用异步I/O请求处理提高流媒体服务器支持并发访问的性能*)

李 中 王 刚 刘 璟

(南开大学信息技术科学学院 天津 300071)

摘 要 流媒体服务是一种I/O请求密集型应用,为了提高流媒体服务器支持并发访问的能力,需要提高其处理I/O请求的能力。目前的操作系统通常支持同步处理I/O请求,但因此产生的进程的阻塞和中断响应处理将对服务器处理I/O请求的效率产生负面影响。我们采用异步方式处理I/O请求,消除了上述的影响,优化了流媒体服务器处理I/O请求的效率,提高了其支持并发访问的能力。

关键词 流媒体,I/O请求,进程,异步,阻塞,中断

Asynchronously Manipulate I/O Requests in the Streaming Media Server to Support More Concurrent Data Accesses

LI Zhong WANG Gang LIU Jing

(Collage of Information Technology Science, Nankai University, Tianjin 300071)

Abstract In order to support concurrent data accesses, Streaming Media Server need to implement large number of I/O requests. Presently, most operation systems supply synchronous method to implement I/O requests, but in this case blocked processes and handling interrupts produce negative performance. In this paper we suggest using asynchronous method to process I/O request to remove above side effects and improve the ability of Streaming Media Server.

Keywords Streaming media, I/O request, Process, Asynchronous, Block, Interrupt

1 研究背景

目前,以对视频、音频等连续媒体数据的访问、传输和播放为主要内容的流媒体技术得到了广泛的研究与应用。在流媒体的应用中,流媒体服务器处于核心地位,它为客户端提供流媒体数据检索、访问与传输服务。为了保证流媒体在客户端连续顺畅地播放,流媒体服务器必须以流媒体的播放速率向客户端持续传输流媒体数据;在整个传输过程中,服务器需要持续地从存储系统中读取流媒体数据。衡量流媒体服务器性能的最重要的技术指标就是它能够支持的客户端并发访问的数量。

为满足一个客户端的连续播放需求,流媒体服务器按一定的周期向存储系统发出I/O请求,每次从存储设备上读取一定大小的数据块;存储系统必须保证在一个周期结束之前,完成该周期的I/O操作,以便服务器将读取的数据块在下一个周期向客户端发送。I/O请求周期的大小由流媒体服务器的参数决定,通常为一秒到几秒;每次读取的数据块的大小通常为几kB到几MB。图1描述了流媒体服务器从存储系统中周期性读取数据块,满足客户端连续播放需求。

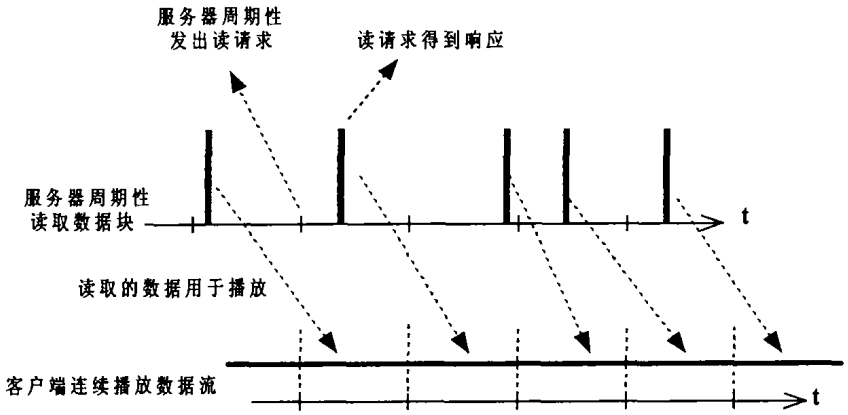

图1

流媒体服务器的存储系统通常以磁盘作为存储介质,众多的磁盘组成磁盘阵列,流媒体数据散布存储在阵列中的磁

*)本课题得到以下基金的资助:国家自然科学基金(编号:60273031),高等学校博士点专项科研基金(编号:20020055021),南开大学科研启动基金。李 中 博士研究生,研究方向为存储系统性能分析、并行与分布式系统;王 刚 讲师,研究方向为并行与分布式系统,海量存储技术;刘 璟 教授,博士生导师,研究方向为分布式系统、并行计算。

盘上。服务器通过主机总线适配器(HBA),利用高速的并行SCSI总线或Fibre Channel访问磁盘阵列上的数据。目前高性能的磁盘阵列每秒可以处理几十到上百k的I/O请求,数据吞吐率达到几百MB/s;但由于磁盘介质自身机械特性的限制,磁盘阵列对I/O请求的响应延迟仍需要10ms左右。

由于存储系统的数据吞吐率远大于流媒体的播放速率,因此流媒体服务器能够支持大量客户端的并发访问,此时服务器要为每一个客户端周期性地从存储系统读取数据块。由于流媒体服务器上存在大量的流媒体对象;流媒体对象的数据量相对较大;每个客户端访问的流媒体对象不同,且开始访问的时刻存在随机性,因此客户端读取的数据块恰好在服务器缓存中的可能性很小,通常需要直接从磁盘阵列读取数据。由此看来,流媒体服务是典型的I/O密集型应用,流媒体服务器支持大规模客户端并发访问的关键在于能否在单位时间内完成大量的I/O操作;而对I/O请求的响应时间要求并不高,只要在请求周期内完成即可。我们通过优化流媒体服务器处理I/O请求的方法,重新设计I/O接口,提高流媒体服务器处理I/O请求的效率,从而提高支持客户端并发访问的能力。

2 传统的同步处理I/O请求的方式

目前大多数操作系统都采用同步的I/O请求处理方式从存储系统中读取数据。在此方式下,发出读I/O请求的进程将处于阻塞状态,直到存储系统完成I/O请求所指定的操作并唤醒阻塞的进程。同步处理方式适用于传统的文件和数据库的访问模式,提供了I/O请求响应速度和吞吐量的折衷。下面以Linux操作系统为例,分析同步I/O请求处理,其他操作系统基本与之类似。图2为同步I/O请求处理的流程。

1. 用户态的进程发出I/O请求,通过系统调用,进程从用户态进入操作系统内核态;

2. 在内核态,操作系统为I/O请求分配缓冲区,并为其建立相应的I/O请求数据结构;

3. 进程将I/O请求数据结构加入存储系统驱动程序的I/O请求队列中,并进入阻塞状态,等待存储系统完成I/O请求;

4. 存储系统驱动程序对I/O请求队列进行调度,将I/O请求发送到存储系统;

5. 存储系统处理I/O请求,将相应的数据块通过DMA机制传输到在第2步为该请求分配的缓冲区;

6. 存储系统完成数据传输后,向主机发出中断,操作系统响应中断并调用相应的中断处理函数;

7. 中断处理函数唤醒被阻塞的进程,通知本次I/O请求已经完成,使其从阻塞点继续执行。

从上面的描述可以看到,发出I/O请求的进程在步骤3完成后被阻塞,等待存储系统完成I/O请求;直到在步骤7被中断处理函数唤醒,继续执行剩下的工作。同步I/O请求处理带来的系统消耗主要有两类:由被阻塞进程引起的,以及由中断响应和处理引起的系统消耗。

发出I/O请求的进程被阻塞后,会对系统的性能产生两个方面的负面影响:无效计算时间(false computation time)和无效空闲时间(false idle time)。无效计算时间是指操作系统调度、运行那些等待I/O请求结束的被阻塞进程所消耗的时间。进程虽然被阻塞,但仍参加操作系统日常的进程调度,因此无效计算时间包括被阻塞进程被调度、放弃调度、与其他进程进行上下文切换的时间消耗。如果进程被阻塞的时间过长,这种无效的进程调度将进行多次。无效空闲时间是指在当前所有的进程都被阻塞,等待各自I/O请求的完成时,处理器循环运行空闲指令(idle)所浪费的时间。无效空闲时间不同于由于处理器因缺少工作而引起的正常空闲时间。

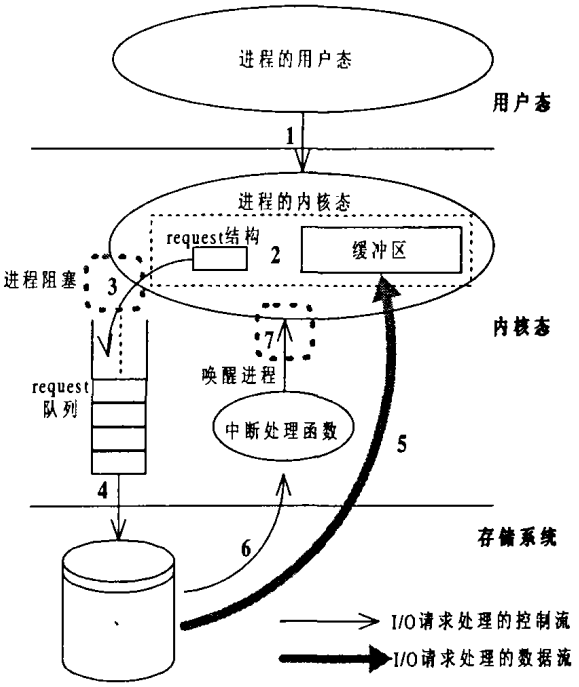

图2 I/O请求的同步处理

在同步I/O请求处理过程中,存储系统每次完成将I/O请求的数据块传输到缓冲区之后,都会触发中断;操作系统将响应中断,并调用中断处理函数,唤醒被阻塞的进程,完成一次I/O请求操作。操作系统响应中断和处理中断都要消耗系统时间。

上述的两类的负面影响在普通的文件或数据库访问中并不明显;但对流媒体服务这种I/O密集型应用来说,它们的影响却很大。随着流媒体服务器所处理的并发客户端的数量增加,并发访问存储系统的进程的数量也将增加,同一时刻被阻塞的进程的数量也会增加,被阻塞的进程被调度的比例也将增加,这将导致无效计算时间的增加。存储系统在单位时间内完成大量的I/O请求,将产生大量的中断,操作系统频繁地响应中断,也将消耗大量的系统时间。目前的主流服务器操作系统的中断响应延迟时间和进程调度切换时间约为数十微秒,而中断处理函数的运行时间可能更长,当系统中只存在较少I/O请求时,它们的消耗似乎微不足道,但当希望系统每秒钟处理上千个I/O请求时,它们的累积对系统的影响就相当可观了。更为严重的是当所有的进程都在等待I/O请求的完成时,所有的进程都无法进行下一步的操作,系统将产生大量的无效空闲时间,而无法向存储系统发出更多的I/O请求。可见,流媒体服务器采用传统的同步方式处理大量的并发I/O请求并不是一个优化的选择。

3 用异步方式处理I/O请求

针对同步I/O请求处理方式在流媒体服务器应用中存在的问题,我们采用异步I/O请求处理方式,希望消除由于进程阻塞和中断响应处理造成的系统消耗。在异步处理方式中,进程向存储系统发送I/O请求之后,不必等待存储系统完成I/O请求,将立即继续执行后续的程序;存储系统将I/O

请求的数据块传输到服务器的缓冲区后，也不再通过中断机制调用中断处理函数。图 3 为异步 I/O 请求处理的流程。

1. 用户态的进程发出 I/O 请求，通过系统调用，进程从用户态进入操作系统内核态；

2. 在内核态，操作系统为 I/O 请求分配缓冲区，并为 I/O 请求建立相应的 I/O 请求数据结构；

3. 进程将 I/O 请求数据结构加入存储系统的 I/O 请求队列中，立即返回，继续执行后续的程序，不等待 I/O 请求操作的结束；

4. 存储系统从 I/O 请求队列中取出 I/O 请求；

5. 存储系统处理 I/O 请求，并根据请求将相应的数据块通过 DMA 机制传输到在第 2 步为该请求分配的缓冲区；

6. 存储系统处理完毕 I/O 请求后，将完成请求的信息加入完成请求队列中；

7. 一个内核查询线程定期启动，查询完成请求队列中的记录；

8. 内核查询线程设置已完成 I/O 请求的缓冲区的标志位，结束本次 I/O 请求。

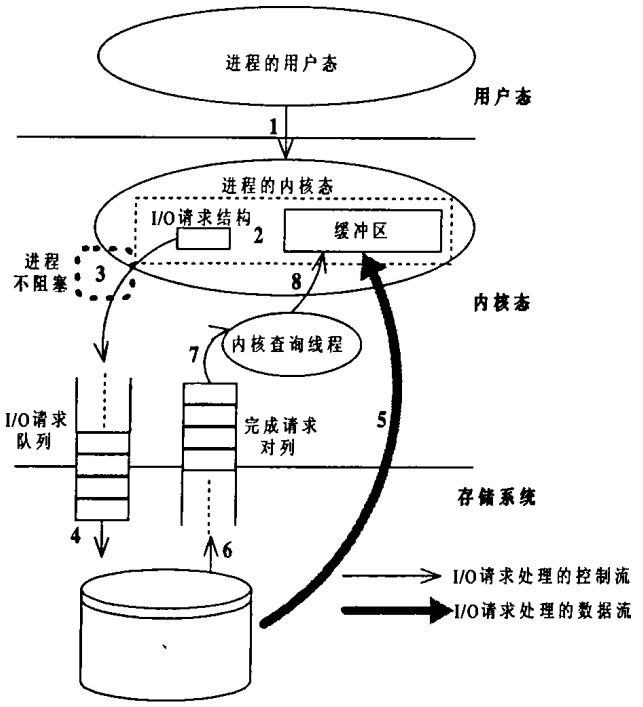

图 3 I/O 请求的异步处理

为了实现对 I/O 请求的异步处理，避免进程阻塞，流媒体服务器采取与操作系统常规 read 操作不同的语义从存储系统中读取数据。用户进程发出读取数据的系统调用后，并不等待 I/O 请求处理的完成，只是将读取数据块的 I/O 请求加入 I/O 请求队列，就立即返回读取操作的调用点，继续执行后续的程序。此时进程读取的数据块并未真正读入缓冲区，进程也不能执行与此数据块相关的操作，但可以执行与此数据块无关的操作或继续发送其他的 I/O 请求；当进程通过缓冲区标志位发现读取数据块的 I/O 请求已经真正完成后，才能执行与此数据块相关的后续操作。无阻塞的进程不仅避免了无效计算时间和无效空闲时间的产生，而且能够向存储系统发送尽可能多的读数据 I/O 请求（只受 I/O 请求队列长度和存储系统处理能力的限制），充分利用高性能磁盘阵列提供的强大的 I/O 请求处理能力。

在异步 I/O 请求处理中，当存储系统将 I/O 请求的数据块传输到缓冲区后，采用查询方法完成 I/O 请求的处理，而不采用中断机制。传统的计算机理论认为采用 CPU 查询方法进行 I/O 操作的效率很低，普遍推荐采用中断技术进行 I/O 操作。但在存在大量 I/O 请求的流媒体服务器环境中，合理地使用查询方法却能得到较好的效果。在服务器操作系统内核中运行着一个查询线程，该线程定期被唤醒，并查询和处理累积在完成请求队列中的 I/O 请求。由于处理完成请求队列中的一个 I/O 请求只需消耗极少的时间（设置缓冲区的标志位），因此查询线程一次唤醒就能处理完成请求队列中累积的所有 I/O 请求，这是一个定时批处理过程。虽然定时查询会增加某些 I/O 请求的响应时间（通常设置线程唤醒周期为 100ms），但由于避免了每次 I/O 请求处理时中断响应的消耗，实际将缩短处理每个 I/O 请求所消耗的时间。可以认为定时查询方法虽然增加了 I/O 请求处理的响应时间，却也增加了 I/O 请求处理的吞吐量，而在流媒体服务中，I/O 请求处理的适当延迟是允许的。

实现异步 I/O 请求处理的关键环节是建立两个从服务器操作系统和存储系统都能访问的请求队列。其中一个队列用于存放等待存储系统处理的 I/O 请求，另一个队列存放存储系统已经处理完毕的 I/O 请求，我们分别称它们为 I/O 请求队列和完成请求队列，它们构成了异步处理 I/O 请求的 I/O 接口，它的实现细节将在下一节描述。

4 异步处理 I/O 请求的 I/O 接口

在流媒体服务器上实现异步 I/O 请求处理需对服务器的操作系统进行必要的修改，主要包括修改读取数据操作的系统调用，实现异步 I/O 语义；编写内核定时查询线程，处理完成请求队列。同时必须重新设计服务器与存储系统之间的 I/O 接口。

在实现异步 I/O 请求处理的 I/O 接口中，流媒体服务器和存储系统之间通过 I/O 请求队列和完成请求队列进行 I/O 请求信息的通讯。为了建立从服务器端和存储系统端都能访问的两个队列结构，我们没有采用基于 SCSI 协议的常规 HBA（主机总线适配器），而是重新设计了 I/O 接口卡。

I/O 接口卡与主机的 PCI 总线相连，并通过多条 SCSI 总线与磁盘阵列连接，接口卡上的高性能 CPU 运行嵌入式实时操作系统，同时还拥有一定数量的内存。I/O 接口卡上的一段内存通过 PCI 总线映射到服务器的总线地址上，形成一段 I/O 内存；服务器的操作系统将这段 I/O 内存映射到操作系统的内核空间；通过对 I/O 内存的访问，服务器能够访问 I/O 接口卡上的内存。服务器和存储系统共同访问的 I/O 请求队列和完成请求队列就建立在这段 I/O 内存上。

每一个请求队列在 I/O 内存上占据一段连续空间，这段空间被划分为大小相等的若干条记录，每条记录能够存放一个请求的信息，从而构成了请求信息的队列。两个请求队列都是循环队列，当对队列中记录的访问超过队列尾部时，将返回到队列的头部。对一个请求队列，从服务器端和存储系统端必须访问同一段物理内存，相同的队列长度和相同的纪录数据结构。I/O 请求队列和完成请求队列中纪录的主要字段如下：

I/O 请求队列中记录的主要字段

队尾标志	状态	读写命令	数据逻辑块号	数据大小	缓冲区物理地址	I/O 请求回调函数指针

完成请求队列中记录的主要字段

队尾标志	状态	读写命令	数据逻辑块号	I/O 请求回调函数指针

图 4　I/O 请求队列和完全队列中的记录

队尾标志字段指示当前记录是否为循环队列的尾部，如到达队列尾部，下一条将记录返回队列首部。状态字段指示当前记录中是否为尚未处理的 I/O 请求记录或完成请求记录。缓冲区物理地址用于存储系统和服务器缓冲区之间的 DMA 操作。I/O 请求回调函数指针将传递给服务器的定时查询线程，通过调用相应的回调函数最终完成异步 I/O 请求的处理。

I/O 接口卡对服务器屏蔽了流媒体数据在存储系统上的实际物理存储位置，存储系统中用于存储流媒体数据的空间在服务器的操作系统内映射为一个具有连续线性地址的逻辑块设备。服务器发出的 I/O 请求只需标明访问数据的起始逻辑块号和数据块的大小，从逻辑地址到实际物理地址的转换由 I/O 接口卡完成。

运行在服务器上的 I/O 接口卡驱动程序维护着两个记录指针 P_{HR} 和 P_{HC}，P_{HR} 指向 I/O 请求队列中下一条 I/O 请求将加入的记录，P_{HC} 指向完成请求队列中下一条将被取出的记录。I/O 接口卡上运行的程序上也维护着两个记录指针 P_{SR} 和 P_{SC}，P_{SR} 指向下一条将从 I/O 请求队列中取出的记录，P_{SC} 指向下一条将向完成请求队列中加入完成 I/O 请求的记录。在初始状态四个指针都指向各自队列的起始记录，并只能向同一个方向逐记录单向循环移动。如图 5 所示，P_{SR} 跟随 P_{HR} 之后移动，存储系统完成服务器加入 I/O 请求队列的 I/O 请求；P_{HC} 跟随 P_{SC} 之后移动，服务器的定时查询线程处理存储系统加入完成请求队列的记录项。通过这四个指针的协调使用，完成 I/O 接口的主要控制功能，实现服务器机端与存储系统端的异步运行。

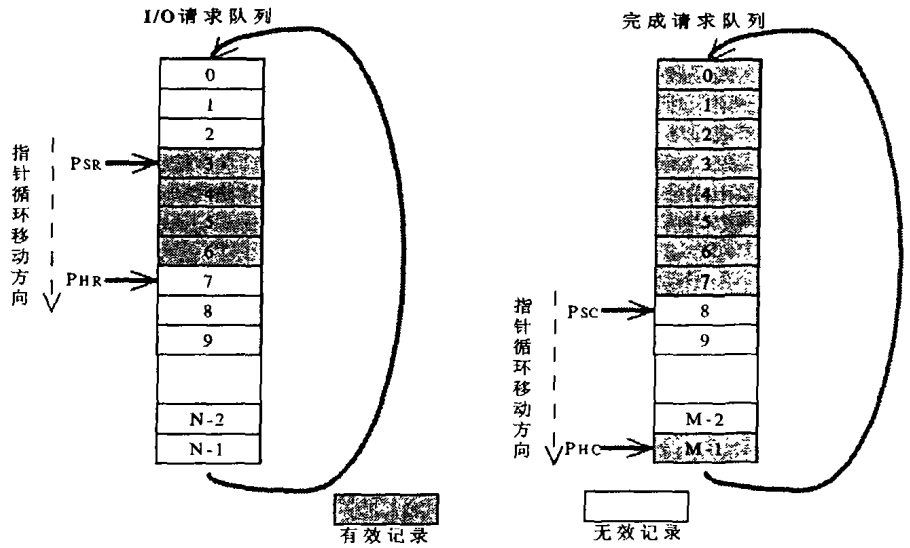

图 5　I/O 请求队列和完成请求队列模型

服务器的进程向存储系统发出一个 I/O 请求，将调用 I/O 接口卡的驱动程序，驱动程序检测 P_{HR} 指向的 I/O 请求队列记录的状态字段（参考图 4），如果状态显示该记录无效，则将 I/O 请求的信息填充到记录的各字段中，并最后设置状态字段为有效。将 I/O 请求加入 I/O 请求队列后，驱动程序修改 P_{HR} 指向下一条记录并返回调用进程，进程不等待 I/O 请求的完成，而执行后续的工作。在 I/O 接口卡上运行的程序持续检测 P_{HR} 指向的 I/O 请求记录的状态字段，如果该记录有效，则将这条 I/O 请求记录取出提交给存储系统，并设置记录状态字段无效，调整 P_{SR} 指向下一条记录。存储系统采用 DMA 完成 I/O 请求所要求的存储介质和服务器缓冲区间数据的传输。完成数据传输的 I/O 请求的信息被加入到 P_{SC} 指向的完成请求队列的记录中，在此之前同样要检测相应的状态字段，加入记录后要修改状态字段和 P_{SC} 指针。服务器端的内核查询线程在定时启动后，将从 P_{HC} 指向的完成请求队列记录开始处理累积的已经完成数据传输的 I/O 请求，处理每一个记录前都要检测状态字段，并在处理后修改状态字段和 P_{HC} 指针。在上述的过程中，对每一条记录操作前都首先要检测记录的状态字段，如果 P_{HR} 和 P_{SC} 指向的记录状态为有效时，说明队列饱和；P_{HC} 和 P_{SR} 指向的记录状态为无效时，说明队列为空；程序要等待上述状态改变后，才能继续操作相应的记录，完成该记录的操作之后必须修改记录的状态字段。通过对状态字段的严格操作，确保服务器端和存储系统端不会同时对同一条记录进行操作，保证了队列中数据的一致性。

确定 I/O 请求队列的长度是一个比较复杂的问题。如果存储系统处理 I/O 请求的速率小于服务器产生 I/O 请求的速率，不论 I/O 请求队列的长短，服务器进程迟早都会出现阻塞并等待 P_{HR} 指向的记录的状态字段变为无效，此时异步 I/O 请求处理又变成同步处理，这显然有悖于我们的初衷。采用异步方式处理 I/O 请求的一个前提就是储系统处理 I/O 请求的能力足够强大，允许服务器产生尽可能多的 I/O 请求。此时 I/O 请求队列类似于排队理论中的 G/G/1/N 队列，I/O 请求队列的长度 N 与服务器产生 I/O 请求的时间间隔的分布和存储系统处理 I/O 请求时间的分布有关，由于确定

上述的分布的方法较为复杂，因此可以根据系统的实际运行效果，动态优化队列的长度。如果 I/O 内存足够大，应使 I/O 请求队列尽可能长些，减少由于服务器产生 I/O 请求的时间间隔的分布不均匀(突发访问)而造成队列饱和的可能性。确定完成请求队列的长度比较简单，可以选择与 I/O 请求队列相同的长度。

5 测试与结论

在异步方式和同步方式处理 I/O 请求的对比测试中，测试平台采用 IBM342 PC 服务器，配置两颗 1.26GHz 的 Pentium III CPU 和 4GB 内存，运行 LINUX-2.4.18 操作系统。I/O 接口卡通过 PCI-X 总线连接到服务器，并通过两条 Ultra 320 SCSI 总线连接两个 U320 磁盘阵列，每个阵列包含 10 个 Ultra 320 SCSI 磁盘，这样的存储系统配置提供了足够的 I/O 请求处理能力。为使测试具有可比性，在同步 I/O 请求处理方式中采用两块 LSI 53C1030 HBA 与两个磁盘阵列相连。

存储系统中所有的磁盘配置成 RAID0，使数据散布在所有的磁盘上；为尽可能使每一个 I/O 请求所访问的数据块包含在同一个磁盘上，我们设置 RAID0 的条纹单元 8192KB。服务器上运行的测试程序随机访问存储系统的线性存储空间，使 I/O 请求均匀分布到每个磁盘上。在同步 I/O 请求处理方式下，进程将等待 I/O 请求的完成，我们通过增加发出 I/O 请求进程的数量使服务器处理 I/O 请求的能力达到饱和；在异步 I/O 请求处理方式下，进程不等待 I/O 请求的完成，我们通过增加进程产生 I/O 请求的速率使服务器处理 I/O 请求的能力达到饱和。在异步 I/O 请求处理方式下 I/O 请求队列和完成请求队列的长度均设置为 5000。表 1 是针对读取数据块的不同大小，服务器采用异步和同步方式处理 I/O 请求的饱和速率。

表 1 同步和异步方式下处理 I/O 请求的饱和速率

读取数据块(kB)	1	2	4	8	16	32	64	128	256	512	1024
同步处理(IO/sec)	4286	4157	3978	3912	3823	2664	2484	1815	963	459	244
异步处理(IO/sec)	5383	5234	5023	4896	4732	3145	2830	1882	975	465	246
增加比例(%)	25.60	25.91	26.27	25.15	23.78	18.06	13.93	3.69	1.25	1.3	0.82

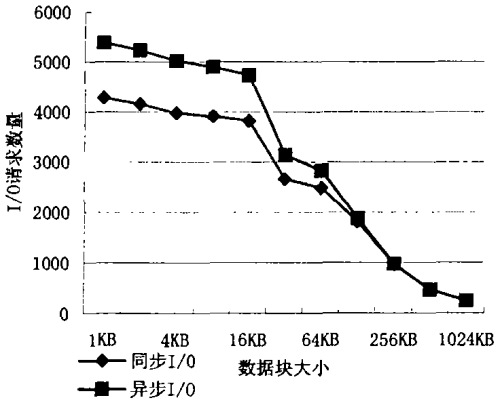

图 6 同步和异步方式下处理 I/O 请求的饱和速率

从上面的图表可以看出，当 I/O 请求的数据块较小时(1kB～16kB)，系统在单位时间内完成的 I/O 请求数量较多，异步处理的效果明显优于同步处理，单位时间内处理的 I/O 请求增加了 20%以上。但随着数据块的增大(大于 64kB)，系统完成 I/O 请求数量的迅速减少，异步处理的结果几乎与同步处理相同。

当 I/O 请求的数据块较小时，存储系统处理每个 I/O 请求的时间较短，存储系统在单位时间内可完成较多的 I/O 请求，整个系统处于 I/O 请求密集型应用的状态，此时异步处理 I/O 请求的优越性就表现出来：减少了由于进程阻塞引起的无效计算时间和无效空闲时间，以及大量中断响应和处理累积的时间消耗。

当 I/O 请求的数据块较大时，存储系统在单位时间内只能完成较少的 I/O 请求，虽然此时数据吞吐量较大，但这并不是我们所希望优化的流媒体服务器应用的主要工作特征，脱离了我们实现异步 I/O 请求处理的前提设想，因此同步和异步处理的效果基本相同。

在本文中，我们分析了流媒体服务器的工作原理，将流媒体服务定义为 I/O 请求密集型应用，提出通过优化系统处理 I/O 请求的能力，提高流媒体服务器支持客户端并发访问的能力；实现了服务器进程的异步读数据操作，减少由于进程阻塞造成的无效计算和空闲时间对服务器 I/O 处理能力的影响；采用定时查询方式代替中断方式，批量处理完成数据传输的 I/O 请求，减少了大量中断响应和处理产生的系统消耗；采用双队列结构实现服务器与存储系统的接口，实现服务器与存储系统的异步操作，简化了接口电路的设计。上述工作的目的都是为了提高以流媒体服务为代表的 I/O 请求密集型应用的性能，提高系统处理 I/O 请求的能力，通过对比测试，我们达到了预期的结果。

参 考 文 献

1 Ozden B, Rastogi R, Silberschatz A. Research issues in multimedia storage servers. ACM Computing Surveys, 1995, 27(4): 617～620

2 Anastasiadis S V, Sevcik K C, Stumm M. Server-Based Smoothing of Variable Bit-Rate Streams. In: ACM Multimedia Conf. Ottawa, Oct. 2001. 147～158

3 Ganger G, Patt Y. The Process-Flow Model: Examining I/O Performance from the System's Point of View. Sigmetrics, 1993. 86～97

4 Fricker C, Jaibi M R. Monotonicity and stability for periodic polling models. Queueing Systems, 1994, 15: 211～238

5 Dan A, Sitaram D, Shahabuddin P. Scheduling Policies for an On-Demand Video Server with Batching. In: Proc. of ACM Multimedia'94, Oct. 1994. 391～398

6 Rubini A, Corbet J. Linux Device Drivers, Second Edition. O'Reilly&Associates, Inc. 2001

7 毛德操，胡希明. LINUX 内核源代码情景分析. 浙江大学出版，2001