

Linux 预取算法分析与研究

潘志华, 张 涛

(解放军理工大学 指挥自动化学院, 江苏 南京 210007)

摘 要: Linux 作为一个多任务、分时、通用的开源操作系统, 越来越广泛地应用于各种商业和企业的服务器。为了提高系统的性能, Linux 采用预取技术将应用程序所需的数据提前加载到缓存中, 减少应用程序的 I/O 等待时间。然而由于服务器系统负载的多样化, 导致了预取算法遇到越来越多的挑战。该文主要从分析 Linux-2.6.29-rc2 内核源代码入手, 对 Linux 预取算法的体系结构与内部机制进行了深入分析与研究, 并提出了一些改进预取算法的方法, 对于进一步提高 Linux 系统服务器的性能以及 Linux 的推广与使用具有重要的意义。

关键词: Linux; 预取; 缓存

中图分类号: TP316

文献标识码: A

文章编号: 1673-629X(2009)12-0093-04

Research and Analysis on Algorithm of Prefetching in Linux

PAN Zhi-hua, ZHANG Tao

(Institute of Command and Automation, PLA University of Sci. & Tech., Nanjing 210007, China)

Abstract: Linux is a multitask, timesharing, general open source operating system which is being applied for many servers of business and enterprises more and more. In order to improve system's performance, Linux introduces prefetching technology which can load data what application will access in future into cache so that applications can reduce their waiting time for I/O. However, system's loads vary more and more, so that prefetching algorithms face to more and more challenges. Firstly make an analysis for the source code about prefetching algorithm in Linux-2.6.29-rc2, to analyse and understand the architectures and internal mechanisms of prefetching algorithm, then put forward some improved methods which have much important significances on improving system's performance and spreading system's ranges.

Key words: Linux; prefetching; cache

0 引言

随着微处理器设计和生产工艺的快速发展, 存储系统的访问速度与处理器的运算速度的差距越来越显著^[1]。而预取可在应用程序访问数据前将数据加载到缓存中, 从而可以有效地减少应用程序的 I/O 等待时间, 解决处理器与磁盘之间的速度差距。目前关于预取算法的研究可分为两类: 通知式预取和启发式预取。

通知式预取是根据应用程序的提示来揭示应用程序未来可能要访问的数据。Cao 等^[2]等研究开发了应用程序控制的预取和缓存, 提出一个两层页面替换机制, 允许应用程序控制缓存替换策略, 而内核控制应用程序之间的全局缓存空间分配。Patterson^[3]等提出了通知式预取和缓存, 利用应用层提示来揭示应用程序

的未来文件访问, 并开发了一个 cost-benefit 模型来引导预取决定。Chang^[4]等提出了通过投机执行来揭示应用程序未来可能要访问的数据, 从而达到精确预取的目的。

启发式预取算法主要通过应用程序的历史访问记录来分析和预测应用程序的访问模式, 并作出预取决策。Linux 预取算法就属于这一类范畴。

随着 Linux 越来越广泛地应用于各种商业和事业的服务器, 系统的负载越来越多样化, 导致预取算法遇到了越来越多的挑战。文中的主要目的是分析 Linux-2.6.29-rc2 的预取算法, 并提出自己的改进方法来提高系统的性能。

1 预取算法体系结构

目前 Linux-2.6.29-rc2 内核中采用了吴峰光提出的按需预取算法。吴峰光^[5]改进了预取算法的数据结构和算法框架, 在按需预取算法框架的基础上, 引入页面状态和页面缓存状态, 采用更宽松的顺序性判决

收稿日期: 2009-03-12; 修回日期: 2009-06-12

作者简介: 潘志华(1981-), 男, 硕士研究生, 研究领域为系统软件; 张 涛, 副教授, 硕士研究生导师, 研究领域为安全操作系统、嵌入式系统开发、网络信息安全、数据库安全。

条件,实现了对更多读取模式的支持,可以有效支持:包括异步/非阻塞 IO、多线程交织 IO、顺序随机混合 IO、大规模并发 IO 等。

当应用程序要进行数据的访问时,主要通过 read() 等系统调用接口,经由页面缓存访问一个磁盘文件^[6]。内核对这一标准文件访问路径调用预取算法,跟踪应用程序的访问序列,并进行恰当的预取。下面对按需预取算法进行详细的介绍与分析。

1.1 预取窗口与数据结构

Linux 记录预取状态的数据结构为:

```
struct file_ra_state {
    pgoff_t start; /* 预取开始的位置 */
    unsigned int size; /* 预取的数量 */
    unsigned int async_size;
    /* 何时进行异步读 */
    unsigned int ra_pages;
    /* 最大预取窗口 */
    int mmap_miss;
    /* mmap 访问 */
    loff_t prev_pos;
    /* 上一次读取缓存的位置 */
}; /* linux-2.6.29/include/linux/fs.h
```

Linux 使用 prev_pos 来跟踪以前的请求最后访问的位置。start 表示上一次访问的页面在文件中的偏移量, size 表示从当前访问页面算起的请求页面数。{start, size} 构成了预取的窗口,如图 1 所示。

图 1 中的 async_size 指示了异步预取的位置提前量,即这个流的前方还剩余多少未访问的预取页面时启动下一次预取 I/O。PG_readahead 作为预取标记是在上一次预取 I/O 中设置的,标记的位置在 (start + size - async_size), 用来指示应用程序已经用尽了足够的提前读窗口,此时应读进更多的页,命中它意味着进行下一个预取 I/O 的时机已经来到,因而应立即调用预取例程,进行异步预取,同时清除 PG_readahead 标记。

1.2 预取算法框架

Linux 预取算法主要通过

监控应用程序的读请求和页面缓存来判定应用程序的访问模式,再根据访问模式来决定预取的位置和大小等。预取框架大致可分为两大部分:监控部分与判断处理部分,如图 2,图 3 所示。

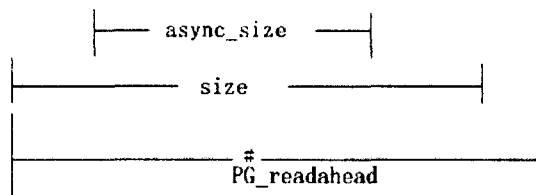


图 1 Linux 预取窗口

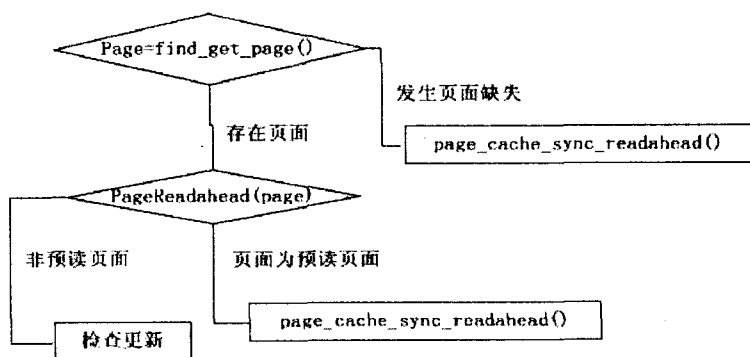


图 2 do_generic_file_read() 处理流程

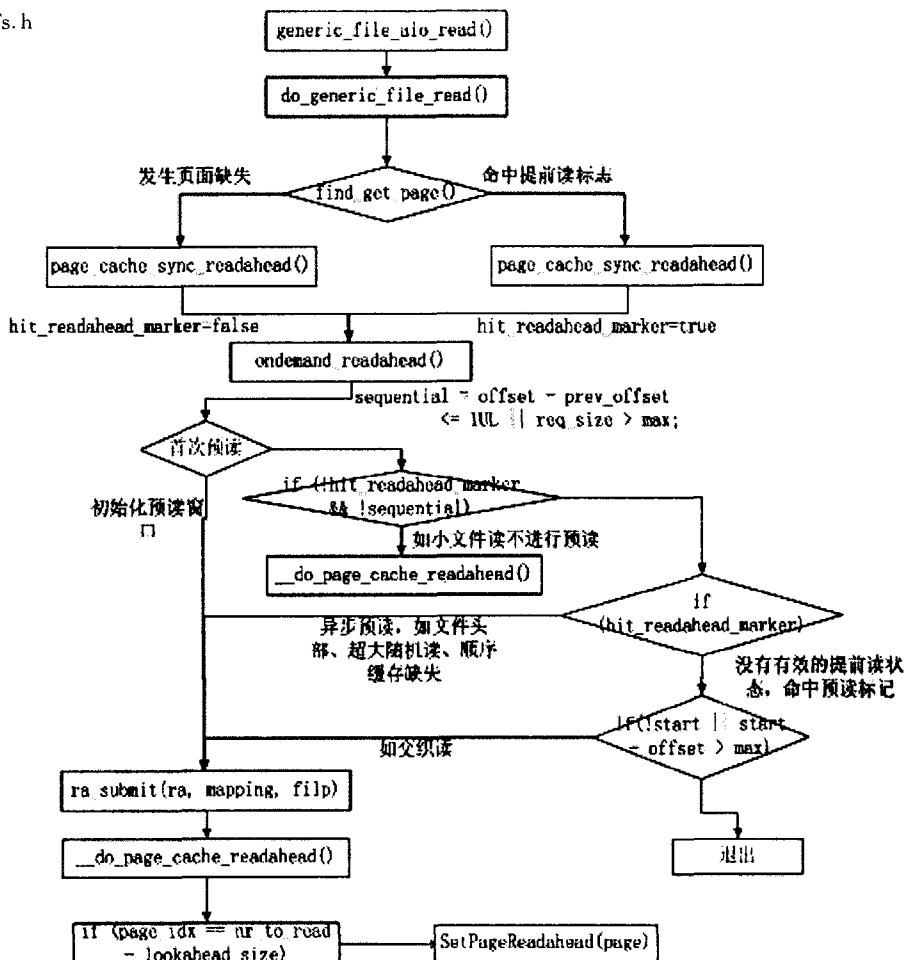


图 3 ondemand_readahead() 处理流程

1.2.1 监控部分

当应用程序要请求访问磁盘上的文件数据时,都会进入统一的文件 I/O 读请求 `generic_file_aio_read()`。该函数分析 I/O 语法控制块,分析读请求的大小、位置后再提交处理函数 `do_generic_file_read()`^[7]。

如图 2 所示:当发生缓存页面缺失时,此时需要立即进行磁盘 I/O 加载当前页面,与此同时应用程序将被临时挂起等待 I/O,并进行同步预取 `page_cache_sync_readahead()`, `hit_readahead_marker = false`;当命中缓存时,如果在已缓存的页面中命中预取标记时,就进行异步预取 `page_cache_async_readahead()`, `hit_readahead_marker = true`。如果缓存页面并不命中预取页面,则检查当前页面是否是最新的,如果不是则需要进一步更新该页面。同时在每次进行预取操作时,要清除旧的 PG_readahead 标记,并在新的页面上设置标记,如果预取窗口进入已缓存页面时,则并不设置 PG_readahead 标记。

1.2.2 判断处理部分

预取算法的主要任务是对应用程序访问模式的识别和预测。Linux 对访问模式的识别与处理主要体现在 `ondemand_readahead()` 部分(\ linux - 2.6.29 \ mm \ Readahead.c),下面对匹配部分进行分析如下:

1) 访问模式的判断。

Linux 预取算法主要支持顺序和随机两种访问模式。当知道一个程序的访问模式为顺序访问时,内核就可将后续要访问的数据从磁盘中提前读到缓存中,这样就可减少访问请求的磁盘寻址与磁头来回移动所带来的时间浪费,从而减少了应用程序的 I/O 等待时间^[8]。然而,在一个多任务的系统环境中,并不可能只有一个单纯的顺序访问模式,而且这种顺序访问也有可能被多个进程的访问所打乱。因此,Linux 对顺序访问模式进行了更加宽松的判断,其将顺序访问模式又分为交织的访问模式、随机顺序混合、顺序性缓存丢失等。

如图 3 所示:当内核是首次预取时,就进行窗口的初始化操作。当内核并没有命中 PG_readahead 标志时且页面偏移为非顺序时,就说明该页面访问为小文件读,此时为了避免污染缓存的状态,应退出预取,进行直接读。当命中 PG-read-ahead 且又没有有效的提交读状态时,则有可能为多线程的交织顺序流,否则则为异步预取,如访问文件头部、超大随机读、顺序缓存丢失等。因为当访问请求为文件的首次首部请求时,通常意味着应用程序将要对文件进行顺序扫描,因而可以适当增加一些后续页面。而当应用程序进行顺序

访问时发生缓存页面丢失时,就可判定应用程序接下来的访问模式为顺序访问。当多个 Linux 线程在同一个文件描述符上进行多个顺序流的访问时,相应的请求就会交织在一起。因而使用页面预取标记 PG_readahead 来识别任意多个并发交织流。同时,当对一个文件进行超大随机读时,为了有效减少访问磁盘时磁头移动和寻道所带来的时间延迟,可对其进行预取。

2) 处理方案。

对于不同的访问模式,Linux 采取不同的方案,具体可见表 1 所示。

表 1 Linux 针对不同访问模式采取的解决方案

访问模式	采用方案
超大读	<code>ra->start = offset;</code>
首次访问文件头部	<code>ra->size = get_init_ra_size(req_size, max);</code>
顺序缓存丢失	<code>ra->async_size = ra->size > req_size? ra->size - req_size: ra->size;</code>
命中标志页面的随机读(如交织读)	<code>ra->start = start;</code> <code>ra->size = start - offset;</code> <code>/* old async_size */</code> <code>ra->size = get_next_ra_size(ra, max);</code> <code>ra->async_size = ra->size;</code>
首次预取	<code>ra->start += ra->size;</code> <code>ra->size = get_next_ra_size(ra, max);</code> <code>ra->async_size = ra->size;</code>
小文件读	退出提前读,以免影响到提前读状态

2 改进方法

Linux 预取算法主要面向通用的系统环境。因而,对于不同的应用环境及负载类型来说,可从以下几方面考虑并加以改进。

2.1 增加支持的访问模式数量

当预取算法支持的访问模式数量越多时,则内核对每一种访问模式的识别就越精确。因此可通过增加预取算法支持的访问模式的数量来提高预取的准确率,以达到提高系统性能的目的,如在科学计算和工程计算中经常使用的跨步读以及在数据库负载中出现的反向读等^[5]。

2.2 块 I/O 层预取

为了防止对预取状态的破坏及缓存的污染,Linux 对小的随机 I/O 访问不进行预取。这对于大多数应用程序来说是一个正确的选择。然而,对以随机访问模式为主的数据库服务器来说,将会导致大量的性能下降。对于大量的小随机 I/O 请求,DingXiaoning^[9]等研究发现通过总结这些请求的规律特点,在块 I/O 层来合并这些小的 I/O 请求,找出这些请求之间存在的顺序性,并进行预取,从而可有效提高系统的性能。

2.3 结合读写的预取操作

Linux 预取算法只实现了对读操作的预取。而 Susanne Albers^[10]等研究表明,结合读和写操作的预取也能有效地提高系统的性能。

2.4 预取调度器

目前针对预取算法的研究主要集中在单系统环境中,Zhe Zhang^[11]等研究发现:在一个多级的存储系统中,可在服务端侧增加一个预取调度器,来根据客户端请求及服务端的资源状况来动态调整预取的大小,从而提高预取的精确度,进一步来提高服务器的服务性能。

3 结束语

文中主要对 Linux2.6.29-rc2 中相关的预取算法源代码进行详细的分析,分析了预取的监控机制及其对访问模式的判断处理方法,最后提出了对预取算法的改进方法与措施,对于提高 Linux 系统的整体性能具有重要的意义。

参考文献:

- [1] 邹丹丹. 高性能存储系统研究[D]. 北京:中国科学院计算技术研究所,2006.
- [2] Cao Pei. Implementation and Performance of Integrated Application - Controlled File Caching Prefetching and Disk Scheduling[J]. ACM Transaction on Computer Systems, 1996, 14(4):311 - 343.
- [3] Patterson H R, Gibson G A, Ginting E, et al. Informed Prefetching and Caching[C]//15th ACM Symp on Operating System Principles. Copper Mountain, Colorado, United States:[s. n.],1995:79 - 95.
- [4] Chang Fay. Using speculative execution to automatically hide I/O latency[D]. [s. l.]:Carnegie Mellon University,2001.
- [5] 吴峰光. Linux 内核中的预取算法[D]. 合肥:中国科学技术大学,2008.
- [6] Love R. Linux 内核设计与实现[M]. 第 2 版. 陈莉君,等译. 北京:机械工业出版社,2006.
- [7] Daniel. 深入理解 Linux 内核[M]. 第 3 版. 陈莉君,等译. 北京:机械工业出版社,2006.
- [8] 孙钟秀. 操作系统教程[M]. 北京:高等教育出版社,2003.
- [9] Ding Xiaoning, Jiang Song, Chen Feng, et al. DiskSeen: Exploiting Disk Layout and Access History to Enhance I/O Prefetch[C]//2007 USENIX Annual Technical Conference. [s. l.]:USENIX Association,2007.
- [10] Albers S, Buttner M. Integrated Prefetching and Caching with Read and Write Requests[C]//Proceedings of the 8th National workshop on algorithms and data structures, 2003, Band 2748 von Lecture Notes in Computer Science. [s. l.]: Springer,2003:162 - 173.
- [11] Zhang Zhe, Lee Kyuhung, Ma Xiaosong. PFC: Transparent Optimization of Existing Prefetching Strategies for Multi-level Storage system[C]//Proceedings of the 28th International Conference on Distributed Computing Systems, DOI 10.1109/ICDCS. Beijing, China:[s. n.],2008.

(上接第 92 页)

上,根据用户和服务的上下文信息进行筛选,将满足上下文信息的 Web 服务优先返回给用户,其结果是对语义 Web 服务查询结果的精化和优化^[8]。文中利用本体来描述上下文的信息模型,定义了用户上下文本体和 Web 服务上下文本体两个本体库,并制定了二者之间的推理规则,根据规则选择出优先返回给用户的 Web 服务,可提高用户的满意度。

参考文献:

- [1] 宋 驰. 基于用户偏好的启发式 Web 服务组合的研究与实现[D]. 北京:北京邮电大学,2008.
- [2] Zheng Xianrong, Yan Yuhong. An Efficient Syntactic Web Service Composition Algorithm Based on the Planning Graph Model[C]//Proc. of ICWS. Beijing, China:[s. n.],2008:691 - 699.
- [3] 唐 磊, 淮晓永, 李明树. 一种基于上下文协商的动态服务组合方法[J]. 计算机研究与发展, 2008, 45(11):1902 - 1910.
- [4] 岳玮宁, 王 悦, 汪国平. 基于上下文感知的智能交互系统模型[J]. 计算机辅助设计与图形学学报, 2005, 17(1):74 - 79.
- [5] Qiu Lirong, Cao Yongcun, Zhao Xiaobing. Promoting adaptation of semantic Web service composition using context information[C]//International Symposium on Computer Science and Computational Technology. Shanghai, China:[s. n.], 2008:652 - 656.
- [6] 郭文英. 基于 SWRL 推理的语义关联发现及其在本体映射与集成中的应用[D]. 杭州:浙江大学,2006.
- [7] 刘克非, 王 红, 王卫玲. 基于语义相似度的 Web 服务发现研究[J]. 计算机技术与发展, 2007, 17(2):16 - 18.
- [8] Maamar Z, Benslimane D, Thiran P. Towards a context-based multi-type policy approach for Web services composition[J]. Data & Knowledge Engineering, 2007, 62(9):327 - 351.