后,根据注册的回调函数及文件系统类型进行相应的处理。

上面为传统文件系统开发的基本思想,所有实现均是在内核态完成。由于网络文件系统需要与网络进行交互,使得开发过程太过复杂,增加了开发难度。为了提高开发效率,Linux内核增加了对开发用户级文件系统的支持。其将内核中文件系统的调用导出至用户态,使得开发文件系统就类似于开发普通应用程序。

FUSE全称为"Filesystem in Userspace"[1],是一个用户态文件系统的框架[1,8]。内核模块向VFS注册FUSE文件系统(类似ext3传统文件系统),用户将文件系统挂载至FUSE上,当应用程序访问FUSE文件系统时,FUSE内核模块就将请求发送至设备/dev/fuse,FUSE用户态通过监听/dev/fuse来读取请求并进行处理时,将结果通过该设备发送至内核[9,10]。实现原理如图1所示,当用户访问FUSE文件系统下的文件(/tmp/fuse)时,FUSE内核态将请求发送至设备/dev/fuse的读取队列,当FUSE用户态从该设备读取请求并进行处理时,将结果通过该设备回写至FUSE内核态,FUSE内核态将处理结果发送至VFS,这样,应用程序就得到返回结果。

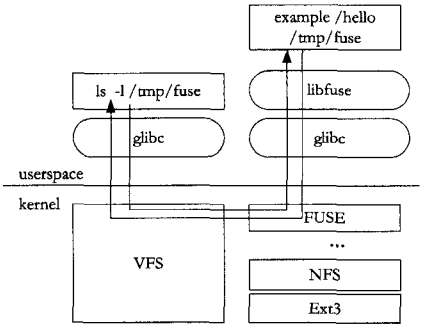

图1 FUSE原理[1]

FUSE开发者在开发内核模块后,实现了用户态的框架[1],经过源码分析,该用户态架构如图2所示。

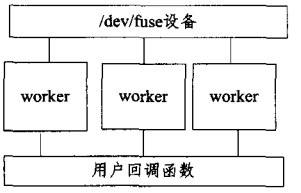

图2 FUSE框架

本文受国家"新一代宽带无线移动通信网"科技重大专项中子课题(2010ZX03004-001-02,2011ZX03002-003-02,2012ZX03002-004-004),广东省部产学研基金(2010A090 200082)资助。

段翰聪(1973—),男,博士,副教授,主要研究方向为云计算、内容分发网络等,E-mail:duanhancong@gmail.com;王勇涛(1987—),男,硕士生,主要研究方向为分布式计算;李 林(1981—),男,博士,副教授,主要研究方向为计算机网络服务质量、网络安全等。

系统在初始化时，创建一个 worker 线程来监听内核请求，获取到一个请求之后，该线程就会处理该请求，最终调用用户注册的回调函数，如果用户注册的回调函数出现阻塞，则该 worker 线程陷入阻塞。主线程必须确保始终有线程在监听/dev/fuse 设备，但如果监听的数量超过 10 个，主线程就会取消多余的线程。在实际的测试工作中，也证实了这一点[5]。

NetBSD 系统下的 FUSE 版本 puffs[2,3] 是一个基于 FUSE 进行修改的版本。为了保持独立性及兼容性，该库也采用 FUSE 一样的设计思想，即多线程版本 worker 模型，每个线程处理来自内核的一个请求。Windows 环境下的 FUSE[4] 也采用类似的设计思路。基于以上分析，该同步请求线程与请求的数量成正比，在应用程序单线程的环境下，该框架吞吐量存在优势。但是在多线程的环境下，线程的创建与退出、请求的数量的波动会造成线程数量急剧地伸缩，同时在线程之间共享的数据会出现同步，线程进入阻塞状态，多线程就失去了快速响应的优势；在系统大量请求的环境下，该框架将大大降低系统的吞吐量。另外，FUSE 用户态框架在缓存方面只缓存了文件的元数据信息，而未缓存对应的文件数据。

本文介绍的框架将改进原系统的不足，并根据具体使用场合来进行优化；设计并实现基于异步事件驱动的 FUSE 用户态框架 EDFUSE，以提高系统性能。

2 FUSE 文件系统框架的设计

2.1 设计思想

多线程并发模型在线程数量少的情形下，具有响应速度快、系统延时小等优势，但在大量请求下，由于线程间的同步互斥，会大大降低系统的处理效率。

为了消除并发模型的劣势，但又需利用多核的优势，就需要采用流水线式设计思想：将系统划分为多个功能相互独立的模块，每一个模块即是一个线程。当其中一个线程收到请求后，处理该请求并将该请求根据请求类型发送至下一个线程。每个线程均为流水线中的一段，处理完与之相关的请求后，继续处理下一个请求，当没有请求时，就进入阻塞状态。根据请求的复杂性，会出现后到达请求先返回的异步处理方式。

2.2 总体设计

为了保证系统的高吞吐量及模块独立性，系统采用了流水线式的设计思想，即将系统按功能及独立性划分为多个模块，每个模块之间均采用线程方式监听来自该模块的请求，为了保持模块之间的独立性，模块之间采用域方式或管道方式通信，当系统监听到一个请求之后，根据请求的类型，路由至不同的模块。

EDFUSE 框架整体设计流程如图 3 所示，系统共有 4 个模块，即创建 4 个线程，每一个线程负责一个模块，其中 fuse 设备监听模块主要通过监听/dev/fuse 设备来读取来自内核的请求，并将请求根据文件不同的操作类型进行调度以及处理用户登录；Cache 模块主要缓存文件的元数据信息及内容信息，DiskIO 模块负责从磁盘读/写具体文件数据，SocketIO 模块则处理所有网络读写。文件的元数据缓存在内存中，数据缓存在磁盘。

当系统初始化时，系统向服务器发送用户登录请求，如果登录失败，则系统提示错误信息并退出，当登录成功后，系统进行挂载，开始读取服务器的根目录信息，并进行缓存。当初始化完成后，系统开始监听内核的文件操作请求。各个模块均采用 epoll 进行异步驱动来监听各个模块的请求，处理请求完成后，对应线程进入睡眠状态。

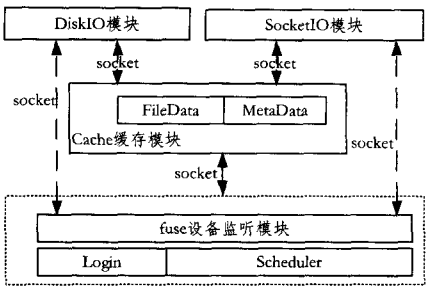

图 3 系统设计

与原框架不同的是，当模块线程等待的请求未获取到回复时，模块并未处于阻塞状态，它可以接收其它模块的请求并对其进行处理或直接将请求路由至其余模块，这样所有模块线程的处理方式就与原框架有明显的区别。

本系统主要满足高性能的计算需要，用于保存一些配置文件及二进制程序或运行二进制程序，应用场景如下：

(1) 大部分为小文件，其中涉及的操作都是小文件的随机读写。在 Linux 服务器环境中，都是以小文件为主，在文件操作方面，可以针对这一特点进行性能优化。

(2) 服务器支持基本的文件上传和下载请求(目前版本不支持文件的随机读写)。由于都是小文件，因此可以进行文件操作语义上的优化，如在文件打开时，客户端就自动下载云端文件至本地目录缓存，之后的所有读写操作都直接本地化；而文件关闭时，客户端将修改的文件上传至服务器，这样就避免在文件读写期间与服务器之间的频繁交互。

(3) 用户在使用文件系统之前需要进行登录操作。在应用环境中，服务器为分布式文件系统，在使用该系统前，都需要进行身份验证，登录成功后，就会转入个人主目录进行下一步的操作。

2.3 典型流程分析

系统所使用的文件操作语义类似于早期的 AFS[7,11]，由于目前服务器只支持基本的文件上传、下载语义，文件的操作语义就针对服务器提供的语义进行对应的调整。如写文件语义，就直接向本地缓存中写，并置为“脏“标志位，当文件关闭时，直接将该文件上传至服务器；当上层应用程序打开文件成功，文件数据就缓存至本地磁盘上，后面的读写过程均在本地进行。针对目录操作，则首先向服务器发送对应的目录操作请求，当服务器返回请求后，向用户线程返回操作结果。

值得注意的是，系统在处理多线程访问同一文件时，需要根据请求的类型进行一些串行化处理，如：两个线程同时打开同一个文件，为了防止数据出现不一致的情况，就向服务器发送一次文件下载请求进行并对两个请求进行串行化操作，当该请求返回后，系统就会向两个请求返回同一结果。相反，在文件关闭操作时，如果文件出现脏，处理方式就采用最后提交，最后的更新版本方式来实现。如果文件正处于上传过程中，而另一线程要删除该文件则该线程就需要等待上传操作的完成。在处理中断操作请求时，如果被中断的操作在等待队列中，则直接将该请求取消；如果处于处理状态，则将该请求的中断标志置为“真”，当返回至监听线程回写设备时，会检查该中断标志，以确定进一步的操作。

框架是为了满足云存储项目而开发的，其中的部分设计思想是根据项目本身的特点而展开的，还有部分设计思想是来自原 FUSE 框架对其进行改善而得到的。与原框架类似，

在使用该框架时，也是通过注册文件操作回调函数来进行处理。

3 实验结果

在系统测试过程中，下层的网络传输均采用相同的操作库，上层为原 FUSE 框架与 EDFUSE 框架，两框架都执行相同的操作。采用的测试环境为：客户端与服务器配置均为 SUSE Linux 系统虚拟机，内存为 512MB，Xeon(R)双核。测试命令为 cp，rm，对应的操作分为下载、上传及删除，其中文件数据缓存在内存文件系统中。测试文件详细信息如表 1 所列。

表 1

数量	460	1000	3000	5000	7000	10000
大小	76M	149M	476M	756M	2.5G	3.0G

通过 cp 命令将服务器上的文件夹拷贝至本地的方式来测试下载文件时的性能。从图 4 可以看出，在下载量少于 5000 个文件时，两框架花费时间接近，当下载文件个数为 5000 时，EDFUSE 框架花费的时间与 FUSE 框架相等，随着文件数量的增加，两框架之间的差距有明显增大趋势。

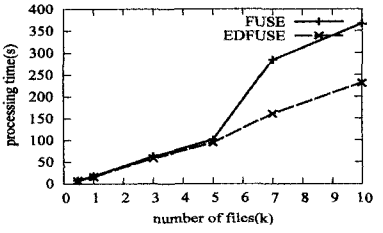

图 4 下载文件

通过 cp 命令将本地文件夹拷贝至服务器的方式来测试上传文件时的性能。从图 5 可以看出，在下载量少于 5000 时，两框架花费的时间接近，在上传文件达到 5000 时，EDFUSE 框架的性能与 FUSE 框架花费时间相同，在上传文件达到 7000 时，EDFUSE 框架花费时间少于 FUSE 框架；并随着文件数量的增加，差距有增大趋势。

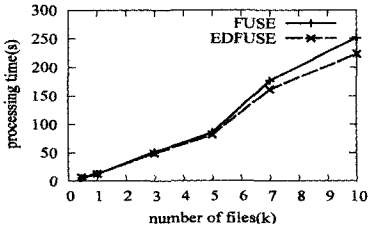

图 5 上传文件

通过 rm 命令删除服务器文件夹的方式来测试删除文件时的性能。如图 6 所示，原框架删除时间与文件个数成线性关系，在删除文件小于 10000 时，新框架花费时间少于原框架，但当删除文件接近 10000 时，两框架所花费的时间开始接近。

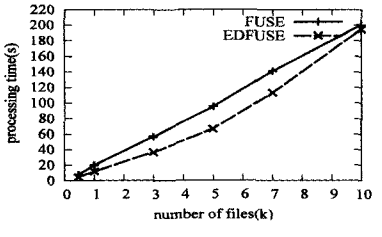

图 6 删除文件

在上传 10000 个文件时，通过每隔 5s 的方式来统计 CPU 的利用率(见图 7)，FUSE 框架采用多线程模式，而 EDFUSE 框架采用异步模型。当请求处理完成时，线程均处于空闲状态，而多线程的处理情况则大不一样，在运行过程中需要进行同步，这样就会降低 CPU 利用率。从图中可以看出，在 43s 左右，EDFUSE 操作完成，而 FUSE 框架继续处理用户请求，在系统 CPU 利用率方面，EDFUSE 框架利用率高于 FUSE 框架。

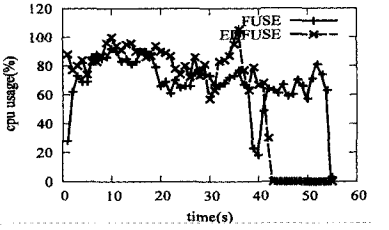

图 7 CPU 使用率

通过实验看出，在下载文件和上传文件操作时，两个框架之间花费的时间的差距增大，但在删除文件操作，两框架之间的时间差距较小，但在 CPU 利用率方面，由于采用异步事件驱动模型，EDFUSE 框架线程数量远远低于 FUSE 框架，因此新框架(EDFUSE)CPU 利用率高于原框架(FUSE)。

结束语 遵循 FUSE 协议[6]并根据系统的需求开发文件系统框架，可以与实际需求有效地结合，并提高系统吞吐率，具有很好的灵活性与扩展性，适用于分布式文件系统和网络文件系统领域。

本文提出 FUSE 文件系统目前存在的性能问题及应用场合，改进并实现了一个基于事件驱动的用户级文件系统框架，将其应用于分布式文件系统。从以上分析可以看出，在局部操作中新框架的性能略低于原框架，但是整体性能仍处于优势。该框架的开发目前处于初级版本状态，在封装性、兼容性方面还有待进一步加强。

参 考 文 献

[1] FUSE: Filesystem in Userspace[EB/OL]. http://fuse.sourceforge.net

[2] Kantee A. puffs-Pass-to-Userspace Framework File System [C]// AsiaBSD-Con 2007. Tokyo, Japan, 2007

[3] Kantee A, Crooks A. Refuse: Userspace FUSE Reimplementation Using puffs[C]// AsiaBSDCon 2007. Tokyo, Japan, 2007

[4] Driscoll E, Beavers J, Tokuda H. FUSE-NT: Userspace File System for Windows-NT [OL]. http://page.cs.wisc.edu/~driscoll/fuse-nt.pdf, 2008

[5] FUSE 测试报告[OL]. http://www.blog.chinaunix.net//space.php? uid=20687780&do=blog&id=371672

[6] FUSE Protocol[EB/OL]. http://research.cs.queensu.ca/~wolfe/misc/fuse/fuse-7.8.html

[7] AFS [EB/OL]. www.openafs.org

[8] 李涛，梁洪亮. 具有事件恢复功能的文件系统的研究与实现[J]. 计算机科学，2009，36(3)：270-274

[9] 栾亚建. 分布式文件系统元数据管理研究与优化[D]. 广州：华南理工大学，2010

[10] 吴宗坤. 基于 Fuse 的资源搜索文件系统设计与实现[D]. 广州：华南理工大学，2011

[11] Sidebotham B. Volumnes: The Andrew File System Data Structuring Primitive[C]// EUUG Conference Proceedings. Manchester, UK, 1986