FastQueue：一种高性能的磁盘队列存储管理机制*)

魏青松　卢显良　周　旭
（电子科技大学计算机学院　成都610054）

FastQueue: A High Performance Disk Queue Storage Management Mechanism

WEI Qing-Song　LU Xian-Liang　ZHOU Xu
(Department of Computer Science of UEST of China, ChengDu 610054, China)

Abstract High reliability is the primary requirement for messaging system. Messaging system always utilizes disk queue to temporarily store message to be delivered. Experiments show that Disk queue I/O is the primary performance bottleneck in the messaging system. In this paper we present a high performance disk queue storage management mechanism—FastQueue. The FastQueue utilizes a large file to serve as disk queue to reduce file manage overhead, in which adjacent messages are stored in adjacent disk block. Several messages are written to disk queue in a one large write by Lazy Gathering Write. Several adjacent messages are read into buffer in a one read by Sequential Grouping Prefetch. The Lazy Gathering Write and Sequential Grouping Prefetch policies take full advantage of the disk bandwidth. Experiment shows that performance of the FastQueue is more than an order of magnitude higher than that of traditional disk queue.

Keywords Messaging system, Disk queue, Lazy gathering write, Sequential grouping prefetch

1. 引言

随着消息通信(如消息、短消息)的爆炸式增长，消息传递系统的性能面临严峻的挑战。消息通信的首要特点是高可靠性，在发送人确认消息收到之前必须将消息保存到磁盘上。传统的做法是用一个单独的文件保存一个消息对象，但这样势必在磁盘队列处产生集中而频繁的文件创建、读、写、删除等I/O操作。磁盘队列I/O将成为影响消息传递系统性能的首要瓶颈。

为了提高磁盘队列或缓冲的性能，文[2]提出多级Hash目录，将保存对象的文件散列到多个Hash目录，这样明显减少文件搜索时间，但没从根本上改变用一个单独文件保存一个消息对象的存储管理方法，繁重的存储管理开销依然存在。文[3]提出一种将多个对象用一个文件保存的存储管理方法，从而减少文件管理开销，其缺点是没有实现紧凑的磁盘布局，读写性能不高。

本文针对消息传递服务的自身特点提出一种高性能的磁盘队列存储管理机制—FastQueue。FastQueue采用预先分配的连续磁盘块作为磁盘队列，将多份消息对象保存到一个文件中，消除或减少文件创建、删除等文件管理开销，并减少了磁盘碎片；采用延迟聚集写策略，通过一次写操作将多份消息对象顺序写入磁盘队列，消息被写入连续的磁盘块，减少了磁盘搜索旋转时间，磁盘带宽得到充分利用，极大地提高写性能；根据消息先来先服务的特点，采用预先连续读策略，将磁盘队列中连续的几份消息预先读入缓存，不仅减少读操作次数，而且读连续的磁盘块也充分利用磁盘带宽，读性能得到极大的提高，加快了消息投递速度。实验表明，FastQueue能获得比传统磁盘队列高出约230%的性能。

2. 文件系统特征

一次磁盘随机访问时间T_{access}由3个部分组成：

$$T_{access}=T_{seek}+T_{rotate}+T_{transfer}$$

式中，T_{seek}代表磁头寻道时间，T_{rotate}代表磁头旋转定位时间，$T_{transfer}$代表磁头实际传输数据时间。在这3部分时间中，只有$T_{transfer}$是真正从磁盘传输数据的有效时间。但是在T_{access}的3个组成部分中，T_{seek}和T_{rotate}通常占据大部分时间。对于小文件，T_{seek}几乎是T_{access}的1/2。

正是由于现代磁盘寻道时间和寻道距离不是线性关系，读写多个4k或8k磁盘块的开销仅比读写单个磁盘块的开销多一点，并不是成倍增长。表明大文件的读写能充分利用磁盘带宽，而小文件的开销主要在磁盘搜索、定位。

Unix文件系统并没有针对这种情况在文件布局、存储方式上进行优化设计，导致文件系统在大文件读写方面比小文件表现好得多[1]。

一方面，小文件在当前的文件系统中无法充分利用磁盘带宽，因而无法获得高的读写性能，另一方面，随着文件系统的运行，小文件操作将产生大量的磁盘碎片，这样文件将保存到不连续的磁盘块，从不连续的磁盘块读写数据将导致长时间的磁盘搜索，严重降低读写性能。

3. 传统磁盘队列I/O分析

消息传递服务器普遍采用的磁盘队列是：一个单独文件保存一份消息(A File per Message)，如图1所示。传统磁盘队列的存储管理有两种方式，一种是扁平存储，将所有文件存放在同一个目录。将大量消息对象以文件方式存放在单个目录中，势必大大减少文件检索速度，单个目录下的文件越多，查

*)本文受国家95重点攻关项目支持。魏青松　博士研究生，主要研究方向：计算机网络、网络存储、分布式操作系统等。卢显良　教授，博士生导师，主要研究方向：计算机网络、操作系统。周　旭　博士研究生，主要研究方向：计算机网络、文件系统。

找文件的时间越长。显而易见，这种存储方式是低效率的。另一种方式是采用 Hash 多目录，该方法为了避免单目录中存放大量文件，减少文件查询时间，将文件按一定算法散列到多个 Hash 目录，如 Qmail 就采用该存储管理方法，这样显著减少了文件查找时间，对磁盘队列进行了优化，提高了消息处理速度。

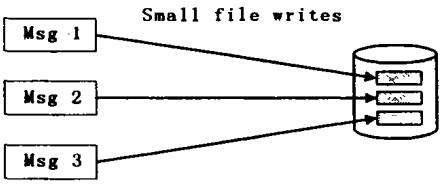

图1 传统队列

在开发消息传递服务器 FSmail 初期，我们采用的是一个文件保存一份消息对象的传统队列方式，并散列到10个 Hash 目录。本文对 FSmail 的消息大小分布进行了统计，统计结果如图2所示。从图2可以看出，85%的消息小于8k。如果每个消息用单独的文件保存，势必产生大量的小文件，不仅将极大增加存储管理开销，并且小文件在当前文件系统中无法获得良好的读写性能。

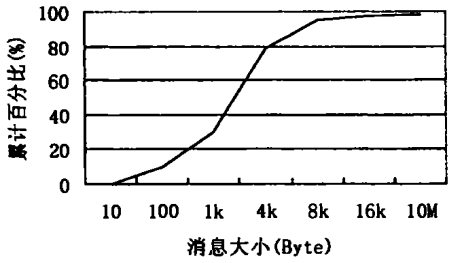

图2 消息大小分布

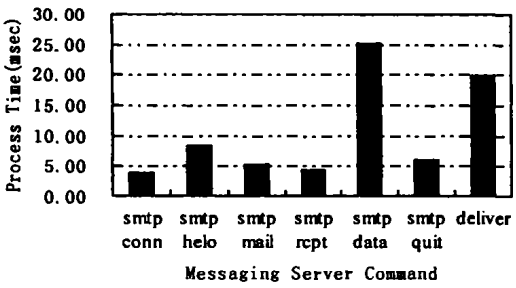

图3 消息传递服务器命令处理时间

采用 Specmail 2000在100Mb/s 以太网环境模拟5000用户对 FSmail 进行了性能测试(平均消息大小为1kB)。图3为 FSmail 各命令的处理时间。可以看出，两个主要与磁盘队列交互的 SMTP DATA 命令和投递过程 Deliver 比其他命令占用的处理时间明显多。

DATA 命令按字节从网络接收用户消息，创建一文件并写入消息，将该文件压入磁盘队列，等待投递。DATA 命令涉及网络 I/O、创建文件、写消息等操作。Deliver 从磁盘队列打开保存消息的文件，读出消息，投递成功后删除该文件。Deliver 过程涉及文件打开、读、删除等文件操作。

为了更详细地了解磁盘队列的 I/O 情况，在消息传递服务器 FSmail 中加入测试代码，对磁盘队列的 I/O 进行了测试，测试结果如图4所示。可以看出，文件 I/O 比网络 I/O 占用多得多的开销，而在文件 I/O 中创建文件和写文件尤为突出。如果能对磁盘队列进行存储优化，减少文件 I/O 的开销，将极大提高服务器的性能。

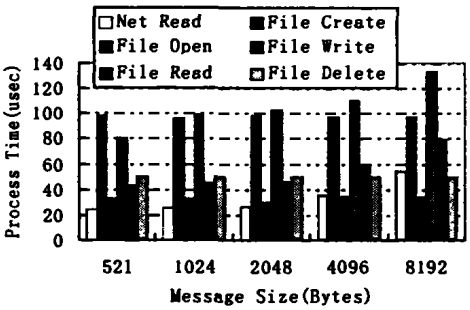

图4 传统磁盘队列 I/O 情况

通过以上测试和分析，Hash 目录式磁盘队列仍存在如下缺点：1)Hash 目录虽然减少了文件检索时间，但没有从本质上改变用单个文件保存一份消息的存储方法，无法根本减少文件管理开销(创建和删除文件)和频繁的文件操作(读写文件)，大量而频繁的小文件操作依然集中在磁盘队列，磁盘队列 I/O 依然是影响消息传递服务器性能的关键；2)随着服务器的长时间运行，由于频繁地创建、删除文件，势必产生大量的磁盘碎片，而不连续的文件系统布局又将极大地降低小文件的读写性能。

所以，针对消息传递服务器的 I/O 特点，对磁盘队列进行存储优化，减少文件管理开销，提高文件读写性能，将极大提高消息传递服务器的性能。

4. FastQueue 的实现机制

4.1 基本思想

针对传统队列的 I/O 问题，并结合消息通信的特点，本文提出快速磁盘队列 FastQueue，其基本结构如图5所示。

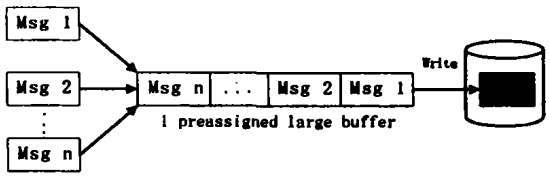

图5 FastQueue 队列基本结构

FastQueue 从3个方面提高 I/O 性能：

1)预先创建连续的磁盘块作为磁盘队列，将多份消息保存到一个文件，消除或减少文件创建、删除等文件管理开销[3]。

2)采用延迟聚集写策略，通过一次写操作将多份消息对象顺序写入磁盘队列，这样消息对象被写入连续的磁盘块，减少了磁盘搜索、旋转时间，磁盘带宽得到充分利用，极大地提高写消息的性能。

3)根据消息先来先服务的特点，采用预先连续读的策略，将磁盘队列中连续的几份消息预先读入缓存，这样不仅减少了读操作次数，而且读连续的磁盘块也充分利用了磁盘带宽，读消息的性能得到极大提高，加快了消息的投递速度。

4.2 FastQueue 的磁盘布局

快速磁盘队列 FastQueue 在系统初始化时预先占据连续的物理磁盘块作为磁盘队列，为提高消息读写创造条件。

在快速磁盘队列 FastQueue 内，采用紧凑的存储方式，消息正文和消息索引节点作为一个数据单元整体存放，相邻消息顺序存放，尽量维护队列内对象分布的连续布局[4]，如图6所示。

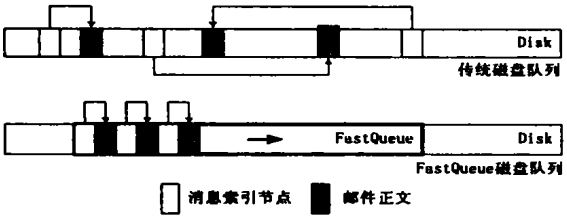

图6 磁盘队列布局比较

聚集写将多份相邻的消息一次性顺序写入磁盘队列，相邻消息对象被写入连续的物理磁盘块。为了维护队列内连续紧凑的存储布局，对于投递失败的消息，FastQueue 将其拷贝到重试队列中等待再次投递，并将删除标志置位。

4.3 消息索引节点 MIN

为了提高从磁盘队列读出消息的速度，FastQueue 在内存中维护一份消息索引节点 MIN(Message Index Node)表。它是等待投递的消息索引。内存中消息索引节点 MIN 包含消息号、消息在 FastQueue 队列中的偏移位置、磁盘消息节点大小、消息正文大小等字段，是磁盘上消息索引节点的简化。消息索引节点 MIN 实现了消息号和消息位置的映射，起到快速定位的作用。如图7所示。

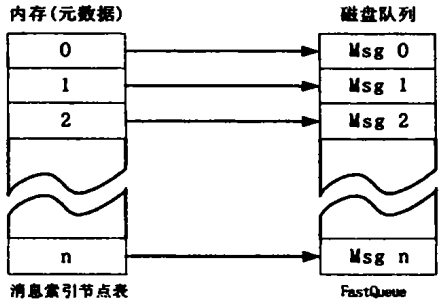

图7 消息索引节点表

根据消息先来先服务的特点，将消息索引节点 MIN 以队列形式组织。当消息到来时，FastQueue 分配一个消息索引节点 MIN，填写相应字段，并追加到消息索引节点 MIN 表尾；当消息投递成功后，FastQueue 删除并释放该消息索引节点 MIN。

服务器崩溃后，扫描整个 FastQueue 队列，从队列中恢复消息索引节点 MIN 表，保证数据一致性。

4.4 FastQueue 延迟聚集写 LGW

客户向服务器发送消息，服务器接收消息并写入磁盘队列，然后向客户发送确认通知，客户接到服务器的确认通知后，才确认消息已交给服务器，这时服务器有义务保证将消息正确投递。客户在发送消息过程中，服务器崩溃，即使没将消息写入磁盘队列，由于没向客户发送确认通知，客户将重发消息。由于这个特点，FastQueue 就可以采用延迟聚集写 LGW (Lazy Gathering Write)策略[5]。因为服务器(并发处理多个客户)不必接到客户的一份消息就立即写磁盘，可以等多个客户的多份消息写满缓冲区后再写磁盘，然后通知客户端，不用担心消息在缓冲区中丢掉，因为即使丢掉，由于没得到服务器确认，客户会重发消息。

延迟聚集写 LGW 策略：当消息到来时，不是马上写入磁盘队列，而是先将消息写入缓冲区，等到缓冲区满或定时器超时，系统将缓冲区中的数据一次写入磁盘队列，客户端不会感到明显的时延。延迟聚集写 LGW 策略不仅大大减少了磁盘同步写的次数，而且可以保证将相邻消息写入相邻的磁盘块，减少了磁盘搜索、旋转时间，磁盘带宽得到充分利用，提高了写消息的效率。延迟聚集写 LGW 策略维护了 FastQueue 紧凑的磁盘布局，为其他操作的高效率执行创造了条件。

4.5 FastQueue 预先连续读 SGP

消息投递采用先来先服务的方式，而在 FastQueue 的磁盘布局中，相邻消息在快速磁盘队列 FastQueue 中顺序排列；另一方面，由于内存的廉价，专用大规模消息传递服务器有条件配备大容量的内存，而在磁盘队列设计中可以采用大的读缓存。因此，在读消息时，FastQueue 采用预先连续读 SGP (Sequential Grouping Prefech)的策略[2]。

预先连续读 SGP 策略：从快速磁盘队列 FastQueue 读消息时，根据读缓冲区大小和内存中消息索引节点 MIN 表，计算出当前要读出的数据块大小，然后将该数据块一次性读入缓冲区。这样不仅大大减少了读磁盘的次数，而且通过一次磁盘搜索和旋转动作，尽量多地读出连续的数据块，磁盘带宽得到充分利用，提高了读消息的效率，加快了消息投递速度。

4.6 FastQueue 删除消息对象

传统的磁盘队列在消息投递完成后，删除保存该消息的文件，这样频繁的文件删除操作不仅增加磁盘 I/O 负荷，而且产生大量的磁盘碎片，而碎片化的文件系统反过来将严重降低文件的读写性能。

FastQueue 采用如下的删除策略：当消息投递成功后，删除并释放该消息索引节点 MIN，并将磁盘中该消息索引节点的删除标志置位，将消息删除操作通过一字节写操作来完成。这样做的目的是当服务器发生硬件故障后，内存中消息索引节点 MIN 表将丢失，服务器重启后，通过扫描 FastQueue 的消息索引节点重建内存中消息索引节点 MIN 表，保持内存中元数据和 FastQueue 中消息数据的一致性。

5. 性能对比

将消息传递服务器 FSmail 中一份消息单独用一个文件保存的传统队列方式改为快速磁盘队列 FastQueue，并采用 Specmail 2000 在 100Mb/s 以太网环境模拟 5000 用户对 FSmail 进行了性能对比测试。图8为 FSmail 采用传统队列和快速磁盘队列各命令的处理时间对比。

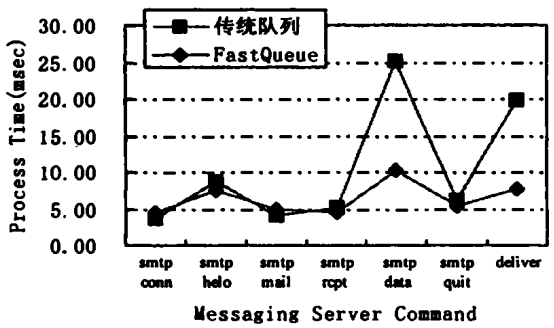

图8 传统磁盘队列和 FastQueue 性能对比

(下转第88页)

to,_switch_to 函数的最后是 ret 语句,而 ret 语句的功能就是弹出堆栈的最上面的值作为当前的 EIP,这样 next 进程的 EIP 就出栈成为当前执行的 EIP,完成了整个任务上下文的切换过程。以下是进程切换的代码。

```
#define switch_to(prev,next,last) do { \
1     asm volatile("pushl %%esi\n\t" \
2                  "pushl %%edi\n\t" \
3                  "pushl %%ebp\n\t" \
4                   "movl %%esp,%0\n\t" / * save ESP,
                    即 %%esp ->prev->tss.esp * /
5                  "movl %3,%%esp\n\t" / * restore ESP,
                   即 next->tss.esp -> %%esp * /\
6                  "movl $1f,%1\n\t" / * save EIP,这里是将第10
                   行的 IP 地址存入 prev->tss.eip * /
7                 pushl %4\n\t" / * restore EIP ,将 next->tss.eip
                  压入栈,这样当执行_switch_to 的 ret 后,程序 EIP
                  就到了 next->tss.eip * / \
8                  "jmp__switch_to\n" \
9                  "1:\t" \
10                 "popl %%ebp\n\t" \/ * 这里就是第6行存储的
                   EIP
11                 "popl %%edi\n\t" \
12                 "popl %%esi\n\t" \
13                 :" = m" (prev->tss.esp)," = m" (prev->
                   tss.eip), \
14                  "=b" (last) \
15                 :"m" (next->tss.esp),"m" (next->tss.eip), \
16                 "a" (prev), "d" (next), \
                   "b" (prev)); \
} while (0)
```

小结 进程实现是分时系统的核心部分之一,且与操作系统多方面的内容联系,到目前,我们已给出了进程调度的完整线索,从时钟系统到调度策略,这是大多数分时操作系统遵循的设计原则。Linux 是十分成功的操作系统,许多方面的设计都十分新颖,而且在实现上也相当富于技巧,其设计思路和方法都是值得借鉴的。

参 考 文 献

1 Tanenbaum A S, Woodhull A S. Operation Systems Design and Implementation, Second Edition, Prentice-Hall, 1997. 47～148

2 The Portable Application Standards Committee of the IEEE Computer Society. P1003. 13 draft standard for information technology--standardized application environment profile--posix realtime application support(aep):[Technical report]. IEEE, 1998

3 Stalling W. 操作系统内核与设计原理,第四版. 电子工业出版社, 2001

4 李善平,郑扣根. Linux 操作系统及实验教程. 机械工业出版社, 2001

5 毛德操,胡希明. Linux 内核源代码情景分析(上册). 浙江大学出版社,2001

(上接第83页)

从图8可以看出,采用快速磁盘队列 FastQueue 后,DATA 和 Deliver 命令的处理时间成倍下降。如上文分析,DATA 和 Deliver 命令主要与磁盘队列交互,DATA 和 Deliver 命令的高效来自快速磁盘队列 FastQueue 优化的存储布局和高效的消息读写策略。因此,快速磁盘队列 FastQueue 从整体上提高了消息传递服务器的性能。

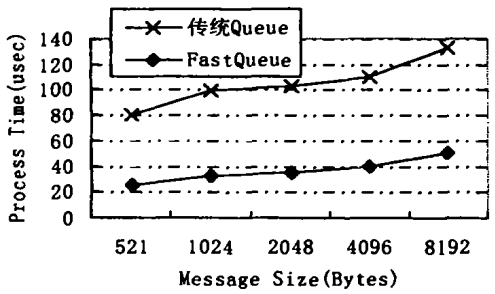

图9 传统磁盘队列和 FastQueue 写性能对比

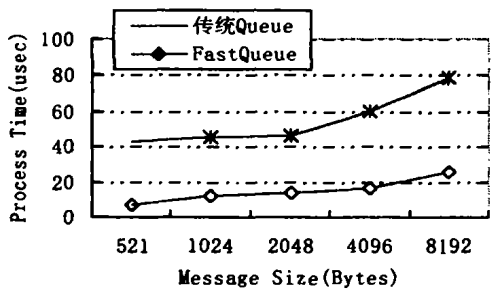

图10 传统磁盘队列和 FastQueue 读性能对比

进一步测试 Data 和 Deliver 命令中读、写性能,图9为写消息性能对比,图10为读消息性能对比。可以看出读、写消息的性能得到成倍的提高。

结束语 本文详尽分析了消息传递服务的存储开销,并提出一种高性能的磁盘队列存储管理机制—FastQueue。FastQueue 采用预先分配的连续磁盘块作为磁盘队列,将多份消息对象保存到一个文件中,消除或减少文件创建、删除等文件管理开销,并减少了磁盘碎片;采用延迟聚集写和预先连续读策略,通过一次磁盘搜索旋转定位,尽量读写更多的有效数据,不仅减少了读写次数,而且读写连续的磁盘块也充分利用了磁盘带宽。实验表明,FastQueue 能获得比传统磁盘队列高出约230%的性能。

参 考 文 献

1 Gregory R, Ganger M, Kaashoek F. Embedded Inodes and Explicit Grouping: Exploiting Disk BandWidth for Small Files. Annual USENIX Technical Conference (Anaheim, CA), Jan. 1997. 1～17

2 Kathy C M, Richardson J. Reducing the Disk I/O of Web Proxy Server Caches, USENIX Annual Technical Conference Monterey, California, USA, June. 1999

3 Evangelos P, Markatos Manolis G, Pnevmatikaos H K D. Secondary Storage Management for Web Proxies, 2nd USENIX Symposium on Internet Technologies & Systems Boulder, Colorado, USA, Oct. 1999

4 John M R, Ousterhout K. The Design and Implementation of a Log-Structured File System. ACM Transaction on Computer Systems, Feb. 1992. 10～22

5 Matthew McCormick Jonathan Ledlie. A Fast File System for Caching Web Objects, First Annual Symposium on Operating Systems Design, Implementation, and Evaluation (OS-DIE 1)