8.5 顺序访问

使用下表中的参数来采集数据。

I/O 支援功能	种类	方式	单次 I/O 请求量
off	r	seq	4、8、16、32、64、128、256、512、1024、2048、4096
off	w	seq	4、8、16、32、64、128、256、512、1024、2048、4096

然后运行程序，结果如下。在笔者的计算机上，*dev*sdb 是 HDD，本次实验将在该设备中未曾使用过的 *dev*sdb5 分区上进行。

```
$ sudo su
# for I in 4 8 16 32 64 128 256 512 1024 2048 4096 ; do ↵
time ./io devsdb5 off r seq $i ; done

real   0m1.972s
( 略 )
real   0m1.311s
( 略 )
real   0m0.916s
( 略 )
real   0m0.695s
( 略 )
real   0m0.595s
( 略 )
real   0m0.537s
( 略 )
real   0m0.527s
( 略 )
real   0m0.509s
( 略 )
real   0m0.539s
( 略 )
real   0m0.499s
```

```
( 略 )
real    0m0.531s
#
```

将读取时与写入时测出的数据分别制作成图表，如图 8-8 与图 8-9 所示。纵轴上的吞吐量通过实验结果中的程序运行时间除以总的 I/O 请求量（64 MB）得到。

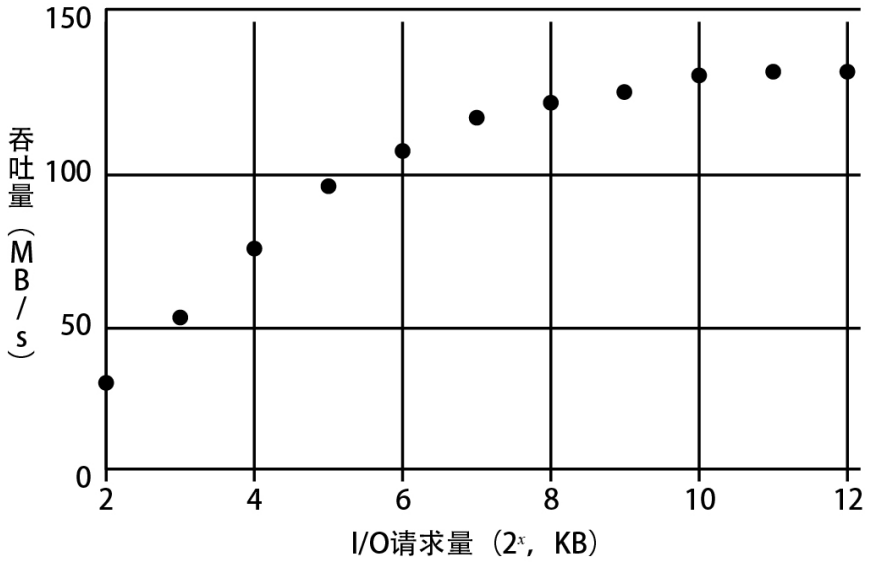

图 8-8　HDD 的顺序读取性能（禁用 I/O 支援功能）

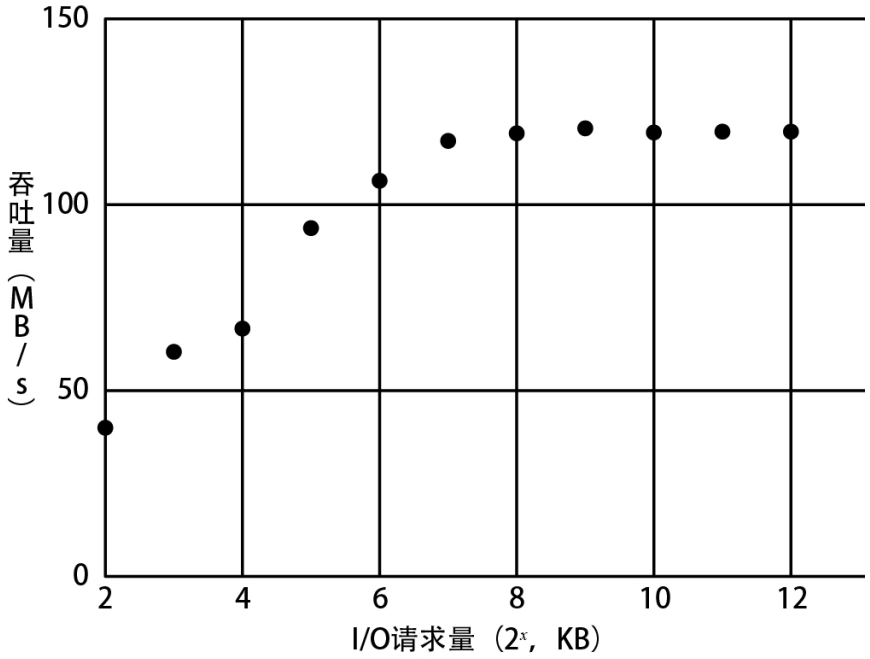

图 8-9　HDD 的顺序写入性能（禁用 I/O 支援功能）

可以看到，不管是读取还是写入，随着单次 I/O 请求量增大，吞吐量都有所提升。

但在单次 I/O 请求量达到 1 MB 后，性能就已经到达峰值了。这个值就是该 HDD 单次访问允许的数据量上限，同时也是该 HDD 设备的性能上限。顺便一提，如果单次 I/O 请求量超过数据访问的上限值，在通用块层中会将该访问划分为多次请求来执行。

8.6 随机访问

使用下表中的参数来采集数据。

I/O 支援功能	种类	方式	单次 I/O 请求量
off	r	rand	4、8、16、32、64、128、256、512、1024、2048、4096
off	w	rand	4、8、16、32、64、128、256、512、1024、2048、4096

将实验结果添加到顺序访问时的图表（图 8-8 与图 8-9）中进行比较，如图 8-10 与图 8-11 所示。

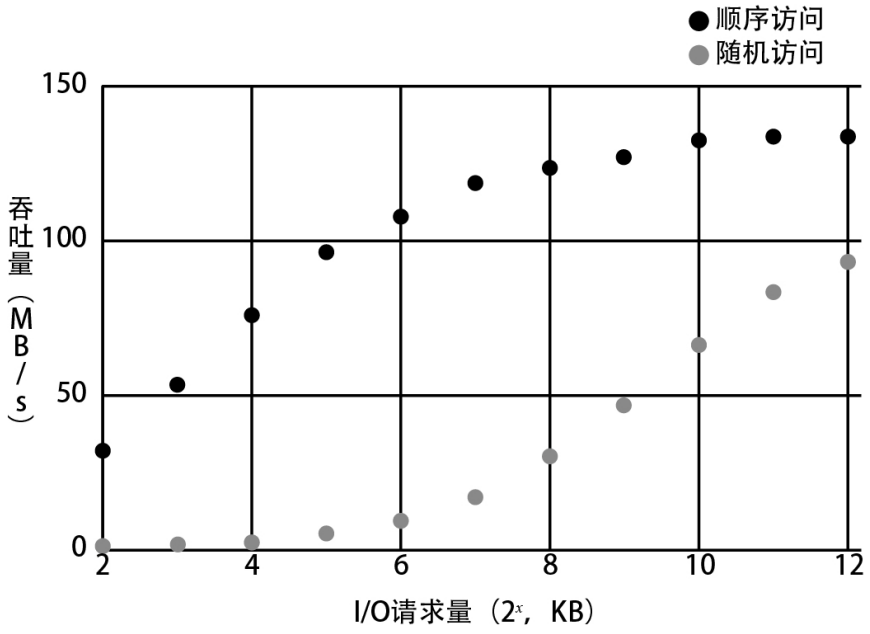

图 8-10 HDD 的读取性能（禁用 I/O 支援功能）

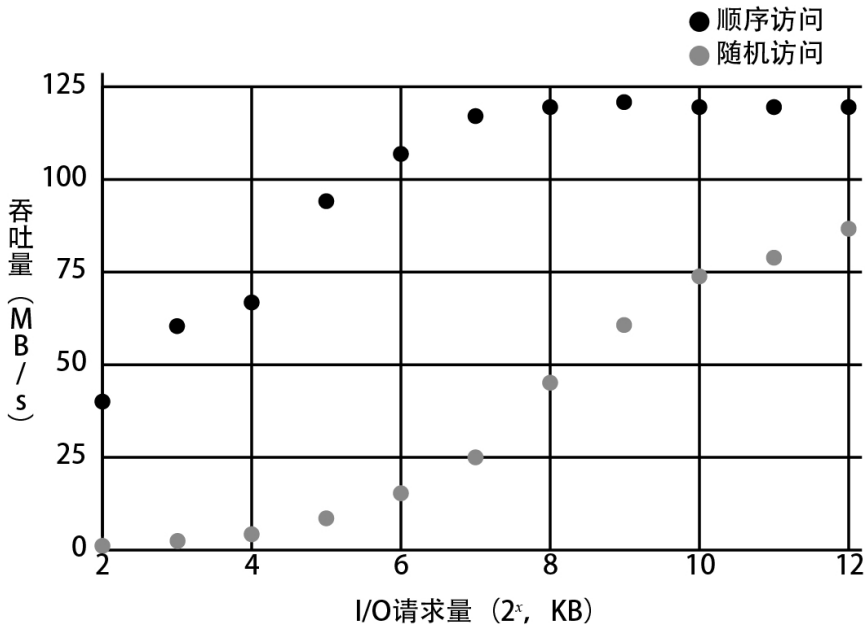

图 8-11　HDD 的写入性能（禁用 I/O 支援功能）

可以看到，不管是读取还是写入，随机访问的性能都比顺序访问差。特别是在 I/O 请求量较小时，差距更加明显。虽然随着 I/O 请求量变大，程序整体上等待访问完成的时间在减少，吞吐量也得以提升，但还是比不上顺序访问的性能。

8.7 通用块层

如第 7 章所述，Linux 中将 HDD 和 SSD 这类可以随机访问、并且能以一定的大小（在 HDD 与 SSD 中是扇区）访问的设备统一归类为块设备。

块设备可以通过设备文件直接访问，也可以通过在其上构建的文件系统来间接访问。大部分软件采用的是后一种方式。

由于各种块设备通用的处理有很多，所以这些处理并不会在设备各自的驱动程序中实现，而是被集成到内核中名为**通用块层**的组件上来实现，如图 8-12 所示。

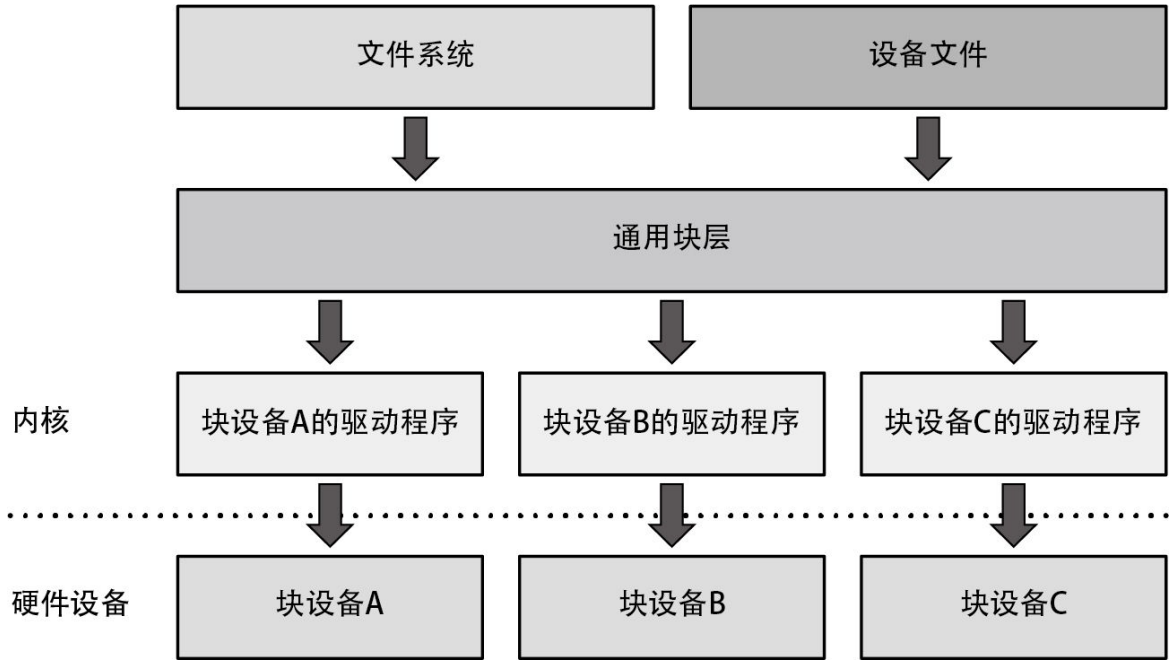

图 8-12 将各种块设备通用的处理集成到通用块层上

接下来将逐一介绍通用块层中的 I/O 调度器与预读功能。

8.8 I/O 调度器

通用块层中的 I/O **调度器**会将访问块设备的请求积攒一定时间，并在向设备驱动程序发出 I/O 请求前对这些请求进行如下加工，以提高 I/O 的性能。

- **合并：将访问连续扇区的多个 I/O 请求合并为一个请求**
- **排序：按照扇区的序列号对访问不连续的扇区的多个 I/O 请求进行排序**

也存在先排序再执行合并的情况，那样可以更大程度地提高 I/O 性能。I/O 调度器的运作方式如图 8-13 所示。

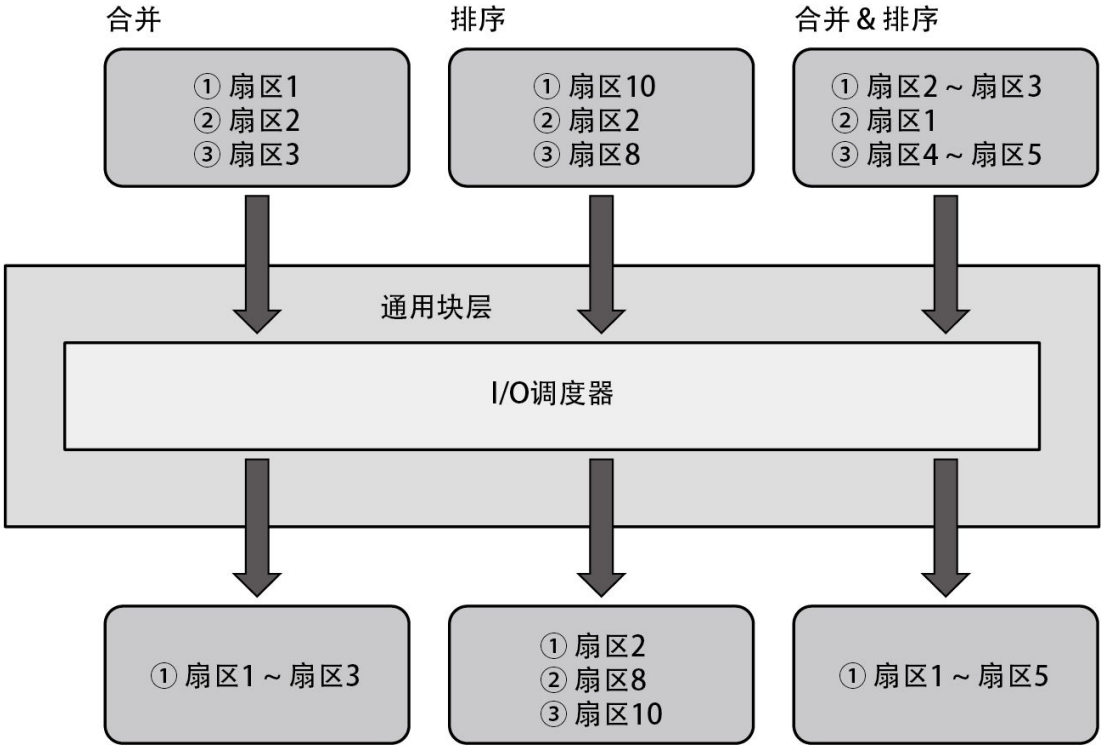

图 8-13 I/O 调度器的运作方式

有了 I/O 调度器，即便用户程序的开发人员不太了解块设备的性能特性，也能够在一定程度上发挥块设备的性能。

8.9 预读

如第 6 章所述，在程序访问数据时具有空间局部性这一特征。通用块层中的**预读**（read-ahead）机制就是利用这一特征来提升性能的。

当程序访问了外部存储器上的某个区域后，很有可能继续访问紧跟在其后的区域，而预读机制正是基于这样的推测，预先读取那些接下来可能被访问的区域，如图 8-14 所示。

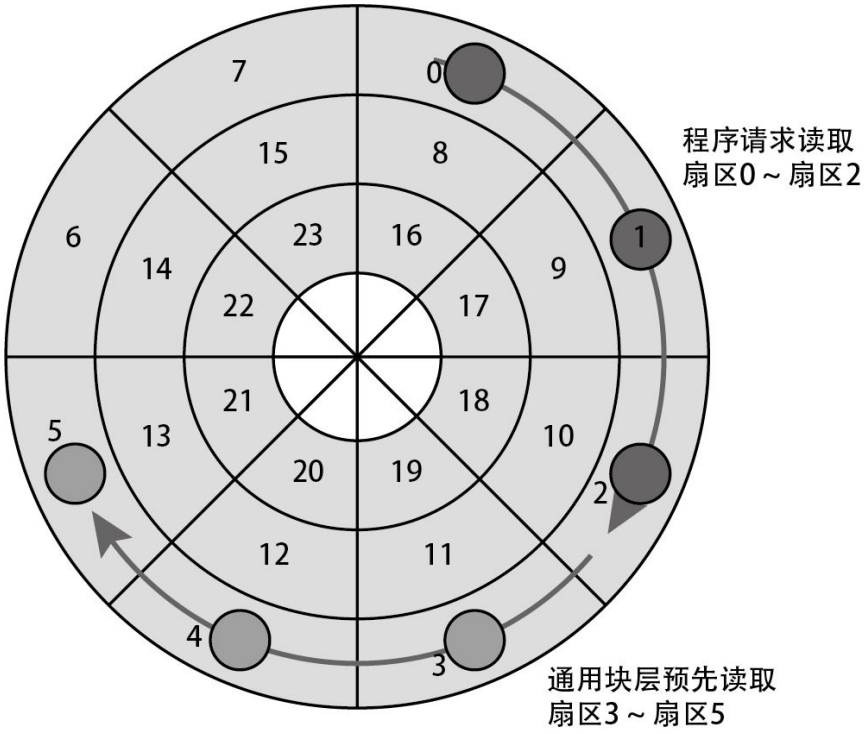

图 8-14　预读（1）

如果之后与推测的一样，程序申请读取后面那部分区域，就可以省略该读取请求的 I/O 处理，因为这些数据已经被预先读取出来了（图 8-15）。

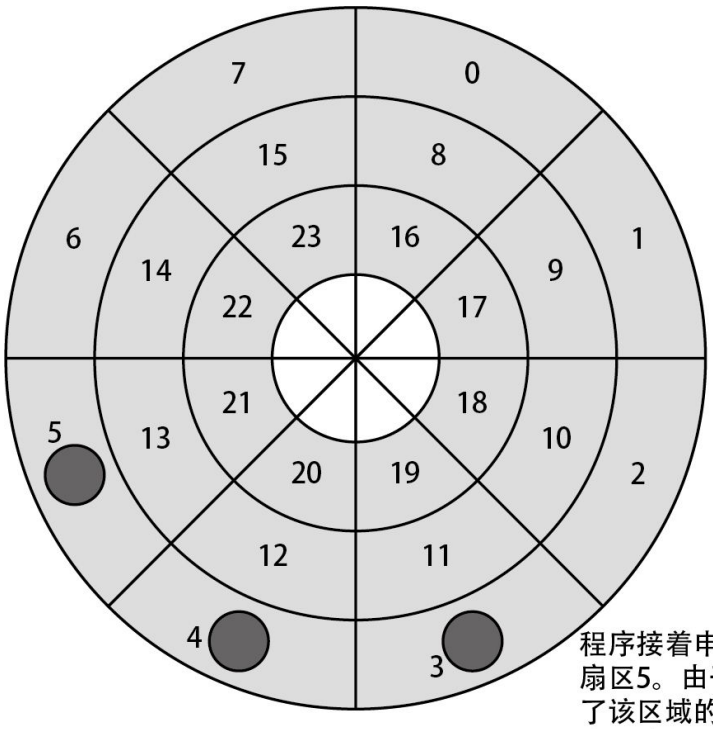

图 8-15　预读（2）

该机制有望提高顺序访问的性能。即使推测的访问没有到来，也只需丢弃预读的数据即可。

8.10 实验

接下来，我们测试一下启用 I/O 支援功能时的 I/O 性能，并与禁用时的性能进行比较。

● 顺序访问

使用下表中的参数来进行数据采集。

I/O 支援功能	种类	方式	单次 I/O 请求量
on	r	seq	4、8、16、32、64、128、256、512、1024、2048、4096
on	w	seq	4、8、16、32、64、128、256、512、1024、2048、4096

将实验结果与禁用 I/O 支援功能时的数据一同制作成图表进行比较，如图 8-16 与图 8-17 所示。

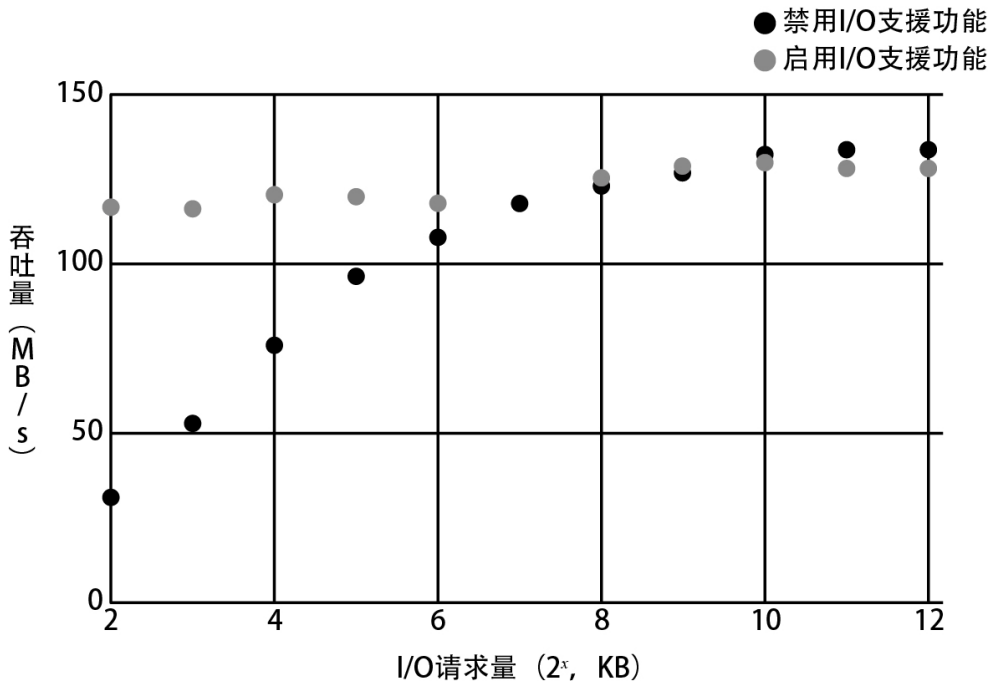

图 8-16　HDD 的顺序读取性能

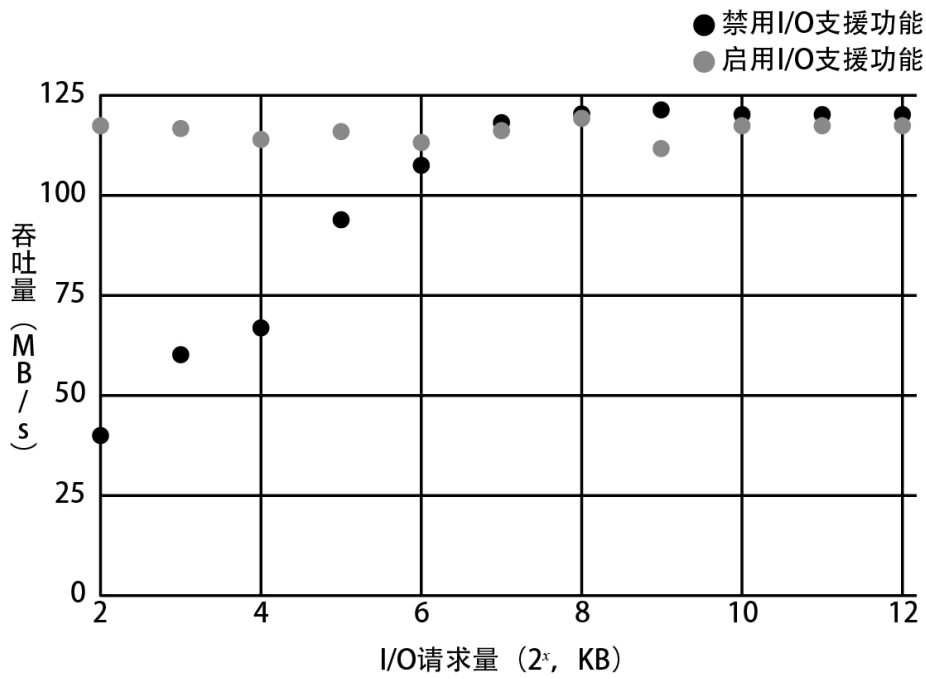

图 8-17　HDD 的顺序写入性能

禁用 I/O 支援功能启用 I/O 支援功能

在启用 I/O 支援功能后，读取和写入两种操作都能从 I/O 请求量较小时就把吞吐量发挥到接近 HDD 的性能极限。读取的性能提升主要得益于预读机制。在 io 程序的运行过程中，通过在另一个终端查看 iostat -x 命令的执行结果，就能知道性能提升的效果如何。首先来看一下当禁用 I/O 支援功能且 I/O 请求量为 4 KB 时的情况。

```
$ iostat -x -p sdb 1
( 略 )
Device      rrqm/s wrqm/s      r/s      w/s    rkB/s  wkB/s ↲
avgrq-sz  avgqu-sz  await  r_await  w_await    svctm  %util
sdb           0.00   0.00     0.00     0.00     0.00   0.00 ↲
0.00          0.00   0.00     0.00     0.00     0.00   0.00
( 略 )
sdb           0.00   0.00  2274.00     0.00  9096.00   0.00 ↲
8.00          0.26   0.12     0.12     0.00     0.12  26.40
                                                      ←①
( 略 )
sdb           0.00   0.00  8487.00     0.00 33948.00   0.00 ↲
8.00          0.98   0.11     0.11     0.00     0.11  97.60
( 略 )
sdb           0.00   0.00  5643.00     0.00 23524.00   0.00 ↲
8.34          0.66   0.12     0.12     0.00     0.12  66.00
                                                      ←②
( 略 )
sdb           0.00   0.00     0.00     0.00     0.00   0.00 ↲
0.00          0.00   0.00     0.00     0.00     0.00   0.00
( 略 )
```

sdb 在①到②处执行 I/O 处理。可以看到，完成 64 MB 数据的读取操作需要耗费接近 3 秒。

接下来看一下启用 I/O 支援功能时的情况（I/O 请求量为 4 KB）。

```
$ iostat -x -p sdb 1
( 略 )
Device     rrqm/s  wrqm/s      r/s     w/s    rkB/s   wkB/s ↲
avgrq-sz avgqu-sz   await  r_await  w_await   svctm   %util
sdb           0.00    0.00     0.00    0.00     0.00    0.00 ↲
0.00          0.00    0.00     0.00    0.00     0.00    0.40
( 略 )
sdb           0.00    0.00   536.00    0.00 66808.00    0.00 ↲
249.28        1.06    1.98     1.98    0.00     1.05   56.40
                                                      ←①
( 略 )
```

```
sdb              0.00    0.00     0.00    0.00     0.00    0.00 ↲
0.00             0.00    0.00     0.00    0.00     0.00    0.00
...
```

sdb 在①处执行 I/O 处理。可以看到，只需不到 1 秒就能读取完 64 MB 的数据。这是因为，在初次访问数据时，紧接在其后的数据也一起被预先读取出来了。在预读机制下，在读取后续的数据时，由于后续的数据已经存在于内存上了，所以处理所花费的时间也得以缩短。

此时的合并处理又如何呢？可以通过 rrqm/s 字段得到读取处理的合并数，在本实验中该值为 0，代表这里并没有执行合并。为什么会这样呢？这是因为实验程序中的读取处理必须同步地从外部存储器读取数据，完成后再紧接着执行下一次读取操作，这令 I/O 调度器没有运行的机会。

虽然在本书中没有涉及，但实际上 I/O 调度器只有在多个进程并行读取时或者在异步 I/O 等无须等待读取完成的 I/O 上才能发挥作用。

启用 I/O 资源后的写入速度提升，则归功于 I/O 调度器的合并处理。在合并处理中，程序发出的零零碎碎的 I/O 申请将被全部合并并积攒起来，直到请求量达到 HDD 单次可以处理的上限后再实际发出 I/O 请求。与之前一样，让我们通过统计数据来看一下合并的实际运作方式。

首先是禁用 I/O 支援功能时的情况。

```
$ iostat -x -p sdb 1
( 略)
Device     rrqm/s wrqm/s      r/s      w/s   rkB/s    wkB/s ↲
avgrq-sz avgqu-sz  await  r_await  w_await   svctm    %util
sdb           0.00   0.00     0.00     0.00    0.00     0.00 ↲
0.00          0.00   0.00     0.00     0.00    0.00     0.00
( 略 )
sdb           0.00   0.00     0.00  4966.00    0.00 19864.00 ↲
8.00          0.46   0.09     0.00     0.09    0.09    46.00
                                                      ←①
( 略 )
sdb           0.00   0.00     0.00 10207.00    0.00 40828.00 ↲
8.00          0.96   0.09     0.00     0.09    0.09    96.60
                                                      ←②
( 略 )
sdb           0.00   0.00    20.00  1211.00 1032.00  4844.00 ↲
```

```
9.55          0.15    0.12      1.80      0.09     0.11     14.00
                                                              ←③
( 略 )
sdb           0.00    0.00      0.00      0.00     0.00      0.00 ↲
0.00          0.00    0.00      0.00      0.00     0.00      0.00
( 略 )
```

外部存储器在①到③处执行 I/O 处理。接下来是启用 I/O 支援功能时的情况。

```
$ iostat -x -p sdb 1
( 略)
Device     rrqm/s   wrqm/s     r/s     w/s   rkB/s    wkB/s ↲
avgrq-sz avgqu-sz    await r_await w_await   svctm    %util
sdb           0.00     0.00    0.00    0.00    0.00     0.00 ↲
0.00          0.00     0.00    0.00    0.00    0.00     0.00
( 略 )
sdb           0.00 16320.00    0.00   18.00    0.00 18432.00 ↲
2048.00       9.02    78.67    0.00   78.67    9.33    16.80
                                                        ←①
( 略 )
sdb           0.00     0.00   20.00   46.00 1032.00 47104.00 ↲
1458.67       8.32   241.39    0.08  346.00    5.64    37.20
                                                        ←②
...
sdb           0.00     0.00    0.00    0.00    0.00     0.00 ↲
0.00          0.00     0.00    0.00    0.00    0.00     0.00
( 略 )
```

外部存储器在①与②处执行 I/O 处理。可以看到，在①处，表示合并操作的 wrqm/s 字段的值增加了。

● 随机访问

使用下表中的参数来采集数据。

I/O 支援功能	种类	方式	单次 I/O 请求量
on	r	rand	4、8、16、32、64、128、256、512、1024、2048、4096

I/O 支援功能	种类	方式	单次 I/O 请求量
on	w	rand	4、8、16、32、64、128、256、512、1024、2048、4096

将实验结果与顺序访问时的数据一同制作成图表进行比较，如图 8-18 与图 8-19 所示。

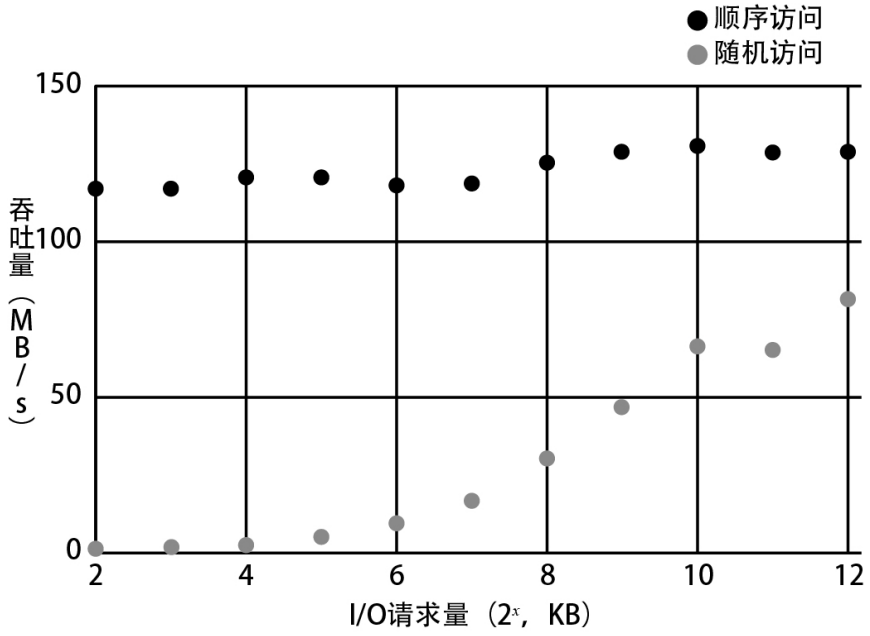

图 8-18　HDD 的读取性能（启用 I/O 支援功能）