复内核操作系统文件系统的设计与实现

王国伟[1,2],王彩芬[1],朱俊杰[2]
WANG Guowei[1,2], WANG Caifen[1], ZHU Junjie[2]

1.西北师范大学 计算机科学与工程学院,兰州 730070
2.中国科学院计算技术研究所 先进计算机系统研究中心,北京 100080
1.College of Computer Science & Engineering, Northwest Normal University, Lanzhou 730070, China
2.Research Center of Advanced Computer Systems, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China

WANG Guowei, WANG Caifen, ZHU Junjie. Design and implementation of file system on multi-kernel operating system. Computer Engineering and Applications, 2016, 52(7):43-49.

Abstract: With the increased resource density especially the many-core architecture in a single server, the scalability of an Operating System (OS) and low resource utilizations become serious problems. Operating system researchers have made many efforts in solving these problems by running multiple kernels within a server, including a replicated kernel OS, Popcorn Linux. But for cloud and big data applications, the file system plays an important role in data storage, which is not implemented in Popcorn Linux. It is because file systems in the traditional single-kernel operating system cannot be directly ported to a replicated kernel OS. A new file system is designed for Popcorn Linux. This paper proposes a method of realizing a user space file system, POPFUSE, based on the FUSE (Filesystem in User Space) framework, which allows all kernels to access their respective file systems without needing directly attached disk controllers. The stability of the communication is guaranteed by the way of sharing memory, and that also improves the efficiency of the file system. The paper conducts extensive evaluations on the performance of POPFUSE. The experimental results show that the overall performance of the multi-kernel operating system is improved when using POPFUSE compared to NFS (Network File System).
Key words: Filesystem in User Space(FUSE); POPFUSE; file-system; multi-kernel operating system

摘 要:随着云计算、大数据进一步的发展,促使提供计算服务的单个节点的硬件性能不断的提升,但数据中心资源利用率较低,且可扩展性较差的问题始终存在。人们试图从各个方面解决这个问题。复内核操作系统Popcorn Linux就是其中一个比较典型的解决方案。文件系统作为操作系统的重要组成部分,直接影响着数据中心应用的执行效率。传统的文件系统因为磁盘控制器的原因,无法移植到复内核操作系统上,从而难以满足新形势下的需求。针对这个问题,提出了一种全新的适用于复内核操作系统的文件系统POPFUSE。该文件系统基于FUSE框架实现,解决了因磁盘控制器有限,多个内核实例无法同时访问磁盘资源的问题,通过共享内存的方式,保证了通信的稳定,提高了文件系统的效率,进而促进了多个内核的操作系统整体性能的提升。
关键词:FUSE框架; POPFUSE; 文件系统; 复内核操作系统
文献标志码:A **中图分类号:**TP316.4 doi:10.3778/j.issn.1002-8331.1511-0106

1 引言

随着云计算与大数据应用的兴起,数据中心作为向互联网用户提供数据、计算、存储、软件等服务的平台越来越受到人们的关注。在数据中心硬件平台的发展过程中,CPU经历了单处理器、多机互联、片上多处理器、片内多核、片内众核,进而由PCI-E总线或在片内集成

基金项目:国家自然科学基金(No.61163038,No.61562077,No.61262057)。
作者简介:王国伟(1984—),男,硕士研究生,主要研究领域为操作系统,E-mail:wangguowei@ict.ac.cn;王彩芬(1963—),女,博士研究生,教授,研究领域为云计算、无线传感器网络;朱俊杰(1989—),男,硕士,主要研究领域为操作系统。
收稿日期:2015-11-09 **修回日期:**2016-01-13 **文章编号:**1002-8331(2016)07-0043-07
CNKI网络优先出版:2016-03-02, http://www.cnki.net/kcms/detail/11.2127.TP.20160302.1655.028.html

异构,成为异构众核等一系列发展阶段。随着数据中心硬件平台逐渐向多核平台、众核平台方向发展,数据中心资源利用率低的问题表现得更加明显。在这种情况下,为了提升数据中心资源利用率,解决众核平台下的可扩展性问题,以Popcorn为代表的复内核操作系统成为了新的研究趋势。

本文致力于为像Popcorn这样由多个不同的内核所组成的复内核操作系统设计开发出能够使用的文件系统。由于传统的类UNIX[1]操作系统是单内核的操作系统,这些操作系统所使用的文件系统都是完全占用并且直接掌控底层硬件的。每个系统内核都需要一套自己的硬件资源,这就导致在复内核操作系统的开发过程中每个Primary OS的文件系统必须占用一个磁盘控制器,才能完成文件的访问。如果将传统的文件系统直接移植到复内核操作系统的每个内核中,就必须给每个Secondary OS的文件系统分别分配一个磁盘控制器。然而每个硬件系统中的磁盘控制器的个数是有限的,大多数情况下,单个节点上仅有一个磁盘控制器。因此无法给每个Secondary OS都分配独享的磁盘控制器。所以传统的Linux文件系统在众多内核所组成的操作系统上是无法直接使用的。这就促使人们必须寻找其他的办法来解决这个问题。

传统解决办法是采用NFS,即网络文件系统。因为NFS允许网络中的计算机之间通过TCP/IP网络来共享磁盘资源。在NFS的应用中,本地NFS的客户端应用访问远端NFS服务器上的文件是透明的,如同访问本地文件一样。但是NFS也有不如意的地方。首先它与内核之间的耦合性高,所以导致开发起来难度较大。其次主要体现在性能上,由于同一时间只能允许单一的一个TCP会话读/写NFS的数据,这导致其性能受到了限制。再次是NFS在动态负载均衡方[2]面开销太大。最后NFS主要是基于主机身份验证来识别身份的,从而完成访问请求,但是这种RPC/Mount验证机制过于简陋,在RPC远程调用中,一个SUID的程序就已经是具有超级用户权限,从而无法保证绝对的安全。而且NFS都是明文传输的,传输过程中数据是很容易被窃取和篡改的。无法保证数据的安全性与完整性,而且NFS会占用额外的网络带宽,影响数据中心应用的网络通信性能。针对以上问题,本文采用了用户空间文件系统FUSE[3-8]框架来开发出满足复内核操作系统的文件系统。现有的,通过FUSE框架开发的文件系统都是针对单内核操作系统设计开发的,还没有针对复内核操作系统进行过设计与开发。如果简单移植现有的用FUSE框架开发的文件系统,依然会出现上面所提到的硬件资源即磁盘控制器不够用的情况。因此,本文选择在FUSE框架下设计开发出能够适用于复内核操作系统的全新的文件系统POPFUSE。

之所以选用FUSE框架来进行开发,首先是因为它能够降低了开发的难度,这样就能够缩短开发周期,从而降低开发成本。其次,就是多路径、多线程的同时访问。NFS由于同一时间只能允许单一的一个TCP会话读/写NFS的数据,这导致其性能受到了限制,而FUSE则没有这方面的限制,所以使得FUSE的优势一下子体现了出来。第三,就是FUSE能够支持的系统相当广泛,已经具备了针对众多系统开发出了接口,例如:2.6.14以后的Linux内核版本,Mac OS X[9]、OpenSolaris[10]、FreeBSD[11]和NetBSD[12]等等。最后FUSE的框架不存在网络传输安全性的问题。

2 相关工作

2.1 应用平台Popcorn

Popcorn复内核操作系统[13]是由美国弗吉尼亚理工学院提出的一个全新概念的操作系统,该操作系统是由多个内核所组成,它包含一个作为全局资源管理和任务调度者的管理操作系统实例(即文中提到的Primary OS),以及若干个作为资源使用者和应用执行环境的可定制的负载操作系统实例(即本文中提到的Secondary OS),其中的每个内核都完成各自相应的功能。该操作系统目前是一个开源的系统,还处于研究阶段,在其研究中已经有成熟的设计和部分功能的实现,但是缺少文件系统的开发。本文将以此作为平台,在Popcorn现有的基础之上开发出一个全新的适用多个内核所组成的操作系统的文件系统POPFUSE。

2.2 用户态文件系统FUSE

用户空间文件系统(Filesystem in User Space,FUSE)是一个开源项目,是在用户空间中开发文件系统的框架。Linux是从2.6.14版本开始支持此框架。FUSE主要包括内核模块和用户空间守护进程两个部分:

(1)内核模块只是体现了一个完整的文件系统运行时的框架,它接收来自VFS的调用,然后通过一个专用的守护进程将内核的磁盘访问请求提取到用户空间,使得VFS层可以对FUSE文件系统和其他的文件系统一视同仁,接下来开发人员可以在用户空间实现具体的操作,从而实现完整的文件系统。

(2)程序员通过用户空间库提供的编程接口来进行文件系统的开发,具体通过实现FUSE提供的FUSE_operations或FUSE_ll_operations的全部操作或部分操作,来实现文件系统;而且通过一系列的实践证明它们之间有着良好的兼容性。FUSE的工作原理[8]如图1所示。

图1是一个基于FUSE所描绘的用户态文件系统hello的文件操作流程,图中展示了FUSE的工作原理。这里规定基于FUSE的用户态文件系统hello会被挂载到/tmp/FUSE的目录下。假定应用程序要访问/tmp/FUSE

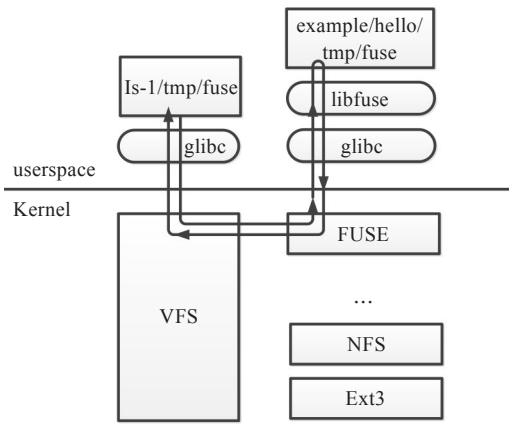

图1 FUSE的工作原理

目录下的某个文件,就会通过busybox中的函数来产生系统调用,接下来VFS[14-15]将会产生响应,进而来处理这些系统调用,接着VFS则会调用位于内核中的FUSE模块。FUSE模块在接收到用户发来的调用请求后,将请求发送给用户态文件系统hello。最后用户的文件读写请求将由用户态文件系统进行相应的处理。最终处理后的结果会通过刚刚调用所经过的路径返回给用户。这样就完成了整个文件系统的访问。

之所以选择使用FUSE框架来开发[16],因为它有如下的优势:

(1)通常来说设计实现一个文件系统是很繁琐的一项工作,但是通过FUSE框架来开发,使得整个开发过程犹如写普通的用户态程序,快捷简便。FUSE的API库不但简单而且功能强大,这就为开发功能完备的文件系统提供了强有力的保障。FUSE把内核态一系列复杂的接口给隐藏起来,实现了一个统一的面向用户的用户态接口。接下来用户在开发过程中,只要方便地调用相应的接口就可以实现自己的文件系统了。这样使得开发者不必太在意文件系统模块与系统内核模块之间的相互关系,可以把主要的精力都放在用户态文件系统的开发上。

(2)通过FUSE框架开发的文件系统,使得文件系统的操作从内核态转移到了用户态,将应用程序同内核有效地隔离开来,使应用程序不在直接和内核相互作用,从而提高了系统的安全性和稳定性。

(3)一般来说内核态的代码开发调试相对来说都很困难,开发效率较低。通过使用FUSE这个开发框架,可以在很大程度上提高开发效率[17],极大地简少了为操作系统提供新的文件系统的工作量,从而缩短开发周期,降低了系统软件的开发成本。

2.3 NFS文件系统

NFS(Network File System)即网络文件系统,是FreeBSD支持的文件系统中的一种,NFS允许网络中的计算机之间通过TCP/IP网络共享资源。NFS文件系统主要分为三个部分:网络协议、NFS客户端和NFS服务器。NFS客户端提供了接口,保证用户或者应用程序能像访问本地文件系统一样访问NFS文件系统,NFS服务器作为数据源,为NFS客户端提供真实的文件系统服务,而网络协议则使得NFS客户端和NFS服务器能够高效和可靠地进行通信。NFS网络协议使用的是RPC(Remote Procedure Call,远程过程调用)/XDR(External Data Representation,外部数据表示)机制。

(1)节省本地存储空间,将常用的数据存放在一台NFS服务器上且可以通过网络访问,那么本地终端将可以减少自身存储空间的使用。

(2)用户不需要在网络中的每个机器上都建有Home目录,Home目录可以放在NFS服务器上且可以在网络上被访问使用。

(3)一些存储设备如软驱、CDROM和Zip(一种高储存密度的磁盘驱动器与磁盘)等都可以在网络上被别的机器使用,这可以减少整个网络上可移动介质设备的数量。

3 设计与实现

3.1 POPFUSE架构设计

POPFUSE是通过使用FUSE框架来开发的,适用于复内核操作系统的文件系统,在使用时需要Linux内核与FUSE用户态程序相互有效的配合,即FUSE用户态程序会接收内核传递出的文件系统操作请求,然后用户态程序才能进行相应处理,具体的处理是需要用户来实现的。图2为POPFUSE的整体结构图。

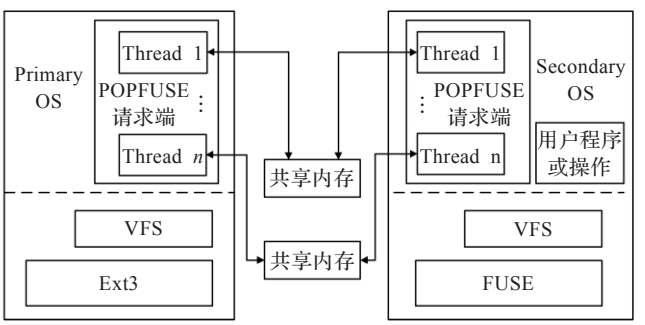

图2 POPFUSE整体结构

如图2所示,POPFUSE文件系统工作过程中主要包括六部分:

(1)Secondary OS端和Primary OS端都存在的VFS模块。VFS是一个文件系统的转换层,是内核中实现的一个软件层,内核通过它来实现虚拟文件系统,通过它可以处理对Unix标准文件系统的系统调用。VFS对上层的用户提供简洁统一的文件操作调用接口,当上层的用户调用读写等函数时,内核则调用特定的文件系统实现。文件具体在内核内存中是以一个名叫file的数据结构来记录管理的。在Secondary OS端,它的功能是接收来自用户程序的文件访问操作。它将接收来的

访问操作处理后,通过调用FUSE在内核中的文件系统传递给FUSE。

(2)Secondary OS端内核FUSE模块。与VFS的接口则是由内核FUSE模块来实现的,其中包括FUSE文件驱动模块的注册,FUSE 的(虚拟)设备驱动,提供supper block、dentry、inode的维护等等,VFS模块将请求发送给内核FUSE模块,然后再由该模块传递给用户态程序的接口进行相应的操作。

(3)Secondary OS端FUSE用户态程序,主要作用是对FUSE模块接收到的请求进行处理,使得Primary OS端更容易识别,然后通过调用ioctl将处理后的文件操作请求写入共享内存发往Primary OS,由于FUSE的工作线程是不断创建和销毁的,所以线程在创建时还需要获取本身应用使用的共享内存信息。

(4)Primary OS端用户态程序,该程序需要一个Secondary OS ID作为参数,程序运行时根据该参数创建多块与该Secondary OS共享的内存,并将创建的可用的共享内存ID通知Secondary OS。以上工作完成后接下来创建多个工作线程,然后主线程进入睡眠状态,每个创建的线程在一段时间内轮询一块共享内存,查看是否有文件操作请求到达,若一段时间没有请求到达则改变共享内存相关状态位并进入睡眠状态,等待Secondary OS端的中断唤醒。当线程发现有请求到达时,则调用本地文件系统对该请求进行处理并返回处理结果。当接收到POPFUSE退出的命令后,主线程会从睡眠状态唤醒并通知工作线程退出,然后释放相关资源终止程序。为了更好地实现资源隔离,该程序在启动时还会根据Secondary OS ID将自身添加到对应的CGroup中。

(5)共享内存,POPFUSE文件系统通过使用共享内存的方式来处理内核间的通信。创建共享内存的功能已经在Popcorn中实现了,它是通过环形缓存的方式来实现的。这种通信方式既快速又高效,因此可以直接来使用它,而不必自己设计开发。接下来的文件数据和文件操作命令的传递都是通过这一方式进行的。

(6)Ext3模块。图中Ext3所代表的是具体的Primary OS所包含的Linux文件系统,也就是在Linux中使用的文件系统。每个传递到Primary OS的文件访问请求都是由它来具体执行磁盘访问请求的。它通过调用磁盘驱动程序操作磁盘控制器来对文件进行读写访问。每个通过VFS传递过来的文件访问请求,都是由它来完成相应的访问。

POPFUSE的运行机制如图2所示,在每个Secondary OS端运行FUSE的用户态程序。当有文件访问请求到达时,请求首先传达到Secondary OS端的VFS层,VFS收到请求后便调用本地的FUSE,FUSE通过FUSE的用户态程序对请求进行相应的处理,然后将请求发往Primary OS。如图每个请求都是以线程的形式组成线程请求队列。Primary OS在接收到Secondary OS端通过共享内存发来的文件请求后,Primary OS对请求进行具体的处理,然后将请求传递给本地的VFS,通过调用Ext3即实际的Linux文件系统来完成文件访问操作。最后将结果沿着请求调用反向的路径,返回给Secondary OS。这样一次文件访问请求就成功地完成了。这是一个典型的前后端程序,从功能角度看,通信渠道和消息通知机制是最重要的两个方面,文件操作是经常发生的,且对文件操作的性能以及程序性能有着非常大的影响,因此POPFUSE的性能和隔离性也是非常重要的方面。

3.2 POPFUSE的实现

3.2.1 通信渠道

使用在Popcorn的Secondary OS与Primary OS之间建立共享内存的方法进行内核间的通信。因为在Popcorn上建立共享内存的机制非常的便利,并且使用共享内存有着很高的通信性能,所以选择共享内存的方式作为通信的渠道。又因为内存在整个系统中是非常宝贵的资源之一,在本文设计中,尽量减小申请共享内存用于通信,将更多的内存留作他用。在实际过程中设计使用128 KB大小的内存用于进行通信的共享内存。之所以设计为128 KB大小,是因为在FUSE中read、write操作能够请求的最大存储块的大小为128 KB,这项标准是Linux内核规定的,如果申请128 KB以上的内存块,那么该请求会被拆分成多个128 KB的请求分别发送。这样不但浪费内存资源,而且效率也会下降。并且其他的文件操作例如:getattr、readdir、link、rename、flush、truncate等等,它们请求的内存块大小都不会达到128 KB。因此采用共享内存的方式进行内核间的通信,共享内存是在Primary OS后端服务程序启动时建立的。

3.2.2 通知机制

在系统实现的初期,选择了中断来通知文件操作请求的到来。但是由于大数据量的读写请求会被拆分为多个小的请求,导致请求在某个时间段内发生的非常频繁,从而产生很长的中断处理队列,进而导致系统性能严重的下降。于是又采用轮询的方式来处理请求的到来。但是系统性能还是没有明显的提升,反而有所下降。通过大量的前期实验分析,最终发现文件访问存在访问密集周期和访问空闲周期的特点。因为通常情况下,文件访问在一个时间点上是相对集中的,请求在一个时间段内集中的产生,经过一段时间的密集访问又会回到访问空闲期。因此在这种情况下,如果采用中断机制会导致在文件访问密集时,中断频繁的产生,导致系统性能的下降。如果采用轮询机制,又会导致在文件访问空闲时轮询是空转的,系统如果不停的使用轮询又非常消耗处理器资源,会对系统的性能造成更大的影响,尤其是在Primary OS需要为多个Secondary OS服务的

情况下。因此最终选择同时分段使用轮询和中断作为消息通知机制。Primary OS端服务程序启动后会进入睡眠状态,先通过中断来等待Secondary OS请求的到来。此时如果Secondary OS端有请求产生,便会向Primary OS发送中断唤醒睡眠的服务线程来进行响应,在此后的一段时间内服务线程不停的轮询共享内存查看是否有请求到达,若一段时间内没有请求到达时,则再次进入睡眠状态等待唤醒。在系统的实现过程中,预留了共享内存开头的24个字节分别用于存放共享内存的锁和两个标志位,通过预留的这两个标志位,分别标识了是否需要发送中断和共享内存的使用状态。每次对共享内存进行操作后都需要修改共享内存的状态标志位;如果发现该标志位被修改了,说明有请求到达或者操作已经完成,可以进行下一步操作。

3.2.3 性能方面

由于对文件系统的实际操作过程中可能同时会有多个请求到达,如果只使用一块共享内存进行操作,势必会使请求排队,增加延迟。因此设计使用多个线程并且使用多个共享内存来同时处理请求,即创建多个工作线程和共享内存,线程和共享内存一一对应起来,将线程与共享内存进行绑定,每个线程接收特定的共享内存传递的请求进行处理,中断到来后根据共享内存ID唤醒相应守护线程。这样,就实现了整个文件系统的并发执行,全面提高了系统的性能。

3.2.4 性能隔离

在多个Secondary OS需要使用POPFUSE时,Primary OS端启动多个POPFUSE后端服务程序分别为对应的Secondary OS服务,并使用CGroup机制对多个后端服务程序进行隔离,CGroup提供一种可以限制、记录、隔离进程组(process groups)所使用的物理资源(如:CPU、memory、IO等等)的机制。CGroup为每一个Secondary OS维护一个进程组,Primary OS端的POPFUSE服务程序在启动时会自动获取本身进程ID和通讯的Secondary OS ID,将两者相互对应并写入到指定的文件中供用户观察和使用,Secondary OS启动完成后会根据该文件中的内容将每一个Secondary OS的POPFUSE后端进程添加到相应的CGroup组中。POPFUSE后端进程结束时也会自动更新该文件,将文件中对应的条目删除,从而保证了虚拟化环境下对资源的隔离。

4 实验评估

4.1 测试环境

存储设备为SATA磁盘,CPU:Intel Xeon E7-8870 @ 2.40 GHz,共计48个核,Cache size:30 720 KB,内存为1 058 679 444 KB,Secondary OS实际分配4 GB,页大小为4 096 Byte,存储设备大小为1 TB,测试软件为benchmark工具iozone。

4.2 测试目的与方法

测试过程中,选用了现有的文件系统NFS来作为性能和功能方面的比较对象。之所以选用NFS来同POPFUSE进行比较,是因为,NFS不仅仅是现有的能够进行多点访问的分布式文件系统,而且它还是开发选择文件系统时的备选方案。因此选择NFS来同POPFUSE文件系统来做性能和功能方面的比较。为了进一步体现系统的性能,还将POPFUSE同传统的Linux文件系统Ext3进行了比较,以便分析出POPFUSE在实现各方面功能时所产生的性能损失。在接下来的实验中使用benchmark工具iozone来对本文文件系统的性能进行测试。通过在多线程下对write、re-write、read、re-read、random read、random write来比较POPFUSE文件系统与NFS文件系统的性能,以及POPFUSE与传统Linux文件系统的性能。之所以设计为128 KB块大小,是因为在FUSE中read、write操作能够请求的最大存储块的大小为128 KB,如果申请128 KB以上的内存块,那么该请求会被拆分成多个128 KB的请求分别发送。

write:用来测试一个新创建的文件被写入数据时的性能。当对一个新创建的文件进行写入时,既要存储文件本身的内容,还要记录数据存储在磁盘中的具体位置,这些额外信息也要被保存下来。它们被称作“元数据”。这些元数据中所包含的内容有:目录信息、所分配的空间以及一些与该文件有关的数据。由于要处理这些额外的信息,因此write的性能通常会比re-write的性能低。

re-write:用来测试向一个已经被创建的文件中写入数据时的性能。当向一个已经被创建的文件中写入数据时,需要的工作量要小的多,因为这时的元数据已经不需要存储了,它已经存在。因此re-write的性能通常比write的性能高。

read:用来测试从一个已经被创建的文件读数据时的性能。

re-read:用来测试从一个最近被读过的文件再次读数据时的性能。re-read在这个过程中所体现的性能要比第一次读该文件要高,这是因为现代的操作系统一般会将最近读过的文件数据进行缓存处理。这个缓存可以被用于读以提高性能。

random read:用来测试从一个文件中通过随机偏移量来读取数据时的性能。操作系统缓存的大小,拥有磁盘数量的多少,寻道的延迟以及其他一些因素都有可能影系统的性能。

random write:用来测试向一个文件中通过随机偏移量来写入数据时的性能。同测试随机读一样,在测试随机写时,操作系统缓存的大小,磁盘数量,寻道的延迟以及其他一些因素都有可能会影响系统性能。

通过对NFS系统和POPFUSE系统进行大文件的拷贝,从而测量其时间的消耗,凸显系统性能的优劣。

4.3 测试结果及分析

通过标准测试软件iozone的反复测试,取得每次测试时的平均值作为最终的数据后,形成了若干的反映各方面性能的数据图。挑选了具有代表性的几幅来说明性能方面的问题。

图3可以反映出,对于大小为1 GB的文件进行测试时,读、连续读、随机读以及写、连续写POPFUSE的性能比NFS只提高了20%,甚至POPFUSE随机写时的性能会略低于NFS。这是因为,文件比较小时,系统的Cache发挥了作用,使得读写的延迟不是太明显。

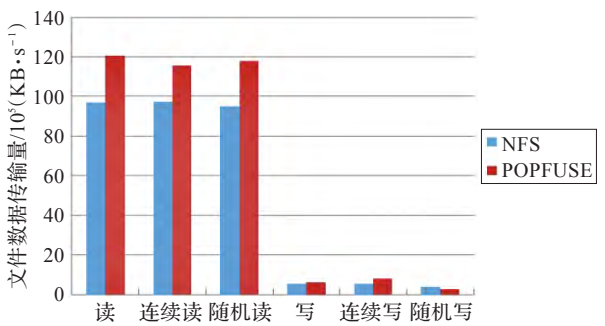

图3　6线程(块儿大小128 KB,文件大小1 GB)

从图4和图5中可以看出,文件增大到8 GB和16 GB时POPFUSE的读、连续读、随机读、写、连续写性能明显要好于NFS的读写性能,只有随机写性能不是太突出。这是因为NFS使用的是高缓存机制,这使得随机读在缓存中读取的概率较高,所以优势不是太明显。

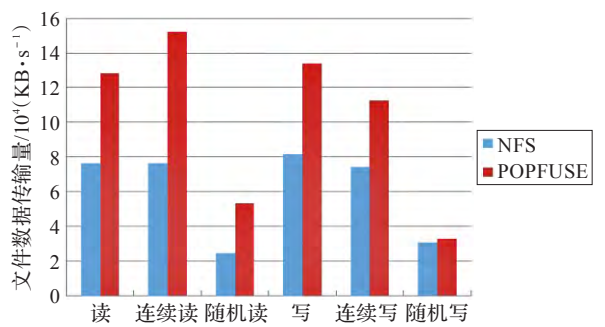

图4　6线程(块儿大小128 KB,文件大小8 GB)

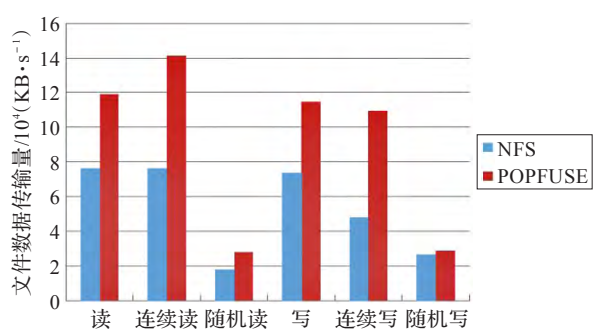

图5　6线程(块儿大小128 KB,文件大小16 GB)

图6、图7、图8是在一次写入/读出并且块大小为4 MB时所测取的数据,它反映的是在块儿大小为4 MB的情况下,不同的的文件大小各种读写方面的性能。同写入/读出的块大小为128 KB时所反映的性能一样,POPFUSE的读、连续读、随机读、写、连续写性能明显要好于NFS的读写性能。随着测试文件的增大,POPFUSE文件系统的性能相对于NFS文件系统的性能优势更加凸显,各方面的性能均超过了NFS性能的1倍。

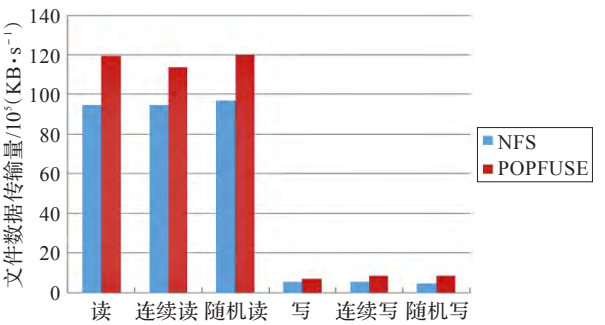

图6　6线程(块儿大小4 MB,文件大小1 GB)

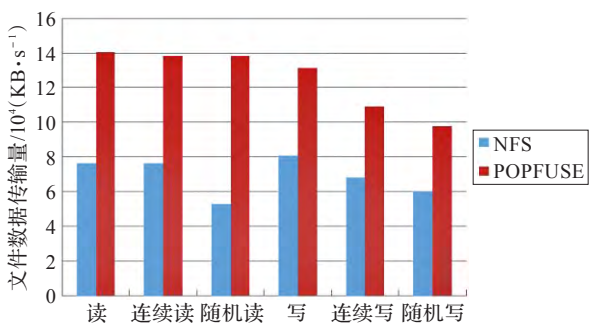

图7　6线程(块儿大小4 MB,文件大小8 GB)

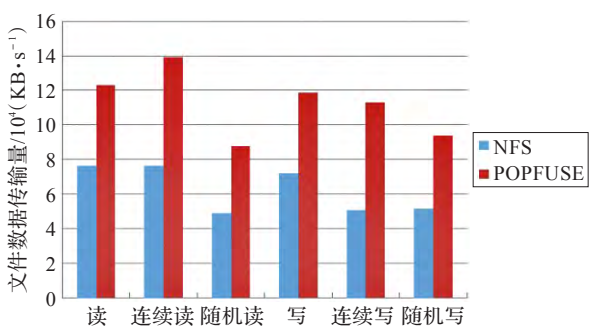

图8　6线程(块儿大小为4 MB,文件大小16 GB)

图9~图12是将POPFUSE与传统的Linux文件系统比较所得到的数据图。图中反应的依然是在多线程,不同的块大小、文件大小的情况下,进行文件的读、写、连续读、连续写、随机读、随机写时的性能。与之前反映的情况有所不同,下面几幅图反映的是POPFUSE相对于传统的Linux文件系统作比较时性能方面的损失。之所以产生性能损失是因为POPFUSE在实现文件访问功能时,增加了中间模块的调用环节,POPFUSE文件系统在实现每个操作时,处理路径变长,从而产生了一定的性能损失。从图中看到POPFUSE相对于传统的Linux文件

系统性能都有不同程度的降低，但是在实际使用过程中由于系统中的Cache也会发挥一定的作用，这些性能的损失会进一步被削弱，所以它在可控和合理的接受范围，并且它相对于前面提到的NFS的性能的确是有很大的提升。所以，有理由相信POPFUSE能够作为由多个内核所组成的操作系统的文件系统。

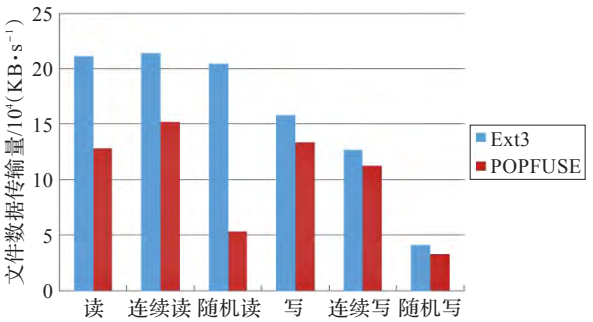

图9　6线程(块儿大小128 KB，文件大小8 GB)

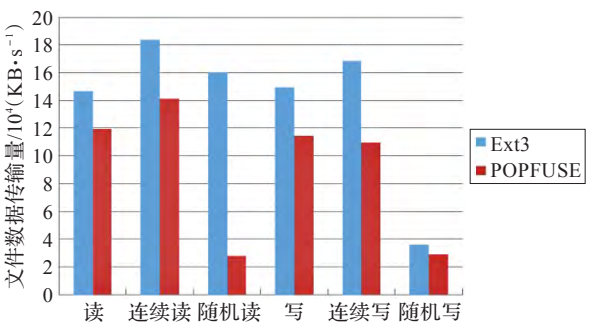

图10　6线程(块儿大小128 KB，文件大小16 GB)

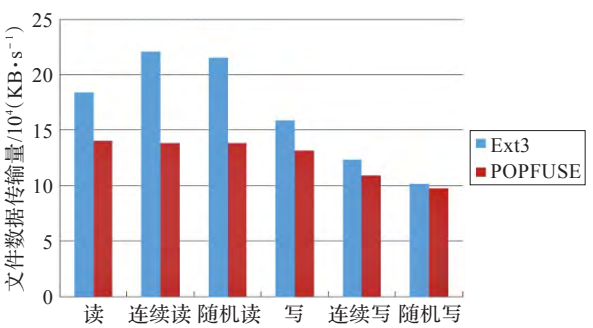

图11　6线程(块儿大小4 MB，文件大小8 GB)

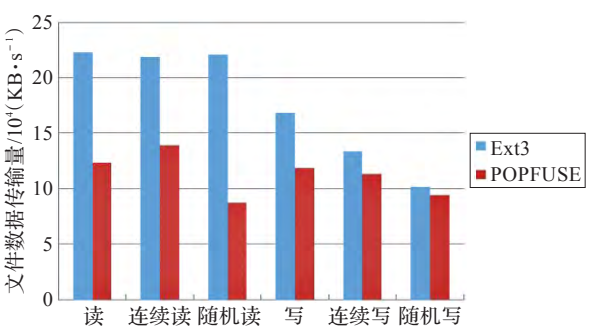

图12　6线程(块儿大小4 MB，文件大小16 GB)

图13、图14、图15是对NFS系统和POPFUSE系统分别进行6 GB大文件拷贝测试时产生的数据图。三幅图集中反映的是在用户态、内核态以及实际使用时三种情况下，大文件拷贝时的时间消耗。如图反应出在三种情况下，POPFUSE文件系统的耗时都要比NFS文件系统的耗时低，从而进一步反应出POPFUSE的性能要优于NFS。

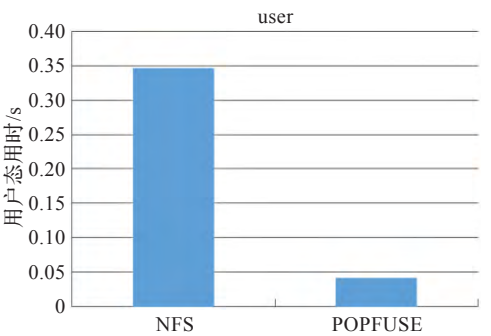

图13　拷贝6 GB大文件用户态用时

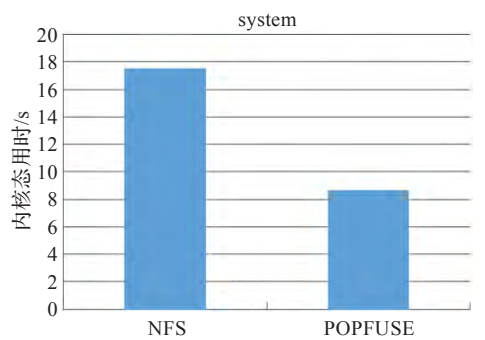

图14　拷贝6 GB大文件内核态用时

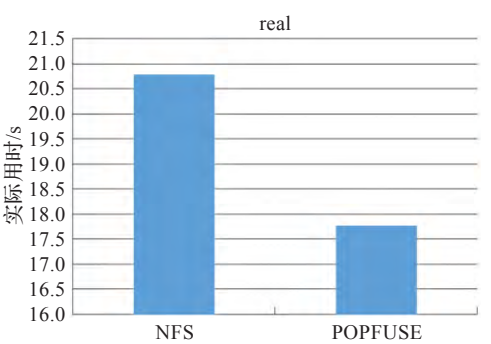

图15　拷贝6 GB大文件实际用时

5　结束语

因为目前现存的文件系统都是针对单内核开发的，没有针对多个内核所组成的操作系统开发过文件系统。传统的文件系统无法在多个内核的操作系统上直接使用，因此本文设计并开发出了适用于多个内核的操作系统的用户态文件系统POPFUSE，并且通过反复的测试证明能够在多个内核的操作系统上稳定工作。性能方面，POPFUSE相对于同类型的NFS文件系统提升了一倍，特别是在多线程，大文件量读取的情况下，性能提升的更为明显。POPFUSE在使用过程中可以动态加入或删除，具有更好的灵活性和扩展性。

（下转85页）

[3] Malewicz G,Austern M H,Bik A J C,et al.Pregel:A system for large-scale graph processing[C]//Proceedings of SIGMOD 2010,2010:135-146.
[4] Welcome to apache giraph![EB/OL].(2014-06-11)[2015-04-30].http://giraph.apache.org/.
[5] 袁培森,舒欣,沙朝锋,等.基于内存计算的大规模图数据管理研究[J].华东师范大学学报:自然科学版,2014(5):55-71.
[6] Mondal J,Deshpande A.Managing large dynamic graphs efficiently[C]//Proc of 2012 ACM SIGMOD Int Conf on Management of Data.New York:ACM,2012:145-156.
[7] METIS[EB/OL].(2014-06-11)[2015-04-30].http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.
[8] Karyphis G,Schloegl K,Kumar V.ParMETIS:Parallel graph partitioning and sparse matrix ordering library[D].University of Minnesota,2003.
[9] Wang L,Xiao Y,Shao B,et al.How to partition a billion-node graph[C]//Proceedings of IEEE International Conference on Data Engineering,2014:568-579.
[10] Fiduccia C M,Mattheyses R M.A linear-time heuristic for improving network partitions//Proceedins of ACM IEEE 19th Design Automation Conference Proceedings,Las Vegas,USA,1982:174-181.
[11] Kernighan B W,Lin S.An efficient heuristic procedure for partitioning graphs[J].Bell Systems Journal,1970,49(2):291-308.
[12] Hendrickson B,Leland R.The Chaco user's guide:Version 2.0,Technical Report 1994-2692[R].New Mexico:Sandia National Laboratories,1994.
[13] Pellegrini F,Roman J.Scotch:A software package for static mapping by dual recursive bipartitioning of process and architecture graphs[J].High-Performance Computing and Networking,1996,1067:493-498.
[14] Chevalier C,Pellegrini F.PT-Scotch:A tool for efficient parallel graph ordering[J].Parallel Computing,2008,34(6/8):318-331.
[15] Zhu Xiaojin,Ghahramani Z.Learning from labeled and unlabeled data with label propagation,CMU-CALD-02-107[R].Pittsburghers:Carnegie Mellon University,2002.
[16] 张俊丽,常艳丽,师文.标签传播算法理论及其应用研究综述[J].计算机应用研究,2013,30(1):21-25.
[17] Symoen P,Yiannis K,Athena V,et al.Community detection in social media performance and application consederations[J].Data Mining and Knowledge Discovery,2012,24(3):515-554.
[18] SNAP:Network datasets:Wikipedia talk graph till January 2008[EB/OL].(2014-06-11)[2015-04-30].http://snap.stanford.edu/data/wiki-Talk.html.
[19] SNAP:Network datasets:Orkut online social network[EB/OL].(2014-06-11)[2015-04-30].http://snap.stanford.edu/data/com-Orkut.html.
[20] SNAP:Network datasets:LiveJournal social network[EB/OL].(2014-06-12)[2015-04-30].http://snap.stanford.edu/data/soc-LiveJournal1.html.
[21] SNAP:Network datasets:Friendster online social network[EB/OL].(2014-06-12)[2015-04-30].http://snap.stanford.edu/data/com-Friendster.html.

(上接49页)

参考文献:

[1] 王浩亮.模拟Unix文件系统的设计与实现[J].电脑开发与应用,2009(1):200-205.
[2] 谷方舟.云计算环境中分布式文件系统的负载均衡问题研究[D].北京:北京交通大学,2012.
[3] 吴一民,刘伟安.基于FUSE的用户态文件系统的设计[J].微计算机信息,2010,26(6):159-160.
[4] Szeredi M.File system in user space[EB/OL].[2015-10-30].http://fuse.sourceforge.net.
[5] 段翰聪,王勇涛,李林.EDFUSE:一个基于异步事件驱动的FUSE级文件系统框架[J].计算机科学,2012(6):56-58.
[6] FUSE.Filesystem in userspace[EB/OL].(2010-03-02/2010-06-19)[2015-10-30].http://fuse.sourceforge.net/.
[7] Zhao T Z,Dong S B,Verdi M,et al.Performance evaluation and relative predictive model of parallel file system[J].Journal of Software,2011,22(9):2206-2221.
[8] 扶小飞,郑善贤.一种Flash文件系统的设计和实现[J].微计算机信息,2010(5):34-36.
[9] Singh A.MacFUSE[EB/OL].[2015-10-30].http://code.google.com/p/macfuse.
[10] FUSE on OpenSolaris[EB/OL].[2015-10-30].http://opensolaris.org/os/project/fuse.
[11] Henk C.FreeBSD FUSE[EB/OL].[2015-10-30].http://fuse4bsd.creo.hu.
[12] Kantee A,Crooks A.Refuse:Userspace fuse reimplementation using puffs[C]//Proc of the 6th European BSD Conference (EuroBSDCon),2007.
[13] Shelton B H.Popcorn Linux:Enabling efficient inter-core communication in a Linux-based multikernel operating system[D].Virginia Polytechnic Institute and State University,2013.
[14] 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.
[15] Machek P.UserVFS[EB/OL].[2015-10-30].http://sourceforge.net/projects/uservfs.
[16] Rajgarhia A,Gehani A.Performance and extension of user space file systems[C]//Proceedings of the 2010 ACM Symposium on Applied Computing,2010:206-213.
[17] Sundararaman S,Subramanian S,Rajimwale A,et al.Membrane:Operating system support for restartable file systems[J].ACM Transactions on Storage (TOS),2010,6(3):11-12.