视频流。换句话说，像这样优化磁盘请求增加了服务器可以同时传送的电影数。回环末尾的富余时间还可以用来服务可能存在的任何非实时请求。

如果服务器有太多的视频流，偶尔也会出现当要求从磁盘的边缘部分读取帧时错过了最终时限的情况。但是，只要错过最终时限的情况足够稀少，以此换取同时处理更多的视频流还是可以容忍的。注意，要紧的是读取的视频流的数目，每个视频流有两个或更多个客户并不影响磁盘性能或调度。

为了保持输出给客户的数据流运行流畅，在服务器中采用双缓冲是必要的。在第i个回环期间，使用一组缓冲区，每个视频流一个缓冲区。在这个回环结束的时候，输出进程或进程组被解除阻塞并且被告知传输第i帧。与此同时，新的请求进来请求每部电影的第i+2帧（每部电影或许有一个磁盘线程和一个输出线程）。这些请求必须用第二组缓冲区来满足，因为第一组缓冲区仍然在忙碌中。当第i+3个回环开始的时候，第一组缓冲区已经空闲，可以重新用来读取第i+3帧。

我们一直在假设每一帧只有一个回环，这一限制并不是严格必需的。每一帧也可以有两个回环，以便减少所需缓冲区空间的数量，其代价是磁盘操作的次数增加了一倍。类似地，每一回环可以从磁盘中读取两帧（假设一对帧连续地存放在磁盘上）。这一设计将磁盘操作的数目减少了一半，其代价是所需缓冲区空间的数量增加了一倍。依靠相对可利用率、性能和内存费用与磁盘I/O的对比，可以计算并使用优化策略。

7.9.2 动态磁盘调度

在上面的例子中，我们假设所有的视频流具有相同的分辨率、帧率和其他特性，现在让我们放弃这一假设。不同的电影现在可能具有不同的数据率，所以不可能每33.3ms有一个回环并且为每个视频流读取一帧。对磁盘的请求或多或少是随机到来的。

每一读请求需要指定要读的是哪一磁盘块，另外还要指定什么时间需要该磁盘块，也就是最终时限。为简单起见，我们假设对于每次请求实际的服务时间是相同的（尽管这肯定是不真实的）。以这种方法，我们可以从每次请求减去固定的服务时间，得到请求能够发出并且还能满足最终时限的最近的时间。因为磁盘调度程序所关心的是对请求进行调度的最终时限，所以这样做使模型更为简洁。

当系统启动的时候，还没有挂起的磁盘请求。当第一个请求到来的时候，它立即得到服务。当第一次寻道发生的时候，其他请求可能到来，所以当第一次请求结束的时候，磁盘驱动器可能要选择下一次处理哪个请求。某个请求被选中并开始得到处理。当该请求结束的时候，再一次有一组可能的请求：它们是第一次没有被选中的请求和第二个请求正在被处理的时候新到来的请求。一般而言，只要一个磁盘请求完成，磁盘驱动器就有若干组挂起的请求，必须从中做出选择。问题是："使用什么算法选择下一个要服务的请求？"

在选择下一个磁盘请求时，有两个因素起着重要的作用：最终时限和柱面。从性能的观点来看，保持请求存放在柱面上并且使用电梯算法可以将总寻道时间最小化，但是可能导致存放在边缘柱面上的请求错过其最终时限。从实时的观点来看，将请求按照最终时限排序并且以最终时限的顺序对它们进行处理，可以将错过最终时限的机会最小化，但是可能增加总寻道时间。

使用scan-EDF算法（scan-EDF algorithm）（Reddy和Wyllie，1994）可以将这两个因素结合起来。这一算法的思想是，将最终时限比较接近的请求收集在一起分成若干批，并且以柱面的顺序对其进行处理。作为一个例子，我们考虑图7-27当t = 700时的情形。磁盘驱动器知道它有11个挂起的请求，这些请求具有不同的最终时限和不同的柱面。它可以决定将具有最早的最终时限的5个请求视为一批，将它们按照柱面号排序，并且使用电梯算法以柱面顺序对它们进行服务。于是，顺序将是110、330、440、676和680。只要每个请求能够在其最终时限之前完成，这些请求就可以安全地重新排列，从而将所需的总寻道时间最小化。

如果不同的视频流具有不同的数据率，那么当一个新的客户出现时将引起一个严重的问题：该客户是否应该被接纳？如果接纳该客户会导致其他的视频流频繁地错过它们的最终时限，那么答案可能就是否。存在两种方法计算是否接纳新的客户。一种方法是假设每个客户平均地需要某些数量的资源，如磁盘带宽、内存缓冲区、CPU时间等。如果剩下的每一资源对于一个平均的顾客来说都是足够的，则接纳新的客户。