基于给定访问序列的NFS预取技术

骆志军[1,2] 许建卫[3] 郑 规[3] 刘新春[3] 邵宗有[3] 聂 华[3]
(中国科学院计算技术研究所国家智能计算机研究开发中心 北京 100190)[1]
(中国科学院研究生院 北京 100190)[2] (国家高性能计算机工程技术研究中心 北京 100089)[3]

摘 要 网络文件系统(Network File System)由于其稳定、易用而得到了广泛的应用。为了提高NFS服务器端的I/O性能,提出一种在给定客户端访问序列的NFS服务器端伪随机序列预取(NPRP:NFS Pseudo Random Prefetch)机制。实验测试结果表明,NPRP机制对特定应用的NFS服务器的I/O带宽提升超过2倍。

关键词 NFS,预取,随机读,I/O加速
中图法分类号 TP302.1 **文献标识码** A

NFS Prefetch Mechanism Based on Given I/O Pattern

LUO Zhi-jun[1,2] XU Jian-wei[3] ZHENG Gui[3] LIU Xin-chun[3] SHAO Zong-you[3] NIE Hua[3]
(NCIC, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)[1]
(Graduate School of the Chinese Academy of Sciences, Beijing 100190, China)[2]
(National High Performance Computer Engineer Technology Center, Beijing 100089, China)[3]

Abstract Network File System (NFS) has been widely used in many domains for its stability and ease of use. To improve the I/O performance of NFS server, we presented a NFS Pseudo Random Prefetch (NPRP) mechanism based on given I/O pattern. Experimental results show that more than 200% I/O performance of NFS server with NPRP mechanism has increased for applications with given patterns.

Keywords NFS, Prefetch, Random read, I/O speed up

1 引言

目前,集群系统逐渐成为HPC应用的主流平台,在全球最新发布的TOP 500[1]系统中,集群系统已经占到84%以上。大多数的集群存储系统都配置成服务器—客户端的模式,即I/O节点作为服务器,计算节点作为客户端。计算过程中所需的大量数据都存储在远程的I/O服务器上,通过高速网络来访问。这种网络存储模式由于具有集中存储、易于管理、方便部署、强可扩展性等优点,其应用已迅速推广至各行各业。

网络文件系统(NFS:Network File System)由于其稳定性、易用性以及开源免费的特性,往往成为不少集群网络存储的首选系统。虽然现在还有不少其它的网络存储产品,但要么其稳定性、易用性还不能与NFS比拟;要么产品价格昂贵,不适用于中小规模集群网络系统。所以对于大部分集群系统,NFS还是不可替代的。

随着NFS架构的日趋成熟和完善,NFS已经深入各种应用领域,比如高性能计算、IPTV(Internet Protocol Television)、遥感信息处理、石油勘探等。但随着处理器计算速度的飞速发展,NFS所提供的I/O带宽已经逐渐成为集群系统的瓶颈。

针对NFS文件系统的I/O性能问题,各种研究和改进也取得了有效的进展。特别是对各种I/O访问模式比较有规律的应用,比如IPTV应用,其访问模式往往具有顺序性、可预测性等特性,通过提取这些应用的访问模式,采取缓存、预取或I/O并行化等技术,可有效提高NFS的I/O性能。

然而,对于具有随机访问模式的各种应用,可以提升NFS服务器的I/O性能的相关研究非常少。比如石油勘探,这些应用的计算量和数据量很大,而且数据操作主要是随机读,不过随机读的访问序列可以提前得到。本文针对这种随机访问模式的应用,提出并实现了一种NFS服务器端伪随机序列预取(NPRP:NFS Pseudo Random Prefetch)机制:

- 在客户端给定随机访问序列的前提下,通过在NFS服务器端预取来缓解I/O压力。
- 在服务器端增加小容量高速存储介质,用来缓解突发性的I/O峰值。
- 在服务器端对所有客户端的预读请求进行局部重排序,从而使得磁盘的访问尽可能连续,提高磁盘系统的吞吐率。

NPRP机制具有如下特点:

- 透明性:NPRP由于是加载在NFS服务器端的内核模块,不需要对客户端的应用、内核做任何的改动,因此对应用具有透明性。
- 对Linux内核改动小:NPRP主要是对NFS内核模块插入了一个函数用于截获NFS客户端的I/O请求。除此之外,其它模块都是可以动态加入和卸载的独立模块,对内核的

到稿日期:2010-09-23 返修日期:2010-12-30
骆志军(1983—),男,硕士生,主要研究方向为计算机系统结构,E-mail:lzhijun90@gmail.com。

改动非常小。

• 可加载额外的高速存储设备：NPRP 可以利用主机内存 ramdisk 作为预取数据的存储介质，也可以加载额外的高速存储设备，如 MCard[8]。增加额外的高速存储设备，相当于拓展了 I/O 服务器的内存，使得 I/O 服务器具有更强的缓存性能。

实验测试结果表明，NFS 服务器端伪随机序列预取机制可提高特定应用的 NFS 服务器端的 I/O 性能 2 倍以上。

2 NFS 相关研究

I/O 的并行化以及预取技术一直以来都是改进存储设备 I/O 性能的重要策略。

NFS 由于不能分布式存储数据，不适合部署超大型集群系统，IETF 工程应用组 NFS v4 小组委员会提出了 pNFS[2,4] (Parallel NFS) 协议，专注于 NFS 的并行化优化。由于 NFS 的单服务器结构，使其在扩展性方面存在缺陷，随着客户端的增多，其性能不能随之提升。pNFS 的设计有效地解决了这个问题，它首先分离数据流和控制流，使得数据可以条带化存取到不同的 NFS 服务器，从而改变了 NFS 传统的单服务器结构，实现了在客户端和多个 I/O 服务器之间直接并行传送数据，将传输速率提高了几个数量级。

针对 NFS 服务端和客户端的负载均衡问题，Batsakis 提出了 CA-NFS[3]。每个 NFS 客户端都会消耗 NFS 服务器端一定的资源，包括 CPU、网络以及内存等。随着客户端的增加，服务器端将因为资源消耗殆尽而导致拥塞。Batsakis 通过对 NFS 客户端做负载均衡，解决了这个问题。Batsakis 为 NFS 增加了一个负载评价系统，把 NFS 服务器端和所有的客户端都看成一个整体。当系统资源到达了极限的时候，负载评价系统应用优先级策略，优先考虑阻塞的请求，这样不那么重要的异步 I/O 操作将不会妨碍即时的同步请求，使得 NFS 服务器端和客户端可以互相感知彼此的资源使用状态，从而更有效地利用整个系统的资源，使系统整体的负载更加均衡，可提升 NFS 的性能 20%以上。

预取是提升 I/O 性能的重要策略。对于计算密集及 I/O 密集的应用，通过预取，重叠计算时间和 I/O 时间，可以降低 I/O 延迟，提高性能。当前 NFS 的预取策略实际上就是 Linux 内核的预取策略，这种预取策略足够通用，但对于很多应用的 I/O 访问模式还是不能有很好的表现，特别是随机访问模式。Patterson[5] 提出由用户程序指定一些访问模式，根据这些模式，去预取和缓存数据；Surendra[6] 通过收集用户进程的 I/O 访问模式，更加有效、精确地预取和缓存数据；Yong Chen[10] 则“复制”一个与应用程序类似的后台进程，它只做和原应用程序相同的 I/O，但跳过所有的计算，于是这个后台进程可以先于应用程序取得数据，降低了 I/O 的延迟。这 3 种方法都收集了应用程序的额外信息，通过这些信息做更有效的预取。但是它们都需要应用程序的支持，或需要直接修改应用程序，不能做到应用的透明性。

以上方案都是针对特定应用对 I/O 的加速，但都不能提高随机访问的 I/O 服务器性能，不能充分利用现有服务器的 I/O 带宽。在网络没有瓶颈的情况下，一个 I/O 服务器的输出最终体现在磁盘的速度上，而由于磁盘的物理特性，顺序访问的磁盘带宽要远大于随机访问。因此，本研究提出预取和 I/O 重排序的方法，以提升磁盘的性能，从而提升 NFS 服务器的 I/O 性能。

3 NPRP 的设计和实现

各个客户端的访问序列是已知的，客户端将其未来的访问序列以索引文件的方式提供给服务器端。服务器端采取 NPRP 机制提升 I/O 性能。

一种简便的方法是在客户端或服务器端直接按照索引文件来预取，但这种预取实际上只是把计算时间和 I/O 时间重叠，降低了 I/O 的延迟，而 I/O 服务器端的随机访问特性并没有改变。我们对给定的访问序列进行局部重排序，将随机的访问序列进行局部的顺序化，从根本上提高服务器端磁盘的 I/O 带宽，从而提升 NFS 整体的 I/O 性能。

NPRP 机制分为两个模块，如图 1 所示。

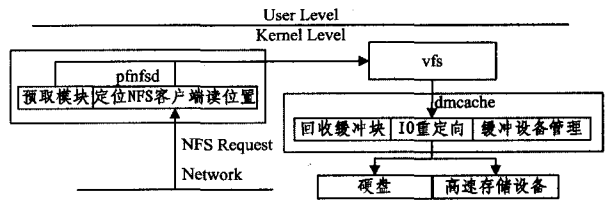

图 1 NPRP 整体框架

图 1 中，NPRP 主要由 pfnfsd (prefetch network file system daemon) 和 dmcache (device mapper cache) 两个内核模块构成。pfnfsd 是对内核模块 nfsd 的改写，主要包括 NFS 客户端请求定位模块和预取模块，用来截获和定位客户端的 NFS 请求，并执行预取等最主要的工作；dmcache 是基于 Linux 内核 Device Mapper[7] 机制实现的一个 target driver，主要包括高速存储设备的管理、地址映射和资源回收模块，它把预取的数据放到高性能存储设备上，如 ramdisk、高性能存储卡 MCard 等。

下面分别详细介绍 pfnfsd 模块和 dmcache 模块的设计和实现。

3.1 pfnfsd 模块的设计和实现

3.1.1 定位 NFS 读位置

pfnfsd 首先获得各客户端的索引文件，在内存中为每个客户端建立一个索引表，并为每个索引表维护一个指针，指向最后一次实际请求所对应的索引位置。

之后，pfnfsd 不断获得 NFS 客户端的读请求，根据这个请求和索引表，可以确定客户端当前请求在索引表中的位置。简化的流程图如图 2 所示。

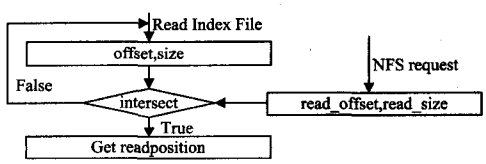

图 2 定位 NFS 读位置

只要 NFS 客户端的读请求和索引表的某个序列有交集，我们就认为客户端已经读到了索引文件的某个请求。假设这个和客户端有交集的请求在索引表中是第 N 个请求。因为客户端读请求是按照索引表的请求顺序读下来的，所以当前读位置可以用当前请求在索引表的位置来表示，即为 N。

虽然通过给定的访问序列文件(索引文件)可以精确地知道 NFS 客户端访问的文件偏移，但是，这个索引文件是用户态的文件偏移，实际上文件访问的偏移在经过 NFS 客户端内

核的 VFS 层之后，会有一些变化。另外，客户端的请求也有可能在 VFS 层被文件系统缓存命中或部分命中，从而导致整个请求不会传递或部分传递到 NFS 服务器端。至于客户端的文件系统的预取，由于访问模式的随机性，内核的预取阈值基本不可能打开，因此不大可能有预取的请求到 NFS 服务器端。针对这些细节问题，定位 NFS 读位置，不仅仅是 NFS 请求和索引表匹配的过程，还必须有模糊匹配错误纠正功能。

模糊匹配错误纠正是指将一个 NFS 客户端的请求与索引表的匹配分为完全匹配和不完全匹配，并分别进行处理。如果是完全匹配，即请求的读区间和索引文件完全重合，就认为当前获取的是一个“绝对准确的位置”；否则，即是不完全匹配。不完全匹配所得到的位置有可能是错误的，因此不能只找到一个不完全匹配就停止，必须在前一个绝对准确位置开始，寻找一定的区间，找出所有的不完全匹配。然后根据前后关系，剔除一些不合理的值，比如前一次的不完全匹配必定在当前不完全匹配之前，如果最终确定的不完全匹配只有一个，则这个值有可能是一个“绝对准确的位置”，否则只是用作后面的不完全匹配剔除不完全匹配数据的一些参考位置。

确定客户端的读位置，一方面，可以和预取线程同步，防止预取线程落后于客户端的应用程序；另一方面，可以确定 dmcache 模块的哪些缓存块可以被回收，这个问题在回收缓存块模块中会有详细描述。

3.1.2 I/O 重排序

预取模块在内核使用一个内核线程根据索引表进行预取，称之为预取线程。之所以使用一个线程而不是多个线程，是因为经过我们的测试发现：对同一个文件，单线程的读性能往往比多线程同时读一个文件的多个不同部分更快，故使用单线程。

为了提高磁盘访问的顺序性，进行了 I/O 重排序。I/O 重排序首先有一个排序窗口，窗口大小以其中所含请求对应的 I/O 容量来衡量。比如对索引表的每 100MB 容量的请求进行排序，排序窗口即为 100MB。在预取模块之外事先对索引表按照这个排序窗口排序，预取模块直接读这个排好序的索引表来预取，或在预取模块中先排序再预取。预取的时候必须查看客户端的读请求位置，如果预取的位置大于客户端的读请求位置，则可以预取，否则，必须让客户端等待。

预取的数据不是由当前模块保存在内存中，而是由 dmcache 模块保存在一个高速缓存设备中，这个设备可以是 ramdisk、MCard[8] 等。

由于预取数据时是根据重排序后的索引文件进行预取的，和 NFS 客户端的读操作的实际数据不一样，这就要求一个排序窗口的所有数据都预取完成之后，才能由 NFS 客户端去读，否则只能让客户端等待。如果在一个排序窗口之内预取线程和 NFS 客户端同时操作数据，那么 NFS 客户端所请求的数据有可能是预取线程还没有预取的数据，从而打乱了预取线程对磁盘的访问，I/O 重排序之后对磁盘的顺序化访问所带来的磁盘带宽提升就可能被抵消。

排序窗口越大，对磁盘的访问顺序性就越强。如果排序窗口为文件大小，则随机读变为顺序读，但对缓存预取数据的高速存储设备的容量要求也越大，客户端的请求也越有可能产生饥饿。

3.2 dmcache 模块设计和实现

dmcache 模块主要对预取的数据进行缓存。它实现为 Device Mapper 的一个 target driver。

Device Mapper 是 Linux 2.6 内核中提供的一种从逻辑设备到物理设备的映射框架机制，它通过模块化的 target driver 插件实现对 I/O 请求重新定向等。借助于 Device Mapper 机制，我们把低速的存储设备（如硬盘）和高速的存储设备（如 ramdisk，MCard 等）映射为一个逻辑的块设备。

3.2.1 dmcache 模块框架

dmcache 模块的流程如图 3 所示。

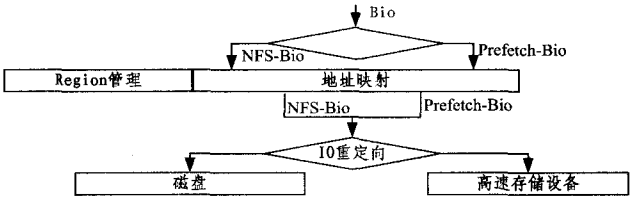

图 3 dmcache 框架

dmcache 模块是在 Linux 文件系统层之下、设备驱动层之上，故在图 3 看到的请求是 bio。

在图 3 中，dmcache 模块把磁盘和高速存储设备映射为一个块设备，其中高速存储设备用来存储预取的数据，其容量是对外隐藏的，相当于磁盘的一个高速缓存。

dmcache 一方面需要把映射出来的逻辑设备的地址映射到磁盘或高速存储设备，这主要是地址映射和 I/O 重定向模块的功能；另一方面，需要管理高速存储设备，这是缓存块（region）的管理模块的功能。

3.2.2 地址映射机制和 I/O 重定向

从文件系统层下来的 bio 请求有两类：

一种是预取线程下来的预取请求。如果这个请求的数据没有在高速存储设备中，则从硬盘读数据并拷贝到高速存储设备，否则直接从高速存储设备读数据返回；

另一种请求是从 NFS 客户端过来的请求。如果这个请求在高速存储设备，则从高速存储设备中取数据，否则直接到硬盘取数据。

对这两种不同的 bio，都必须经过哈希查询、I/O 重定向才能到达真正的物理设备。

高速存储设备被划分为以 region 为单位的缓存块来管理，region 是负责从磁盘的地址空间映射到高速存储设备 cache 的地址空间的一个数据结构。所以，在查找每个 bio 读取的数据是否在高速存储设备中时，只需要找到这个数据在磁盘的地址对应的 region 即可。

为了从磁盘的地址空间找到对应的 region，我们使用了一个哈希表来满足这个映射。所有填充了磁盘数据的 region 都会在这个哈希表中，还没有填充数据的 region 则放在一个链表中。所以在图 3 中，判断 bio 是否被预取，实际上是用其对应的磁盘地址查询这个哈希表。找到 region 则表示数据已经被预取了，而通过这个 region 可以得到这个数据在高速存储设备的地址，从而得到数据。

利用找到的 region，对 bio 的目标设备和设备内偏移进行改变，这实际上就是 I/O 重定向。

3.2.3 region 管理

就像操作系统的 page 需要回收一样，空闲的 region 被用完之后，需要替换回收。

由于高速存储设备保存的是预取的数据，如果预取的数据和 NFS 客户端读取的数据顺序是一样的，那么在 NFS 客

户端命中之后即可回收这个 region。但是，预取线程是对索引文件进行了 I/O 重排序的，预取数据的读取顺序和 NFS 客户端实际读取数据的顺序是不一样的，所以不能简单地使用“命中就回收”的方法。而传统的 LRU 之类的算法也不合适，因为 NFS 客户端的读取完全是随机离散的，现在命中的数据往往都不会被再次读取。

我们采用适合于上述特征的 region 回收算法：NFS 客户端每读完一个 I/O 重排序窗口的数据，就回收一个窗口的 region。这种算法比较有效，因为 NFS 客户端请求序列和预取线程请求序列在每个 I/O 重排序的开始和结束时，总有一个“交点”。如果 NFS 客户端读数据越过了一个窗口，则前一个窗口所有的 region 肯定都可以回收了。

4 性能测试和分析

4.1 测试环境

硬件环境采用曙光公司的两台 A620r-H 服务器，该服务器 CPU 型号为 Quad-Core AMD Opteron(tm) Processor 2378，双路四核 CPU，主频为 2.4GHz，内存为 16GB；硬盘采用 SAS 硬盘，150GB 大小；网卡是 Intel Corporation 82576 千兆网卡；操作系统为 Redhat Enterprise 5.1，内核版本是 linux-2.6.18。一台作为 NFS 服务器，另一台作为 NFS 客户端，NFS 版本采用默认的 NFSv3，所有设置都是默认。

测试时采用从中国石油东方地球物理公司[9]获取的数据处理索引文件，并编写了模拟石油勘探应用的测试程序，其主要工作是按照索引文件给定的索引，读取从 NFS 服务器端挂载过来的数据文件，并且每执行一次 I/O 就执行一些计算。

4.2 性能测试和分析

NPRP 性能测试对比如图 4 所示，其中：横轴是 I/O 重排序窗口大小，若排序窗口为 0，即为随机读；纵轴是测得的 I/O 带宽。

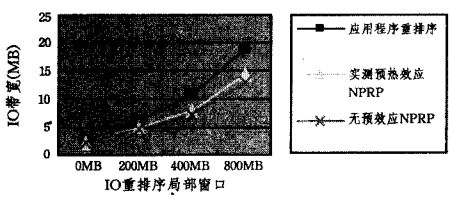

图 4 NPRP 性能测试对比

测试的数据文件都是 20GB 的。应用程序重排序测试时两台测试机内存都设定为 1GB。当测试 NPRP 时，服务器内存提升为 3GB，其中 2GB 分配给 ramdisk 作为 NPRP 的高速存储介质，用于存储预取的数据。

应用程序重排序是指测试程序首先对索引文件做 I/O 重排序，再读数据。排序窗口为 0 即表示完全按照索引文件随机读的测试。

“实测预热效应 NPRP”即 NFS 服务器端的预取，也是同样的排序窗口，不过客户端是无排序的测试程序。测试时，需要预取线程先把高速存储设备填满，这相当于一个预热过程，然后才开始测试。

“无预热效应服务器端 NPRP”不是一个测试值，是为公平起见从“实测预热效应服务器端 NPRP”所测的值去除预热效应之后计算得到的值。去除预热效应的方法如下：

NFS 无预热效应 NPRP 带宽 = NFS 实测预热效应 NPRP 带宽×(20GB−2GB) / 20GB = 0.9×NFS 实测预热效应 NPRP 带宽

所以，需要对比的是“应用程序排序”带宽和“无预热效应 NPRP”的各种排序窗口的带宽。

从图 4 测试结果来看，我们有如下结论：

• I/O 排序确实能提高 I/O 服务器性能。图 4 显示，随着排序窗口增大，I/O 带宽也逐渐增加，当排序窗口大于等于数据文件大小时，随机读退化为顺序读，所以上图的极限性能应该是硬盘的极限性能。

• NPRP 能大幅度提高 I/O 服务器性能。对比“无预热效应 NPRP”和 0MB 排序窗口的“应用程序重排序”的带宽可以发现：随着排序窗口的增加，提升的性能也越大。图中最大提升了约 7 倍的 I/O 速度。

• NPRP 不如在 NFS 客户端重排序的性能。在相同排序窗口对比“无预热效应 NPRP”和“应用程序重排序”的带宽可以发现：在 NFS 客户端重排序有更好的性能。这是因为服务器端的预取线程是按照重排序之后的索引进行预读的，而客户端还是随机读，客户端到达服务器端的请求还是不能全部命中预取线程预取的数据。NFS 客户端重排序之后的性能可以说是服务器端预取的理想极限性能。

结束语 针对一些可以事先给定访问序列的 I/O 密集型应用，本文提出并实现了 NFS 服务器端的预取技术，它在不需修改客户端应用程序的前提下，可以有效地提升 NFS 服务器端的 I/O 性能。

由于预取之后的命中率关系，预取的性能相比理想的性能还有一定的差距，我们准备在命中率以及多客户端的支持方面做相应的改进。

参 考 文 献

[1] Statistics about clusters from top500 supercomputer sites[EB/OL]. http://www.top500.org/stats/list/35/archtype, 2010-06-30

[2] Resources sites for information regarding Parallel NFS and its standardization[EB/OL]. http://www.pnfs.com, 2010-06-30

[3] Batsakis A, Burns R, Kanevsky A, et al. CA-NFS: A Congestion-Aware Network File System [J]. ACM Transactions on Stora-ge, 2009, 5(4): 15-24

[4] Hildebrand D, Honeyman P, Adamson W A. pNFS and Linux: Working Towards a Heterogeneous Future [C]// 8th LCI International Conference on High-Performance Cluster Computing. 2007

[5] Patterson R H, Gibson G A, Ginting E, et al. Informed Prefetching and Caching [C]// Proc. 15th ACM Symposium on Operating Systems Principles. Copper Mountain, CO, Dec. 1995: 79-91

[6] Byna S, Chen Yong, Sun Xian-he, et al. Parallel I/O Prefetching Using MPI File Caching and I/O Signatures [C]// SC08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing. Piscataway, NJ, USA: IEEE Press, 2008: 1-12

[7] Inc. RH. Device-mapper Resource Page [EB/OL]. http://sourceware.org/dm, 2010-06-30

[8] 翟佳，许建卫，谢晖，等. 面向 IO 服务器的高性能存储器的实现与优化[J]. 计算机工程与科学，2009，31(11)

[9] BGP INC. China national petroleum corporation Home page [EB/OL]. http://www.cnpc.com.cn/bgp, 2010-6-30

[10] Chen Yong, Byna S, Sun Xian-he, et al. Hiding I/O Latency with Pre-execution Prefetching for Parallel Applications [C]// SC08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing. Piscataway. NJ: IEEE Press, 2008: 1-10