第3章 chapter 3

Benchmark 工具

测量性能，还需要使用适当的基准测试（Benchmark）工具。本章将会介绍一些有用的 Benchmark 工具。衡量性能的明智做法是使用好的 Benchmark 工具。有很多优秀的工具可用，它们一般都具有以下全部或部分功能：生成负载、监控性能、监控系统使用率、报告。

Benchmark 是一个特定工作负载的模型，可能接近或不接近运行在系统上的工作负载。如果系统拥有良好的 Linpack 得分，它仍有可能不是理想的文件服务器。记住，Benchmark 不能模拟终端用户出现的不可预测的反应。Benchmark 无法告诉你，一旦用户访问它们的数据或备份文件，服务器是如何运转的。一般情况下，当在任何系统上执行 Benchmark 时，应该遵守以下规则，如表 3-1 所示。

表 3-1　Benchmark 使用规则（标准）

使用规则（标准）	注　　释
隔离 Benchmark 系统	如果 Benchmark 测试一个系统，那么最重要的是要从任何尽可能多的其他负载中隔离它。打开会话，运行 top 命令，可以极大地影响 Benchmark 的结果
平均结果	即使尝试隔离 Benchmark 系统，在使用 Benchmark 测试的时候也可能有未知因素会影响系统性能。运行任何 Benchmark 至少 3 次，计算平均结果，这是很好的做法，以避免一次性事件影响整个分析
模拟预期的工作量	所有 Benchmark 都有不同的配置选项，可使用这些选项定制 Benchmark 监控的系统工作负载。如果应用程序依赖低磁盘延迟，那么提高 CPU 的性能是毫无用处的
为服务器工作负载使用 Benchmark	服务器系统拥有明显的特点，使它们与典型的桌面计算机非常不同，即使它们通过 IBM System x 平台共享很多技术用于桌面计算机。生成多个线程是为了利用系统的 SMP（对称多处理）功能，为了模拟真正的多用户环境服务器 Benchmark。PC 启动一个 Web 浏览器可能要比高端服务器快，服务器启动 1000 个 Web 浏览器比 PC 更快

在下面的章节中，我们基于这些标准选择一些工具，如表 3-2 所示。

表 3-2 Benchmark 使用规则涉及的工具

涉及的工具	注 释
在 Linux 上工作	Linux 是 Benchmark 的目标
在所有硬件平台上工作	例如，IBM 公司提供了 3 种不同的硬件平台（假设 IBM System p 和 IBM System i 的硬件技术都基于 IBM POWER 架构），重点是选择一个 Benchmark 用于所有架构上，而无须大的移植工作
开源	Linux 在多种平台上运行，所以若源代码不可用，则二进制文件可能无法使用
报告功能	报告功能将大大减少性能分析的工作
易于使用	一个易于使用的工具
广泛使用	可以找到很多关于广泛使用的工具的信息
积极维护	废弃的旧工具可能无法遵循最新的规范和技术。它可能产生错误的结果
好的文档	当执行 Benchmark 的时候，必须了解工具。文档将帮助你熟悉工具。在决定使用某些工具之前，通过查看概念、设计、细节，有助于评估哪个工具可以满足自己的需求

基准测试是“测量或评估的标准”。一台计算机的基准测试就是一个典型的计算机程序，执行严格定义的一组操作，返回某种形式的结果，度量、描述、测试计算机如何执行。计算机基准测试的度量标准通常是测量“速度”（完成工作量有多快）；或是“吞吐量”（每个单位时间内可以完成多少单位的工作量）。可以在多台计算机上运行相同的计算机基准测试进行比较。

理想情况下，对于系统的最佳对比测试，可以使用你自己的应用程序，用你自己的工作负载。遗憾的是，对于不同的系统，得到可靠的、可重复可比较的测量是不切实际的。问题可能包括：生成一个好的测试案例、保密问题、难以确保的对比条件、时间、金钱等限制。

不妨考虑一下使用标准化的基准测试作为一个参考点。理想情况下，一个标准化的基准测试是可移植的，并且可能已经在你感兴趣的平台上运行。但是，在考虑结果之前，你需要确保自己了解相关应用/计算的需求和基准测试要测量什么。

注意：一个标准化基准测试可以作为参考点。然而，当选择供应商或产品的时候，一般不主张标准化基准测试取代实际应用程序的基准测试。

本章涉及的 Benchmark 工具及功能，如表 3-3 所示。

表 3-3 本章涉及的 Benchmark 工具及功能

工 具	最有用的工具功能
UnixBench	CPU Benchmark 工具
STREAM	测量内存带宽的 Benchmark 工具
Bonnie++	测试磁盘驱动器性能的 Benchmark 工具

3.1 UNIXBench

UNIXBench源于1995年，基线系统是George，一个工作站：SPARCstation 20-61，128MB RAM，Solaris2.3，此系统的指数值被设定为10，所以，如果一个系统的最后结果分数为520，意思是指此系统比基线系统运行快52倍。

UNIXBench也支持多CPU系统的测试，默认的行为是测试两次：第一次是一个进程的测试；第二次是N份测试，N等于CPU个数。这样的设计是为了①测试系统的单任务性能；②测试系统的多任务性能；③测试系统并行处理的能力。

UNIXBench是一个基于系统的基准测试工具，不单纯是CPU内存或者磁盘测试工具。测试结果不仅取决于硬件，也取决于系统、开发库，甚至编译器。UNIXBench的测试方法，如表3-4所示。

表3-4 UNIXBench的测试方法

测试方法	注释
Dhrystone测试	测试聚焦在字符串处理，没有浮点运算操作。这个测试用于测试链接器编译、代码优化、内存缓存、等待状态、整数数据类型等，硬件和软件设计都会非常大地影响测试结果
Whetstone测试	这项测试项目用于测试浮点运算效率和速度。这项测试项目包含若干个科学计算的典型性能模块，包含大量的C语言函数——sin、cos、sqrt、exp和日志，以及使用整数和浮点的数学操作，也包含数组访问、条件分支和过程调用
Execl Throughput测试	(execl吞吐，这里的execl是类UNIX系统非常重要的函数，非办公软件的excel) 这项测试测试每秒execl函数的调用次数。execl是exec函数家族的一部分，使用新的图形处理代替当前的图形处理，它有许多命令和前端的execve()函数命令非常相似
File Copy测试	这项测试衡量文件数据从一个文件被传输到另外一个文件，使用大量的缓存，包括文件的读、写、复制测试，测试指标是一定时间内(默认是10s)被重写、读、复制的字符数量
Pipe Throughput(管道吞吐)测试	pipe是简单的进程之间的通信。管道吞吐测试是测试在一秒钟一个进程写512比特到一个管道中并且读回来的次数。管道吞吐测试和实际编程有差距
Pipe-based Context Switching(基于管道的上下文交互)测试	这项测试衡量两个进程通过管道交换和整数倍地增加吞吐的次数。基于管道的上下文切换和真实程序很类似。测试程序产生一个双向管道通信的子线程
Process Creation(进程创建)测试	这项测试衡量一个进程能产生子线程并且立即退出的次数。新进程真的创建进程阻塞和内存占用，所以测试程序直接使用内存带宽。这项测试用于典型的比较大量的操作系统进程创建操作
Shell Scripts测试	shell脚本测试用于衡量在一分钟内，一个进程可以启动并停止shell脚本的次数，通常会测试1,2,3,4,8个shell脚本的共同副本，shell脚本是一套转化数据文件的脚本

续表

测 试 方 法	注　　释
System Call Overhead (系统调用消耗)测试	这项测试衡量进入和离开系统内核的消耗,例如,系统调用的消耗。程序简单重复地执行 getpid 调用(返回调用的进程 id)。消耗的指标是调用进入和离开内核的执行时间

3.1.1 安装与运行

1. 下载

```
https://github.com/kdlucas/byte-unixbench/archive/v5.1.3.tar.gz
```

2. 修改 Makefile 交叉编译

```
#CC=gcc
CC =arm-linux-gnueabihf-gcc
make
```

3. 修改 Run

将 main()函数中的 preChecks();注释掉,因为其中有 system("make all");。

3.1.2 Run 的用法

```
Run [ -q | -v ] [-i <n>] [-c <n>[-c <n>...]] [test ...]
```

其中,Run 的参数,如表 3-5 所示。

表 3-5 Run 的参数

参　　数	注　　释
-q	不显示测试过程
-v	显示测试过程
-i<n>	执行次数,最低 3 次,默认 10 次
-c<n>	每次测试并行 *n* 个副本(并行任务)

备注:-c 选项可以用来执行多次,如:

```
Run -c 1 -c 4
```

表示执行两次,第一次执行单个副本的测试任务,第二次执行 4 个副本的测试任务。

对于多 CPU 系统的性能测试策略,需要统计单任务、多任务及其并行的性能增强。以 4 个 CPU 的 PC 为例,需要测试两次,4 个 CPU 就是要并行执行 4 个副本,如:

```
Run -q -c 1 -c 4
```

的测试结果是：单个并行任务的得分为171.3，4个并行任务的得分为395.7。对比测试时需要关注这个值。测试结果对比，如表3-6所示。

表3-6 测试结果对比

测试项目	项目说明	基准线
Dhrystone 2 using register variables	测试string handing	116700.0lps
Double-Precision Whetstone	测试浮点数操作的速度和效率	55.0MWIPS
Execl Throughput	测试考查每秒钟可以执行的execl系统调用的次数	43.0lps
File Copy 1024 bufsize 2000 maxblocks	测试从一个文件向另外一个文件传输数据的速率	3960.0KBps
File Copy 256 bufsize 500 maxblocks	测试从一个文件向另外一个文件传输数据的速率	1655.0KBps
File Read 4096 bufsize 8000 maxblocks	测试从一个文件向另外一个文件传输数据的速率	5800.0KBps
Pipe-based Context Switching	测试两个进程(每秒钟)通过一个管道交换一个不断增长的整数的次数	12440.0lps
Pipe Throughput	一秒钟内一个进程可以向一个管道写512B数据然后再读回的次数	4000.0lps
Process Creation	测试每秒钟一个进程可以创建子进程然后收回子进程的次数(子进程一定立即退出)	126.0lps
Shell Scripts(8 concurrent)	测试一秒钟内一个进程可以并发地开始一个shell脚本的n个副本的次数，n的取值一般为1,2,4,8	42.4lpm
System Call Overhead	测试进入和离开操作系统内核的代价，即一次系统调用的代价	6.0lpm

3.2 STREAM

STREAM Benchmark是一个简单的合成基准测试程序，可持续地测量内存带宽(Mb/s)和相应地为简单的向量内核计算速率。通常使用由John McCalpin创建和维护的STREAM Benchmark。

1. 为什么要关心内存

计算机CPU的速度越来越快，已经远远超过计算机内存系统。照这样发展，越来越多的程序性能会受到系统内存带宽的限制，而不是CPU的计算性能。尽管这看起来很简单，但确定可持续内存带宽的架构因素多而复杂，且显得很微妙。供应商很少能提供足够的有效的硬件细节，从而可以准确地估计可持续的内存带宽。

比如一个极端的例子，目前有些高端的机器运行简单的算术运算，内核因为缓存不足，操作数只占它额定峰值速度的4%～5%，这意味着它95%～96%的时间是空闲的，

在等待未命中的缓存得到满足。

STREAM Benchmark就是特别针对数据集远远大于可用的缓存的系统而设计的，因此其结果更能反映大型向量风格的应用程序的性能。

STREAM Benchmark使用FORTRAN 77和C的相应版本编写。

访问http://www.cs.virginia.edu/stream/FTP可下载相应的源代码。

2. 单处理器运行

如果希望在一个单处理器上运行STREAM，则这件事情很容易做到，需要FORTRAN或C的主STREAM代码，并且需要一个计时器代码。对于UNIX/Linux系统，计时器代码执行(second_wall.c)正常工作。有些系统提供更高精度的计时器，可检查UNIX/Linux文档获知这些信息。

注意，一般拒绝基于“CPU计时器”的新的测量，因为在很多系统上这个功能精度不高并且系统不准确。

有很多STREAM版本的代码使用FORTRAN和C编写。最新的版本被命名为stream.f和stream.c。

3. 简单的编译指令

首先需要得到C代码或是FORTRAN代码，加上外部计时器代码。

在大多数Linux系统上，可以编译一个符合标准的(单CPU)STREAM版本，使用简单的命令：

```
gcc -O stream.c -o stream
```

在FORTRAN中，相应的编译有：

```
gcc -c mysecond.c
g77 -O stream.f mysecond.o -o stream
```

或有时为：

```
gcc -c -DUNDERSCORE mysecond.c
g77 -O stream.f mysecond.o -o stream
```

4. 相关优化结果的注意事项

STREAM在运行时具有一定的灵活性(下面有详细讨论)。需要的数组要远远大于所使用的最大缓存。默认数组的大小为200万个元素(这是足够大的，可以满足缓存达到4MB的系统的运行，例如大多数当前计算机系统)，并且默认偏移量为0个元素。

许多人都通过设置offset变量(OFFSET在stream.c的第59行定义)得到改进的结果。这里很难给出指导，因为每个计算机家族对于内存冲突都有略微不同的处理细节，试验是比较好的方法。

5. 多处理器运行

如果想在多处理器上运行 STREAM，那么情况就不那么简单了。

首先，需要解决如何并行运行代码。这里有一些选择：OpenMP、MPI。

(1) OpenMP。标准的 STREAM 代码现在都包含了 OpenMP 指令。如果有一个支持 OpenMP 的编译器，那么所有你要做的就是找出用来启用 OpenMP 编译的标志，以及需要什么环境变量控制使用的处理器/线程的数量。

一些最新版本的 gcc 支持 OpenMP，下面是在 Red Hat Enterprise Linux 8.x 中编译并运行：

```
[root@zgh stream]#gcc -fopenmp stream.c -o stream
[root@zgh stream]#./stream
```

检查输出，看是否会出现类似下面的行：

```
Number of Threads requested =16
Number of Threads counted =16
```

在 POWER 或 PowerPC 系统上运行 IBM 编译器，过程可能看起来像：

```
xlc -qsmp=omp -O stream.c -o stream
export OMP_NUM_THREADS=4
./stream
```

或者在 FORTRAN 中：

```
xlc -qsmp=omp -c mysecond.c
xlf_r -qsmp=omp -O stream.f mysecond.o -o stream
export OMP_NUM_THREADS=4
./stream
```

遗憾的是，基本上所有编译器的命令行选项都有所不同。

(2) MPI。如果想得到多处理器的结果，但是你有一个集群或是没有 OpenMP 编译器，那么可以考虑 STREAM 的 MPI 版本(stream_mpi.f 在 Versions 子目录中)，这就要求安装 MPI 支持(像是 MPICH)，这是一个非常大的话题，这里不过多讨论。

目前 MPI 的结果不是很多，主要是因为结果是显而易见的，除非有些事是错误的。一个集群的性能将会是一个节点的数倍。STREAM 不会尝试测量集群网络的性能，它只用来帮助控制计时器。

一个使用 STREAM 的 MPI 版本的 Benchmark 是 HPC Challenge Benchmark，其针对大型超级计算集群。该网站是 http://icl.cs.utk.edu/hpcc。

有些人做到了他们自己的 pthreads 实现，但这些不能证实。

6. 调整大小的问题

STREAM 的目的是测量主内存的带宽。当然，它也可以用来测量缓存的带宽。

STREAM 的一般规则是，每个数组必须至少是运行中所使用的所有末级缓存总和的 4 倍，或是 100 万个元素，通常以较大的为准。

因此，对于一个拥有 256KB L2 Cache 的单处理器机器(例如老式的 Pentium Ⅲ)，每个数组至少需要 128 000 个元素。标准 2 000 000 个元素的测试大小，适用于 4MB L2 Cache 的系统。我们需要结果具有可比性，一旦每个数组的大小变得明显大于缓存大小，但是由于存在一定的差异(通常与 TLB 的范围有关)，因此在不同大小的性能中有相对较小的差异。即使机器的缓存使用 100 万个元素，也仅需要 22MB，因此，即使在 32MB 的机器上，它应该也是可行的。

若在 16 个 CPU 上自动并行运行，每个都有 8MB L2 Cache，则大小必须增加到至少 $N=64\ 000\ 000$，这将需要大量的内存(大约 1.5GB)。如果得到的比这大得多，将需要编译 64 位寻址，并且一旦 N 超过 20 亿，将需要确保使用 64 位整数值。

可以编辑 stream.c 文件，调整数组大小。注意，从上面的注释信息中，也可以得到很好的帮助信息。

7. 调整数组的帮助信息

调整数组的帮助信息，如图 3-1 所示。

```
/*-----------------------------------------------------------------------
---
 * INSTRUCTIONS:
 *
 *      1) STREAM requires different amounts of memory to run on differe
nt
 *           systems, depending on both the system cache size(s) and the
 *           granularity of the system timer.
 *     You should adjust the value of 'STREAM_ARRAY_SIZE' (below)
 *           to meet *both* of the following criteria:
 *       (a) Each array must be at least 4 times the size of the
 *           available cache memory. I don't worry about the difference
 *           between 10^6 and 2^20, so in practice the minimum array size
 *           is about 3.8 times the cache size.
 *           Example 1: One Xeon E3 with 8 MB L3 cache
 *               STREAM_ARRAY_SIZE should be >= 4 million, giving
 *               an array size of 30.5 MB and a total memory requirement
 *               of 91.5 MB.
 *           Example 2: Two Xeon E5's with 20 MB L3 cache each (using
OpenMP)
 *               STREAM_ARRAY_SIZE should be >= 20 million, giving
 *               an array size of 153 MB and a total memory requirement
 *               of 458 MB.
 *       (b) The size should be large enough so that the 'timing calibration'
 *           output by the program is at least 20 clock-ticks.
```

图 3-1 调整数组的帮助信息

8. 统计字节数和 FLOPS

对于一个类似 STREAM 的 Benchmark，至少有 3 种不同的方法来统计字节，这 3 种方法都是最常用的，分别为 bcopy、STREAM、hardware。

bcopy 统计有多少字节从内存中的一个位置移到另一个位置。因此，如果计算机 1 秒在一个位置读取 100 万字节，并且将 100 万字节写入第二个位置，由此产生的“bcopy 带宽”被称为“每秒 1 MB”。

STREAM 统计用户要求读取多少字节加上用户要求写入多少字节。对于简单的 Copy 内核，这恰好是两次 bcopy 获得的数量。STREAM 为什么会这样做？因为 4 个内核中的 3 个指定运算，因此统计读入 CPU 的数据和从 CPU 写回的数据是有意义的。Copy 内核确实没有运算，但是我们选择与其他 3 个一样的方式统计字节。

hardware 可以移动与用户指定的不同的字节数。尤其是，大多数缓存系统在一个存储操作未命中数据缓存时执行所谓的“写分配”。在覆盖它之前，系统加载包含数据的缓存行。

9. 它为什么这样做

如此，在系统中将会有缓存行的单个副本，其是目前有效的所有字节。如果仅在缓存行中写入 1/2 的字节，例如，其结果可能是与来自内存的其他 1/2 的字节合并了。最好在缓存中做这件事情，因此数据首先被加载。

表 3-7 显示了在 STREAM 循环的每次迭代中统计了多少字节和 FLOPS。

表 3-7 每次迭代中统计的字节和 FLOPS

名 称	Kernel	Bytes/iter	FLOPS/iter
COPY：	a(i)＝b(i)	16	0
SCALE：	a(i)＝q＊b(i)	16	1
SUM：	a(i)＝b(i)＋c(i)	24	1
TRIAD：	a(i)＝b(i)＋q＊c(i)	24	2

4 个内核多次重复测试，并且选择 10 次试验的最好结果。

所以需要小心比较来自不同源的 MB/s。STREAM 总是使用相同的方法，并且始终仅统计请求用户程序加载或存储的字节，因此总是可以直接比较结果。

这些操作是长向量操作的“构建块”的典型操作。定义数组的大小以便每个数组大于测试机器的缓存，并且构建代码以便数据不重复使用。

STREAM 不建议“真正”的应用程序没有数据可以重复使用，而是从假设的机器的“高峰”性能中分离出内存子系统来测量。在现代的计算机上独立的内存系统是获得全速的一个非常大的部分。

4 个测试中的每个都增加了独立信息作为结果，如表 3-8 所示。

当浮点运算的成本可与内存访问相比时，STREAM 可以追溯到一个时间，从而 Copy 测试明显快于其他方法。在不再是任何机器都感兴趣的高性能计算的情况下，4 个 STREAM 带宽值通常彼此相当接近。

表 3-8 测试结果的独立信息

独立信息	注 释
Sum	添加一个第三方操作数,在向量机器上允许对多次加载/存储端口进行测试
Copy	在缺乏算术运算的情况下测量传输速率
Triad	允许链接/重叠/融合/乘/加运算
Scale	添加一个简单的算术运算

这里给出的所有结果都采用 64 位值。大多数结果是标准测试情况,没有指定数组偏移量的 200 万个元素向量。少数几个结果是通过运行许多不同数组偏移量的代码“优化的”,并选择最佳的结果。这是可以接受的,因为 STREAM 的目的是让用户使用标准的 FORTRAN 测量最佳可用带宽,不要在多种方式的探索中迷失,从而内存系统可以提供最佳性能。

3.3 Bonnie++

Bonnie++ 是一个 Benchmark 套件,主要执行一些简单的硬盘驱动器测试和文件系统性能测试。在运行它之后可以决定哪些测试是重要的,决定如何比较不同的系统。Bonnie++ 基于由 Tim Bray 编写的 Bonnie Benchmark。

Bonnie++ 的安装十分简单,下载 Bonnie++ 的源代码后,对其解压并进入源代码目录中,执行 configure(如下所示,使用--prefix 可以指定安装目录)、make、make install:

```
[root@zgh Software]#tar xf bonnie++-1.03a.tgz
[root@zgh Software]#cd bonnie++-1.03a
[root@zgh bonnie++-1.03a]#./configure --prefix=/usr/local/bonnie++
[root@zgh bonnie++-1.03a]#make
[root@zgh bonnie++-1.03a]#make install
```

注意:在 make 编译的时候有可能出现下面的错误提示,如图 3-2 所示。

```
zcav.cpp: In function ?.nt main(int, char**)?.
zcav.cpp:73: error: ?.trdup?.was not declared in this scope
zcav.cpp:112: error: ?.trcmp?.was not declared in this scope
make: *** [zcav] Error 1
```

图 3-2 make 编译错误提示

不过不用担心,可以编辑源代码目录中的 zcav.cpp,在其中添加如图 3-3 所示的内容解决此问题。

```
using namespace std;
#include <string.h>
#include <unistd.h>
#include <sys/time.h>
…
```

图 3-3 编辑 zcav.cpp

有许多不同类型的文件系统操作，不同的应用程序使用不同的系统。Bonnie++测试它们中的一些，并对每个测试给出每秒完成的工作量和花费CPU时间的百分比。较高数字的性能结果较好，较低的CPU使用率更好（注意：性能配置得分2000和90%的CPU使用率要比配置1000的性能和60%的CPU使用率更好）。程序的操作分为两部分：第一部分是测试I/O吞吐量，将其设计成模拟数据库类型的应用程序；第二部分是测试许多小文件的创建、读取和删除情况，类似于squid的使用模式。

通过-u和-d选项指定以root身份对/databak目录进行测试，如图3-4所示。

```
[root@zgh ~]# cd /usr/local/bonnie++/sbin
[root@zgh sbin]# ./bonnie++ -d /databak -u root
```

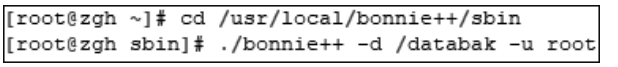

图3-4 指定以root身份对/databak目录进行测试

1. 测试详细信息

1）文件I/O测试

（1）连续输出，如表3-9所示。

表3-9 连续输出

选项/参数/功能	注　　释
每个字符	使用putc()标准输入输出宏写入文件。写足够小的循环，以适合任何适当的I-cache。CPU的开销需要完成标准输入输出代码加上操作系统的文件空间分配
块	使用write(2)创建文件。CPU的开销应该只是操作系统的文件空间分配
重写	文件的每个BUFSIZ通过read(2)读取，通过write(2)产生脏数据和重写，需要lseek(2)。因为没有空间分配并且I/O良好地局部化，所以应该测试文件系统高速缓存和数据传输速度的有效性

（2）连续输入，如表3-10所示。

表3-10 连续输入

选项/参数/功能	注　　释
每个字符	使用getc()标准输入输出宏读取文件。再次，内循环很小，应该只会练习标准输入输出和顺序输入
块	使用read(2)读取文件。这应该是一个非常纯正的连续输入性能的测试

（3）随机寻道。

这个测试并行运行SeekProcCount进程（默认3个），在bsd系统中使用random()，在sysV系统中使用drand48()，对文件指定位置共进行8000个lseek()。

在每种情况下，使用read(2)读取块。10%的情况下，它使用write(2)产生脏数据和写回。SeekProcCount进程背后的理念是确保总有一个寻道队列。

注意：对于任何 UNIX 文件系统，一旦缓存的效果失败，每秒有效的 lseek(2)调用的数量就会下降至近 30 个。

有一点要注意的是，RAID 中磁盘的数量会增加寻道的次数。RAID-1(镜像)的读取将增加一倍的寻道次数。RAID-0 的写入将导致 RAID-0 中磁盘的数量乘以写入的次数(前提是存在足够的寻道进程)。

2) 文件创建测试

创建的测试文件使用的文件名称由 7 位数字和一个随机数字(0～12)构成。对于顺序测试，文件名的随机字符遵循数字。对于随机测试，首先是随机字符。顺序测试涉及以数字顺序创建文件，然后以 readdir()顺序 stat()它们(即它们存储在目录中的顺序，很可能与它们被创建的顺序相同)，并以相同的顺序删除它们。对于随机测试，我们以出现的随机顺序创建文件到文件系统(文件中的最后 3 个字符是数字的顺序)。然后，我们 stat()随机文件(注意，存储目录的文件系统将会返回很好的结果，因为不是每个文件都将被 stat())。之后以随机顺序删除所有文件。如果指定的最大大小大于 0，则在每个文件被创建的时候将会在它里面写入一个随机数据，然后当文件被 stat()的时候，它的数据将被读取。

如果测试在不到 500ms 的时间内完成，那么输出将显示“++++”。这是因为由于舍入误差导致这样测试的结果不能被准确地计算，所以作者宁愿没有结果显示，也不愿显示一个错误的结果。

还可以使用-s 选项测试文件的大小。但是，Bonnie++一般要求指定的测试文件的大小至少为物理内存的 2 倍，如图 3-5 所示。

```
[root@zgh sbin]# ./bonnie++ -d /databak -u root -s 100 -m zgh.power.com
```

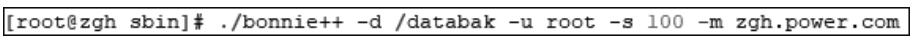

图 3-5 使用-s 选项测试文件的大小

Bonnie++可以以 CSV 电子表格的格式输出到标准输出。可以使用“-q”选项保持安静模式，那么可读的描述将输出到 stderr，因此重定向 stdout 到一个 CSV 文件，如图 3-6 所示。

```
[root@zgh sbin]# ./bonnie++ -d /databak -u root -q >/tmp/bonnie++.data
```

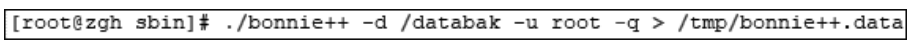

图 3-6 重定向 stdout 到一个 CSV 文件

如此，磁盘的性能就测试出来。

Sequential Output 下的 Per Char 的值用 putc 方式书写，因为 cache 的 line 总是大于 1B，所以不停地骚扰 CPU 执行 putc。CPU 使用率是 99%，写的速度是 87 MB/s。

Sequential Output 是按照 block 写的，明显 CPU 使用率就下来了，速度也上去了，大约是 140 MB/s。

Sequential Input 下的 Per Char 是指用 getc 的方式读文件，速度是 65.5MB/s，CPU 使用率是 92%。

Sequential Input 下的 block 是指按照 block 读文件，速度是 166MB/s，CPU 使用率是 11%。

2. Bonnie++ 支持的选项（表 3-11）

表 3-11 Bonnie++ 支持的选项

选项/参数	注　　释
-g	使用的组 ID，与-u 参数使用的：group 相同，只是以不同的方式指定其他程序的兼容性
-u	用户 ID。当以 root 运行的时候指定测试使用的 UID。不推荐使用 root，所以如果真想以 root 身份运行，就使用-u root。此外，如果想以指定组运行，就使用 user：group 格式。如果通过名称指定用户但是没有指定组，那么将选择用户的主组。如果通过数字指定用户但是没有指定组，那么组将是 nogroup
-x	运行测试的数量。如果想执行多个测试，那么这是很有用的，它将以 CSV 格式连续地转储输出，直到完成测试，或者它被杀死
-r	RAM 的大小，以 MB 为单位。如果指定了这个选项，则其他参数将被检查，以确保在大内存机器上它们是有意义的。在一般使用过程中不需要使用这个选项，因为它应该能够发现 RAM 大小。注意：如果指定大小为 0，那么将禁用所有的检查
-m	机器的名字，仅用于显示的目的
-n	文件创建测试的文件数量，这个测量为 1024 个文件的倍数，因为没有人想测试小于 1024 的文件，并且需要显示额外的空间。 如果指定为 0，那么测试将被跳过。 这个测试默认使用 0B 文件测试。如果要使用其他大小的文件，则可以指定 number：max：min：num-directories，max 是最大大小，min 是最小大小（如果没有指定，默认值都为 0）。如果指定了最小和最大大小，那么在 min..max 范围内每个文件将有一个随机的大小。如果指定了目录的数量，那么文件将平均分布到许多子目录中。 如果 max 为-1，那么将创建硬链接，而不是文件；如果 max 为-2，那么将创建软链接，而不是文件
-s	测量 I/O 性能的文件的大小，以 MB 为单位。如果大于 1GB，那么将用来存储多个文件数据，并且每个文件将高达 1GB 大小。参数可能包括以冒号分隔的 chunk 大小。测量的 chunk-size 以字节为单位，并且必须是 2 的幂，从 256 到 1048576。注意：如果在数字的末尾分别添加“g”或“k”，则可以指定 GB 大小或 KB 大小的 chunk-size。 如果指定的大小为 0，那么测试将被跳过
-d	用于测试的目录
-q	安静模式。如果指定了这个选项，那么一些额外的信息性消息将被抑制
-f	快速模式，跳过每字符 I/O 测试
-b	不写缓冲。每次写操作之后 fsync()
-p	服务信号量的进程数。其用来创建信号量用于同步多个 Bonnie++ 进程。通过-y 信号量告诉所有进程同时开始每个测试。通常使用-1 的值删除信号量
-y	每次测试前等待信号量

3. 测试原始硬盘驱动器的吞吐量的程序 ZCAV

现代硬盘驱动器具有恒定的转速，但是每个磁道有不同的扇区数(外磁道更长，拥有更多的扇区)，这被称为分区恒定角速度(ZCAV)。较外的磁道具有较高的数据传输速率，因为每个磁道有更多的扇区，这些磁道通常具有较低的磁道/扇区号。

这个程序测试硬盘驱动器的 Zcav 性能，以指定的次数读取整个数据。给定的文件名作为第一个参数，它可以通过“-”指定标准输入。该文件将以只读的方式打开，它将像往常一样操作/dev/sdX，这取决于是否使用 devfs(注意：Linux 以外的操作系统将具有不同的设备名称)，如下所示。

```
[root@zgh bonnie++]#./sbin/zcav -f /dev/sda
```

我们看到的输出结果也可以很容易地使用 gnuplot 图形化。

Zcav 支持的选项，如表 3-12 所示。

表 3-12 Zcav 支持的选项

选项/参数	注　　释
-b	从磁盘读取块的大小(默认为 100MB)
-c	读取整个磁盘的次数
-f	输入数据的文件名称。在最近 Glibc 配置的系统中不需要该选项，若没有-f 标志，也可以指定文件名
-u	用户 ID。当以 root 运行的时候指定测试使用的 UID。不推荐使用 root，所以如果真想以 root 身份运行，就使用-u root。此外，如果想以指定组运行，就使用 user：group 格式。如果通过名称指定用户但是没有指定组，那么将选择用户的主组；如果通过数字指定用户但是没有指定组，那么组将是 nogroup
-g	使用组 id，与-u 参数使用的：group 相同，只是以不同的方式指定其他程序的兼容性